

<b>Contents:</b>	<b>Page:</b>
1) Description of implementation	2
2) Preparation and prerequisites	3
3) Output from demonstrations	10
Registration process	10
Simulation between controller and device	13
4) Testing	20
5) Discussion of vulnerabilities identified and mitigations	25
6) Discussion of any omissions / Lack of mitigations	30
7) References	32
8) Appendices	34

Total discussion word count: 634

## **1. Description of implementation (264 words)**

The smart lock device prototype development demonstrates the implementation of the interaction and operation of a simulated door lock with a simulated controller. The interactions between the two simulated systems address issues such as latency, communication messages between client and server, low power consumption, security, reliability and temperature of the device. The prototype is consistent with PEP8 formatting and has considered input sanitisation through testing.

Initially, WebSocket technology was considered to support TCP protocol with low latency (Pavelić et al., 2018); however, through running the prototype through localhost, this was deemed unnecessary. Further consideration was taken in communicating the door lock to 'lock the device' (Figure 20) and 'unlock the device'. Following our planning and good practice of the dated theory 'separation of concerns' principle (De Win et al., 2002), the lock clients could be distributed, facilitating better usability by listening for commands from the controller/server unit.

Users must register the device manually, encrypt user-defined passwords and collect the MAC address. Consequently, in this simulation, an administrator was not needed; however, as planned for a web-based scenario, this could be applied for increased security.

To support the safety of the lock device and energy consumption, we adapted planning to manage the device's temperature. Higher temperatures could cause malfunction and higher energy consumption (Conex, 2021), so warning sensors would be attributed to the system should the temperature be in the range of 40-70 degrees celcius. In addition,

we could monitor the latency of communications between the devices, as illustrated in (Figure 25). Further discussion on vulnerabilities, mitigations and omissions are in sections 6 and 7.

## 2. Preparation and prerequisites

### 2.1. Environment

- 2.1.1. Two environments are required; one Controller and one Device
- 2.1.2. It's not a multiple server/client environment
- 2.1.3. Controller is also referred to as Server and Device refers to Client.
- 2.1.4. Run under Linux Ubuntu
  - 2.1.4.1. OS: Ubuntu 20.04.5 LTS
  - 2.1.4.2. Size: 1vCPU, 2GG memory and 50GB Disk

### 2.2. Software requirements, libraries and modules

#### 2.2.1. Python 3.8

```
root@ubuntu-s-1vcpu-2gb-intel-sgp1-01:~# python3 --version
Python 3.8.10
```

*Figure 1: Python version check*

#### 2.2.2. pwinput

- 2.2.2.1. It displays \*\*\*\* for password input rather than the actual input
- 2.2.2.2. Command to install: pip install pwinput

#### 2.2.3. pythonping

2.2.3.1. It is for checking the latency between the controller and its device

2.2.3.2. Command to install: pip install pythonping

#### 2.2.4. Socket

2.2.4.1. It is for allowing creation of servers and clients for communication with sockets between the controller and its device

2.2.4.2. Command to install: pip install sockets

#### 2.2.5. uuid

2.2.5.1. It is a library to generate random objects of 128 bits as ids.

2.2.5.2. Command to install: pip install uuid

#### 2.2.6. random

2.2.6.1. It is for generating random numbers

2.2.6.2. Command to install: pip install random

#### 2.2.7. ssl

2.2.7.1. It is designed to create a secure connection between client and server with encryption

2.2.7.2. Command to install: pip install ssl

#### 2.2.8. DateTime

2.2.8.1. Supplies classes to work with date and time.

2.2.8.2. Command to install: pip install DateTime

#### 2.2.9. time

2.2.9.1. provides local time from the number of seconds elapsed since the called local time.

2.2.9.2. Command to install: pip install python-time

#### 2.2.10. threading

2.2.10.1. It runs multiple threads simultaneously, such as tasks or function calls.

2.2.10.2. Command to install: pip install threaded

#### 2.2.11. re

2.2.11.1. It specifies a set of strings that matches a regular expression.

2.2.11.2. Command to install: pip install regex

#### 2.2.12. os

2.2.12.1. It provides functions for creating, changing, removing and identifying directories

2.2.12.2. Command to install: pip install os-sys

## 2.2.13. hashlib

2.2.13.1. It is for hashing in an encrypted format any raw message.

2.2.13.2. Command to install: pip install hashlib

## 2.2.14. pylint

2.2.14.1. It is used for testing by analysing code without running it to check for coding errors, smells and gives suggestions with a rating.

2.2.14.2. Command to install pip install pylint

## 2.2.15. Wireshark (This is optional)

2.2.15.1. It captures every packet getting in or out of a network interface.

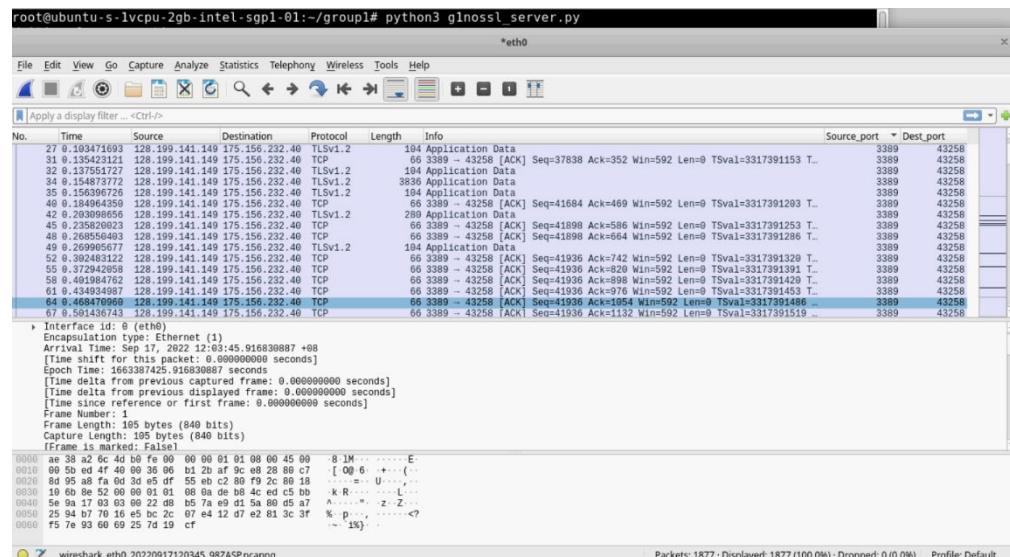


Figure 2: Wireshark report example from g1nossi\_server.py

## 2.2.16. Self-signed Certificate using openssl

2.2.16.1. Use it for securing the applications using TLS (Transport Layer Security) and SSL (Secure Sockets Layer) protocol.

```
root@ubuntu-s-1vcpu-2gb-intel-sg1-01:/# openssl version -a
OpenSSL 3.0.2 15 Mar 2022 (Library: OpenSSL 3.0.2 15 Mar 2022)
built on: Thu Sep 15 04:59:42 2022 UTC
platform: linux-x86_64
```

*Figure 3: Example of self-signed certificate*

2.2.16.2. Run the command in bold and the response in italic.

```
openssl genrsa -des3 -out server.orig.key 2048
Enter PEM pass phrase:: MyTesting123890

openssl rsa -in server.orig.key -out server.key
Enter pass phrase for server.orig.key:MyTesting123890

openssl req -new -key server.key -out server.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:SG
State or Province Name (full name) [Some-State]:Singapore
Locality Name (eg, city) []:Singapore
Organization Name (eg, company) [/Internet Widgits Pty Ltd]:Group1
Organizational Unit Name (eg, section) []:IT
Common Name (e.g. server FQDN or YOUR name) []:group1
Email Address []:ivancbt@yahoo.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:MyTesting123890
An optional company name []:Group1

openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

#### 2.2.16.3. Move the server.crt and server.key to server directory

#### 2.2.16.4. Rename server.crt to g1server.crt (Optional)

```
root@ubuntu-s-1vcpu-2gb-intel-sgpl-01:~/group1/server# ls -l
total 12
drwxr-xr-x 2 root root 4096 Sep 15 18:14 Backup
-rw-r--r-- 1 root root 1302 Sep 15 15:20 g1server.crt
-rw----- 1 root root 1708 Sep 15 15:16 server.key
```

*Figure 4: Moving certificates and keys*

#### 2.2.16.5. Since we are using self-signed certificates, copy

g1server.crt to the client

```
root@racknerd-d4d095:~/group1/client# ls -l
total 4
-rw-r--r-- 1 root root 1302 Sep 16 08:41 g1server.crt
root@racknerd-d4d095:~/group1/client#
```

*Figure 5: Copy server cert to client*

### 2.3. Configuration

#### 2.3.1. File Location

##### 2.3.1.1. Create a directory called: group1

##### 2.3.1.2. To store password file and python scripts

```
root@ubuntu-s-1vcpu-2gb-intel-sgpl-01:~/group1# ls -l
total 48
drwxr-xr-x 2 root root 4096 Sep 16 22:12 Backup
drwxr-xr-x 3 root root 4096 Sep 17 09:43 Backup2
-rw-r--r-- 1 root root 351 Sep 16 22:26 asmis.txt
-rw-r--r-- 1 root root 3624 Sep 16 22:29 glnoss1_client.py
-rw-r--r-- 1 root root 5930 Sep 16 15:20 glnoss1_server.py
-rw-r--r-- 1 root root 3621 Sep 16 22:29 glssl_client.py
-rw-r--r-- 1 root root 1326 Sep 16 22:30 glssl_reg_client.py
-rw-r--r-- 1 root root 3618 Sep 16 22:30 glssl_reg_server.py
-rw-r--r-- 1 root root 5926 Sep 16 22:30 glssl_server.py
drwxr-xr-x 3 root root 4096 Sep 15 18:55 server
root@ubuntu-s-1vcpu-2gb-intel-sgpl-01:~/group1#
```

*Figure 6: Creating directory and storing scripts*

2.3.1.3. For the Controller:

2.3.1.3.1. g1\_sslreg\_server.py: script to handle device registration

2.3.1.3.2. g1nossal\_server.py: no SSL is used

2.3.1.3.3. g1ssl\_server.py: Using SSL

2.3.1.4. For the client:

2.3.1.4.1. G1\_sslreg\_client.py: script to register a device with Controller

2.3.1.4.2. g1nossal\_server.py: no SSL is used

2.3.1.4.3. g1ssl\_server.py: Using SSL

2.3.2. Password file

2.3.2.1. Create an empty text file called asmis.txt and save it.

### 2.3.3. Certificate location

#### 2.3.3.1. Under the controller, create a server directory

```
drwxr-xr-x 2 root root 4096 Sep 15 18:14 Backup
-rw-r--r-- 1 root root 1302 Sep 15 15:20 glserver.crt
-rw----- 1 root root 1708 Sep 15 15:16 server.key
root@ubuntu-s-1vcpu-2gb-intel-sgpl-01:~/group1/server#
```

Figure 7: Creating a server directory

## 3. Output from demonstrations

### Registration process

#### 3.1. Registration process and current stage of asmis.txt file



The screenshot shows a text editor window with the file name \*asmis.txt and the path ~/group1. The file contains three lines of text:

```
1 door 07b6b098fc0cddd5b6854e07cf67d865eba4fec132f62738a0c43ee3d6cadfff CF-1A-FF-09-E6-DE
2 fan 07b6b098fc0cddd5b6854e07cf67d865eba4fec132f62738a0c43ee3d6cadfff CF-1A-FF-09-E6-D1
3 oven 07b6b098fc0cddd5b6854e07cf67d865eba4fec132f62738a0c43ee3d6cadfff D5-C9-83-46-10-35
```

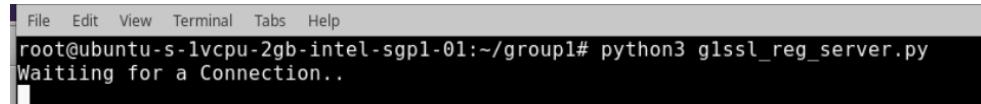
Figure 8: Username, encrypted passwords and MAC addresses

3.1.1. It contains the device name, encrypted password and MAC address.

3.1.2. Device: door and oven are the actual registered device. Fan is a device with an incorrect MAC address, for testing purposes.

#### 3.2. At the Server end:

3.2.1. Enter: g1ssl\_reg\_server.py



```

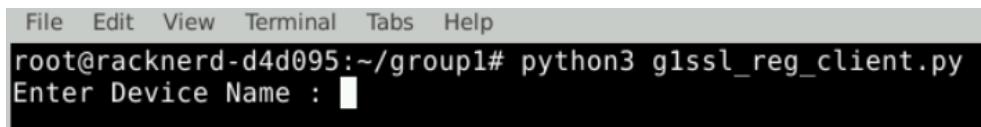
File Edit View Terminal Tabs Help
root@ubuntu-s-1vcpu-2gb-intel-sgpl-01:~/group1# python3 glssl_reg_server.py
Waiting for a Connection..

```

*Figure 8: Server connection*

### 3.3. At the Client end:

#### 3.3.1. Enter: g1ssl\_reg\_client.py



```

File Edit View Terminal Tabs Help
root@racknerd-d4d095:~/group1# python3 glssl_reg_client.py
Enter Device Name : iron

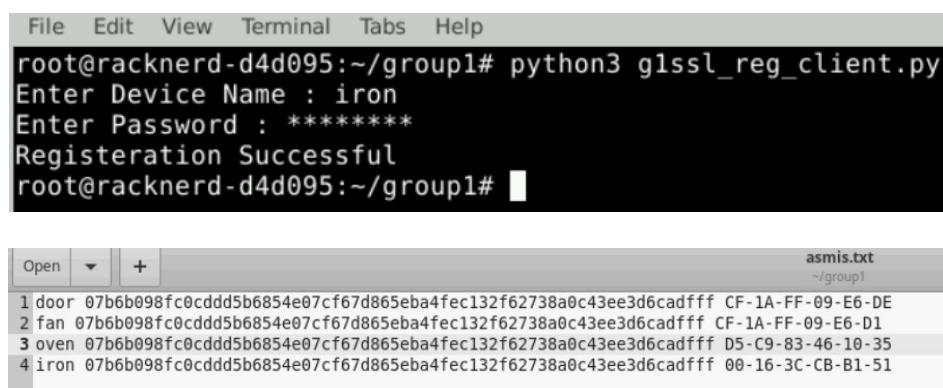
```

*Figure 9: Choosing a device name in client*

### 3.4. Communication between Server and Client

#### 3.4.1. To register a new device: iron

#### 3.4.2. Since it's a new device, registration is successful



```

File Edit View Terminal Tabs Help
root@racknerd-d4d095:~/group1# python3 glssl_reg_client.py
Enter Device Name : iron
Enter Password : *****
Registration Successful
root@racknerd-d4d095:~/group1#

```

asmis.txt  
~/group1

Open	+
1 door 07b6b098fc0cddd5b6854e07cf67d865eba4fec132f62738a0c43ee3d6cadfff CF-1A-FF-09-E6-DE	
2 fan 07b6b098fc0cddd5b6854e07cf67d865eba4fec132f62738a0c43ee3d6cadfff CF-1A-FF-09-E6-D1	
3 oven 07b6b098fc0cddd5b6854e07cf67d865eba4fec132f62738a0c43ee3d6cadfff D5-C9-83-46-10-35	
4 iron 07b6b098fc0cddd5b6854e07cf67d865eba4fec132f62738a0c43ee3d6cadfff 00-16-3C-CB-B1-51	

*Figure 10: Confirmation of communication*

Above: a new record "iron" is added to asmis.txt

### 3.4.3. Try to register the device: iron again

#### 3.4.3.1. It errors out since it's an existing device

#### 3.4.3.2. As seen from the Server end:

```
root@ubuntu-s-1vcpu-2gb-intel-sgpl-01:~/group1# python3 glssl_reg_server.py
Waiting for a Connection..
Connection Request: 1
Connection Request: 2
Device: iron is an existing device. Please register a new device
```

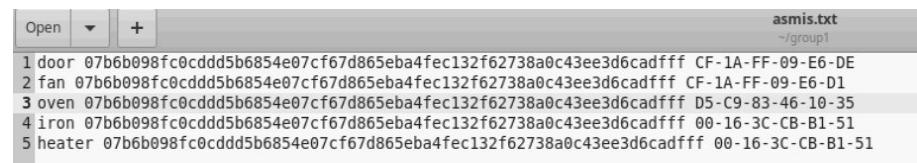
Figure 11: Checking for existing devices

### 3.4.4. Create a new device called: heater

#### 3.4.4.1. This device will be used for testing later as the MAC

address will be modified.

```
root@racknerd-d4d095:~/group1# python3 glssl_reg_client.py
Enter Device Name : heater
Enter Password : *****
Registration Successful
root@racknerd-d4d095:~/group1#
```

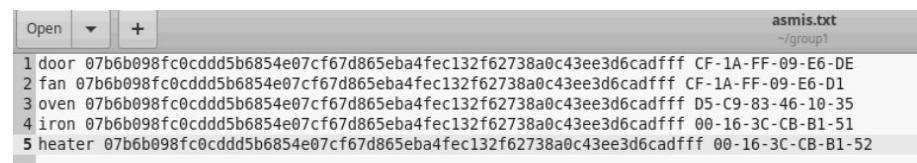


The screenshot shows a terminal window with the title 'asmis.txt' and the path '~/group1'. The window contains the following text:

```
1 door 07b6b098fc0ddd5b6854e07cf67d865eba4fec132f62738a0c43ee3d6cadfff CF-1A-FF-09-E6-DE
2 fan 07b6b098fc0ddd5b6854e07cf67d865eba4fec132f62738a0c43ee3d6cadfff CF-1A-FF-09-E6-D1
3 oven 07b6b098fc0ddd5b6854e07cf67d865eba4fec132f62738a0c43ee3d6cadfff D5-C9-83-46-10-35
4 iron 07b6b098fc0ddd5b6854e07cf67d865eba4fec132f62738a0c43ee3d6cadfff 00-16-3C-CB-B1-51
5 heater 07b6b098fc0ddd5b6854e07cf67d865eba4fec132f62738a0c43ee3d6cadfff 00-16-3C-CB-B1-51
```

Figure 12: MAC address confirmation

Note: the MAC address for iron and heater are the same, and for testing purposes, the heater's MAC address will be manually changed.



The screenshot shows a terminal window with the title 'asmis.txt' and the path '~/group1'. The window contains the following text:

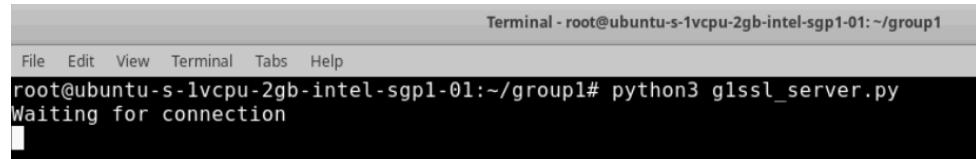
```
1 door 07b6b098fc0ddd5b6854e07cf67d865eba4fec132f62738a0c43ee3d6cadfff CF-1A-FF-09-E6-DE
2 fan 07b6b098fc0ddd5b6854e07cf67d865eba4fec132f62738a0c43ee3d6cadfff CF-1A-FF-09-E6-D1
3 oven 07b6b098fc0ddd5b6854e07cf67d865eba4fec132f62738a0c43ee3d6cadfff D5-C9-83-46-10-35
4 iron 07b6b098fc0ddd5b6854e07cf67d865eba4fec132f62738a0c43ee3d6cadfff 00-16-3C-CB-B1-51
5 heater 07b6b098fc0ddd5b6854e07cf67d865eba4fec132f62738a0c43ee3d6cadfff 00-16-3C-CB-B1-52
```

Figure 13: MAC address change

Note: heater's MAC address changed to 00-16-3C-CB-B1-52

## Simulation between Controller and Device

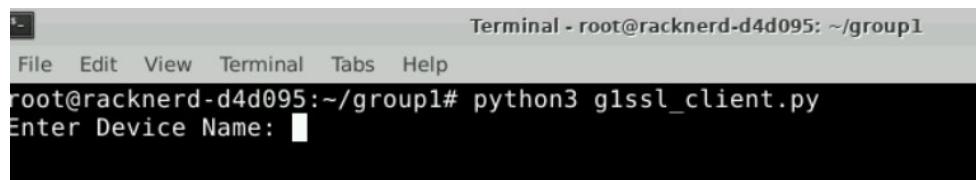
- 3.5. At the Server's end, enter: g1ssl\_server.py



```
Terminal - root@ubuntu-s-1vcpu-2gb-intel-sgp1-01:~/group1
File Edit View Terminal Tabs Help
root@ubuntu-s-1vcpu-2gb-intel-sgp1-01:~/group1# python3 g1ssl_server.py
Waiting for connection
```

*Figure 14: Server connection*

- 3.6. At the Client's end, enter: g1ssl\_client.py

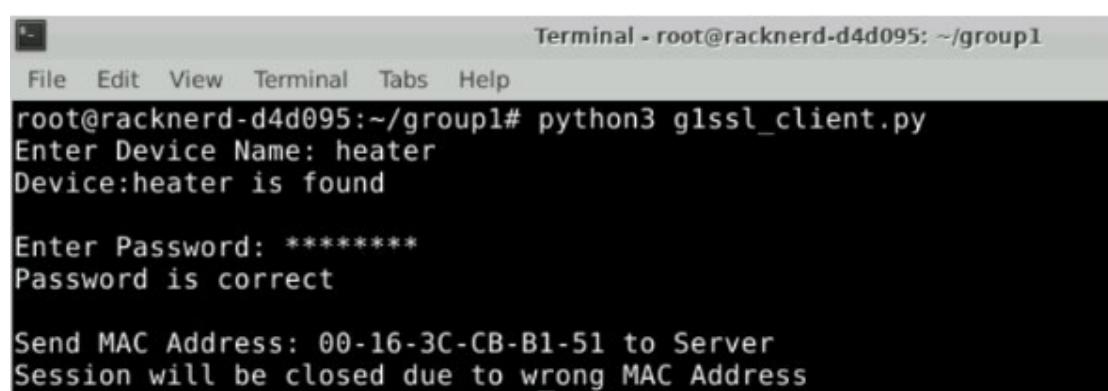


```
Terminal - root@racknerd-d4d095: ~/group1
File Edit View Terminal Tabs Help
root@racknerd-d4d095:~/group1# python3 g1ssl_client.py
Enter Device Name: 
```

*Figure 15: Client connection*

- 3.7. Communication between Server and Client:

- 3.7.1. Scenario 1: Device and Password are correct, but wrong MAC address



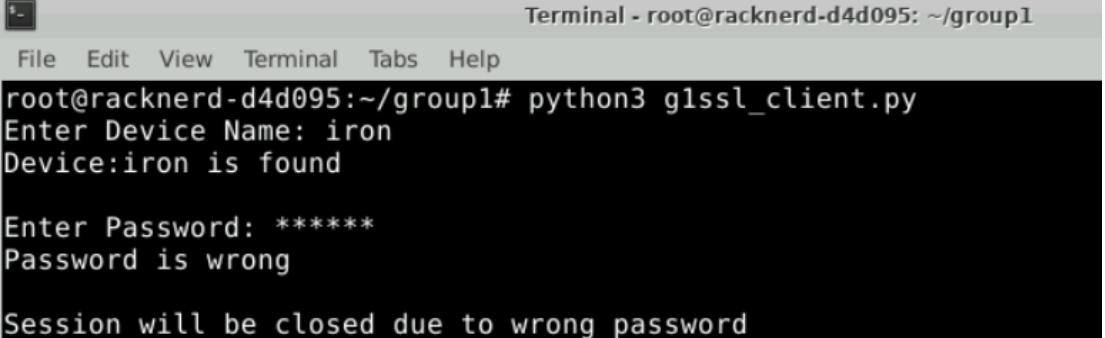
```
Terminal - root@racknerd-d4d095: ~/group1
File Edit View Terminal Tabs Help
root@racknerd-d4d095:~/group1# python3 g1ssl_client.py
Enter Device Name: heater
Device:heater is found

Enter Password: *****
Password is correct

Send MAC Address: 00-16-3C-CB-B1-51 to Server
Session will be closed due to wrong MAC Address
```

*Figure 16: Communicating errors*

### 3.7.2. Scenario 2: Device is correct but wrong password



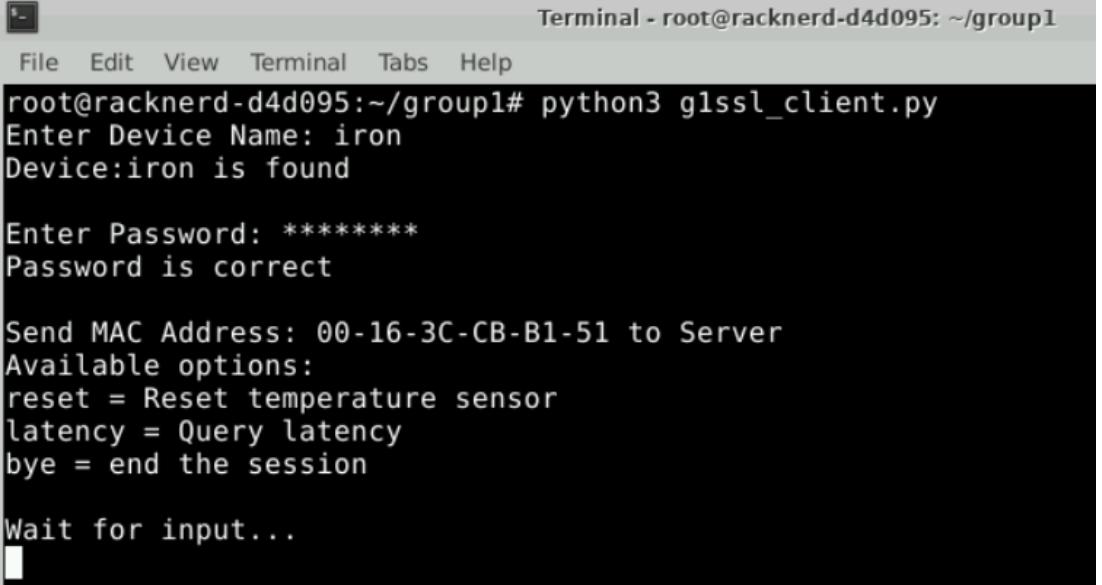
```
Terminal - root@racknerd-d4d095: ~/group1
File Edit View Terminal Tabs Help
root@racknerd-d4d095:~/group1# python3 g1ssl_client.py
Enter Device Name: iron
Device:iron is found

Enter Password: *****
Password is wrong

Session will be closed due to wrong password
```

Figure 17: Communication incorrect password

### 3.7.3. Scenario 3: All device, password and MAC address are all correct



```
Terminal - root@racknerd-d4d095: ~/group1
File Edit View Terminal Tabs Help
root@racknerd-d4d095:~/group1# python3 g1ssl_client.py
Enter Device Name: iron
Device:iron is found

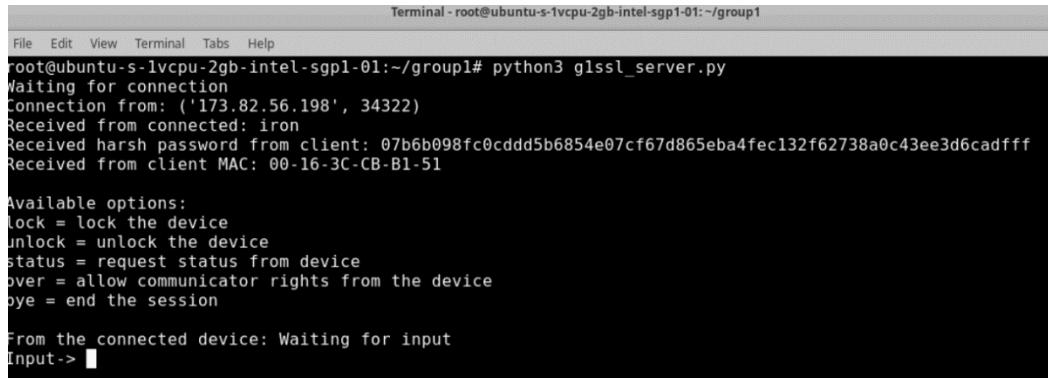
Enter Password: ******
Password is correct

Send MAC Address: 00-16-3C-CB-B1-51 to Server
Available options:
reset = Reset temperature sensor
latency = Query latency
bye = end the session

Wait for input...
```

Figure 18: Communicating correct credentials

At the Client, login successfully. Waiting for instructions from the Server



```

Terminal - root@ubuntu-s-1vcpu-2gb-intel-sgp1-01:~/group1
File Edit View Terminal Tabs Help
root@ubuntu-s-1vcpu-2gb-intel-sgp1-01:~/group1# python3 glssl_server.py
Waiting for connection
Connection from: ('173.82.56.198', 34322)
Received from connected: iron
Received harsh password from client: 07b6b098fc0cddd5b6854e07cf67d865eba4fec132f62738a0c43ee3d6cadfff
Received from client MAC: 00-16-3C-CB-B1-51

Available options:
lock = lock the device
unlock = unlock the device
status = request status from device
over = allow communicator rights from the device
bye = end the session

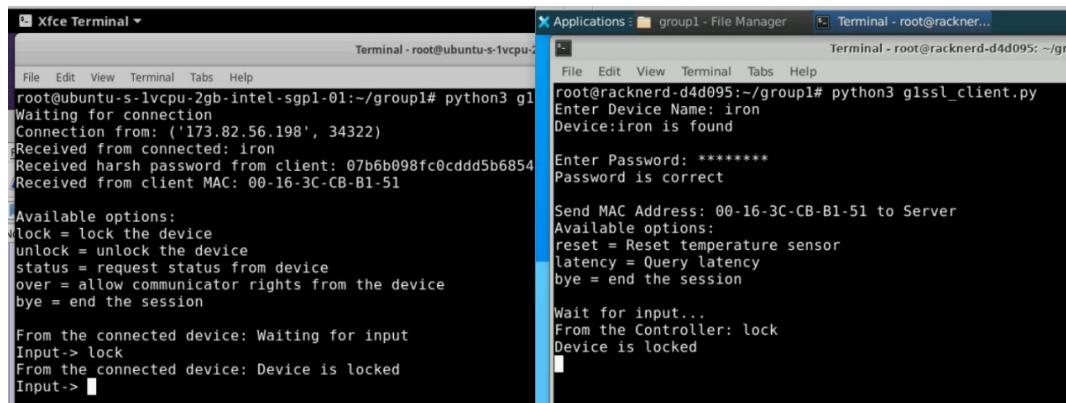
From the connected device: Waiting for input
Input-> [REDACTED]

```

*Figure 19: Server input*

At the Server end, showing available command

### 3.7.4. Scenario 4: lock the device via lock command



```

Xfce Terminal - Terminal - root@ubuntu-s-1vcpu-2gb-intel-sgp1-01:~/group1
File Edit View Terminal Tabs Help
root@ubuntu-s-1vcpu-2gb-intel-sgp1-01:~/group1# python3 glssl_server.py
Waiting for connection
Connection from: ('173.82.56.198', 34322)
Received from connected: iron
Received harsh password from client: 07b6b098fc0cddd5b6854
Received from client MAC: 00-16-3C-CB-B1-51

Available options:
lock = lock the device
unlock = unlock the device
status = request status from device
over = allow communicator rights from the device
bye = end the session

From the connected device: Waiting for input
Input-> lock
From the connected device: Device is locked
Input-> [REDACTED]

Applications - Terminal - root@rackner-d4d095:~/group1
File Edit View Terminal Tabs Help
root@rackner-d4d095:~/group1# python3 glssl_client.py
Enter Device Name: iron
Device:iron is found

Enter Password: *****
Password is correct

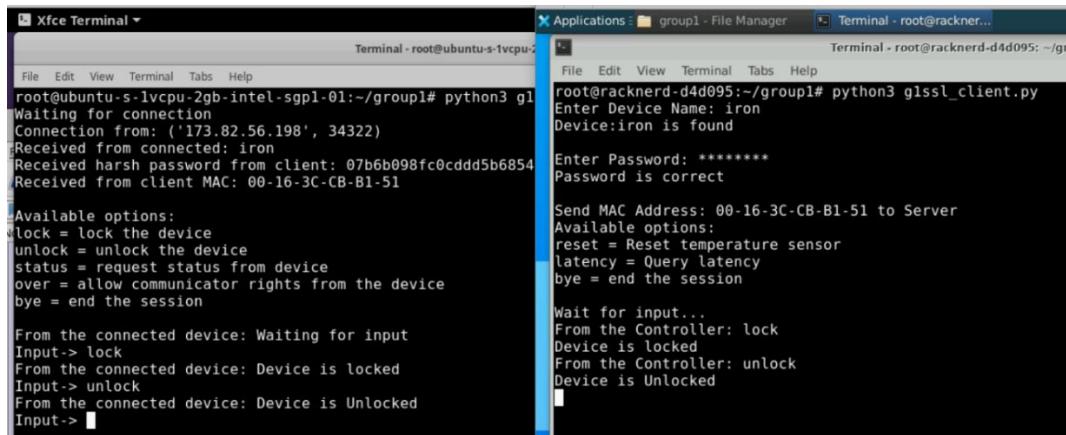
Send MAC Address: 00-16-3C-CB-B1-51 to Server
Available options:
reset = Reset temperature sensor
latency = Query latency
bye = end the session

Wait for input...
From the Controller: lock
Device is locked
[REDACTED]

```

*Figure 20: Locking the device*

### 3.7.5. Scenario 5: Unlock the device via unlock command



```

Xfce Terminal - Terminal - root@ubuntu-s-1vcpu-2gb-intel-sgp1-01:~/group1
File Edit View Terminal Tabs Help
root@ubuntu-s-1vcpu-2gb-intel-sgp1-01:~/group1# python3 glssl_server.py
Waiting for connection
Connection from: ('173.82.56.198', 34322)
Received from connected: iron
Received harsh password from client: 07b6b098fc0cddd5b6854
Received from client MAC: 00-16-3C-CB-B1-51

Available options:
lock = lock the device
unlock = unlock the device
status = request status from device
over = allow communicator rights from the device
bye = end the session

From the connected device: Waiting for input
Input-> lock
From the connected device: Device is locked
Input-> unlock
From the connected device: Device is Unlocked
Input-> [REDACTED]

Applications - Terminal - root@rackner-d4d095:~/group1
File Edit View Terminal Tabs Help
root@rackner-d4d095:~/group1# python3 glssl_client.py
Enter Device Name: iron
Device:iron is found

Enter Password: *****
Password is correct

Send MAC Address: 00-16-3C-CB-B1-51 to Server
Available options:
reset = Reset temperature sensor
latency = Query latency
bye = end the session

Wait for input...
From the Controller: unlock
Device is Unlocked
[REDACTED]

```

*Figure 21: Unlocking the device*

3.7.6. Scenario 6: Invalid command, command 'status' was wrongly entered and rejected.

```

Xfce Terminal - Terminal - root@ubuntu-s-1vcpu-2gb-intel-sgp1-01:~/group1# python3 gssl_se
File Edit View Terminal Tabs Help
root@ubuntu-s-1vcpu-2gb-intel-sgp1-01:~/group1# python3 gssl_se
Waiting for connection
Connection from: ('173.82.56.198', 34322)
Received from connected: iron
Received harsh password from client: 07b6b098fc0cddd5b6854e07cf6
Received from client MAC: 00-16-3C-CB-B1-51

Available options:
lock = lock the device
unlock = unlock the device
status = request status from device
over = allow communicator rights from the device
bye = end the session

From the connected device: Waiting for input
Input-> lock
From the connected device: Device is locked
Input-> unlock
From the connected device: Device is Unlocked
Input-> statu
Invalid Command
Input-> 

Applications : group1 - File Manager Terminal - root@rackner...
File Edit View Terminal Tabs Help
root@racknerd-d4d095:~/group1# python3 gssl_client.py
Enter Device Name: iron
Device:iron is found
Enter Password: *****
Password is correct

Send MAC Address: 00-16-3C-CB-B1-51 to Server
Available options:
reset = Reset temperature sensor
latency = Query latency
bye = end the session

Wait for input...
From the Controller: lock
Device is locked
From the Controller: unlock
Device is Unlocked

```

*Figure 22: Incorrect input communication*

3.7.7. Scenario 7: Get status from the device via status command

```

Xfce Terminal - Terminal - root@ubuntu-s-1vcpu-2gb-intel-sgp1-01:~/group1# python3 gssl_se
File Edit View Terminal Tabs Help
root@ubuntu-s-1vcpu-2gb-intel-sgp1-01:~/group1# python3 gssl_se
Waiting for connection
Connection from: ('173.82.56.198', 34322)
Received from connected: iron
Received harsh password from client: 07b6b098fc0cddd5b6854e07cf6
Received from client MAC: 00-16-3C-CB-B1-51

Available options:
lock = lock the device
unlock = unlock the device
status = request status from device
over = allow communicator rights from the device
bye = end the session

From the connected device: Waiting for input
Input-> lock
From the connected device: Device is locked
Input-> unlock
From the connected device: Device is Unlocked
Input-> statu
Invalid Command
Input-> status
From the connected device: Device temperature is normal: 46
Input-> 

Applications : group1 - File Manager Terminal - root@racknerd-d4d095:~/group1# python3 gssl_client.py
Enter Device Name: iron
Device:iron is found
Enter Password: *****
Password is correct

Send MAC Address: 00-16-3C-CB-B1-51 to Server
Available options:
reset = Reset temperature sensor
latency = Query latency
bye = end the session

Wait for input...
From the Controller: lock
Device is locked
From the Controller: unlock
Device is Unlocked
From the Controller: status

```

*Figure 23: Communication status updates such as temperature from the lock device*

### 3.7.8. Scenario 8: Assign commander's rights to client via over command

The screenshot shows two terminal windows. The left window is on an Ubuntu host (root@ubuntu-s-1vcpu-2gb-intel-01) and the right window is on a Rackner host (root@racknerd-d4d095). Both windows run the gssl\_client.py script.

**Ubuntu Host Terminal Output:**

```
root@ubuntu-s-1vcpu-2gb-intel-01:~/group1# python3 gssl_se
Waiting for connection
Connection from: ('173.82.56.198', 34322)
Received from connected: iron
Received harsh password from client: 07b6b098fc0cddd5b6854e07cf0
Received from client MAC: 00-16-3C-CB-B1-51

Available options:
lock = lock the device
unlock = unlock the device
status = request status from device
over = allow communicator rights from the device
bye = end the session

From the connected device: Waiting for input
 lock
From the connected device: Device is locked
 unlock
From the connected device: Device is Unlocked
 statu
 status
From the connected device: Device temperature is normal: 46
 over
```

**Rackner Host Terminal Output:**

```
root@racknerd-d4d095:~/group1# python3 gssl_client.p
Enter Device Name: iron
Device:iron is found

Enter Password: *****
Password is correct

Send MAC Address: 00-16-3C-CB-B1-51 to Server
Available options:
reset = Reset temperature sensor
latency = Query latency
bye = end the session

Wait for input...
From the Controller: lock
Device is locked
From the Controller: unlock
Device is Unlocked
From the Controller: status
From the Controller: over
Assigned with communication role
Input-> |
```

Figure 24: Assigning to client

### 3.7.9. Scenario 9: Query latency status between Server and Client via latency command

The screenshot shows two terminal windows. The left window is on an Ubuntu host (root@ubuntu-s-1vcpu-2gb-intel-01) and the right window is on a Rackner host (root@racknerd-d4d095). Both windows run the gssl\_client.py script.

**Ubuntu Host Terminal Output:**

```
root@ubuntu-s-1vcpu-2gb-intel-01:~/group1# python3 gssl_se
Waiting for connection
Connection from: ('173.82.56.198', 34322)
Received from connected: iron
Received harsh password from client: 07b6b098fc0cddd5b6854e07cf0
Received from client MAC: 00-16-3C-CB-B1-51

Available options:
lock = lock the device
unlock = unlock the device
status = request status from device
over = allow communicator rights from the device
bye = end the session

From the connected device: Waiting for input
 lock
From the connected device: Device is locked
 unlock
From the connected device: Device is Unlocked
 statu
 status
From the connected device: Device temperature is normal: 46
 over
From the connected device: latency
avg_latency: 169.04ms
min_latency:168.63ms
max_latency:169.46ms
packet_loss:0.0
 |
```

**Rackner Host Terminal Output:**

```
root@racknerd-d4d095:~/group1# python3 gssl_client.p
Enter Device Name: iron
Device:iron is found

Enter Password: *****
Password is correct

Send MAC Address: 00-16-3C-CB-B1-51 to Server
Available options:
reset = Reset temperature sensor
latency = Query latency
bye = end the session

Wait for input...
From the Controller: lock
Device is locked
From the Controller: unlock
Device is Unlocked
From the Controller: status
From the Controller: over
Assigned with communication role
Input-> status
Invalid Command
Input-> latency
|
```

Figure 25: Communicating latency check

### 3.7.10. Scenario 10: reset temperature sensor by the client via reset command

The screenshot shows two terminal windows. The left window is titled 'terminal - root@ubuntu-s-1vcpu-2gb-intel' and displays a session with a connected device named 'iron'. The right window is titled 'Terminal - root@racknerd...' and shows a client-side script running on a server with MAC address '00-16-3C-CB-B1-51'. The client sends a 'reset' command to the server, which responds with 'Device is found' and then performs a 'reset' operation.

```
terminal - root@ubuntu-s-1vcpu-2gb-intel
File Edit View Terminal Tabs Help
Waiting for connection
Connection from: ('173.82.56.198', 34322)
Received from connected: iron
Received harsh password from client: 07b6b098fc0cddd5b6854e07cf
Received from client MAC: 00-16-3C-CB-B1-51

Available options:
lock = lock the device
unlock = unlock the device
status = request status from device
over = allow communicator rights from the device
bye = end the session

From the connected device: Waiting for input
Input-> lock
From the connected device: Device is locked
Input-> unlock
From the connected device: Device is Unlocked
Input-> statu
Invalid Command
Input-> status
From the connected device: Device temperature is normal: 46
Input-> over
From the connected device: latency
avg_latency: 169.04ms
min_latency:168.63ms
max_latency:169.46ms
packet_loss:0.0
Input-> status
From the connected device: Device temperature is normal: 45
Input-> over
From the connected device: reset
Reset temperature sensor
Input->

Applications group1 - File Manager Terminal - root@racknerd-d4d095
File Edit View Terminal Tabs Help
root@racknerd-d4d095:~/group1# python3 glssl_client.py
Enter Device Name: iron
Device:iron is found

Enter Password: *****
Password is correct

Send MAC Address: 00-16-3C-CB-B1-51 to Server
Available options:
reset = Reset temperature sensor
latency = Query latency
bye = end the session

Wait for input...
From the Controller: lock
Device is locked
From the Controller: unlock
Device is Unlocked
From the Controller: status
From the Controller: over
Assigned with communication role
Input-> status
Invalid Command
Input-> latency
From the Controller: status
From the Controller: over
Assigned with communication role
Input-> reset

```

Figure 26: Reset the temperature sensor

### 3.7.11. Scenario 11: ending the session via bye command

```

terminal - root@ubuntu-s-1vcpu-2gb-intel-sgpl-01:~/.group1# 
File Edit View Terminal Tabs Help
Connection from: ('173.82.56.198', 34322)
Received from connected: iron
Received harsh password from client: 07b6b098fc0cddd5b6854e07cf
Received from client MAC: 00-16-3C-CB-B1-51

Available options:
lock = lock the device
unlock = unlock the device
status = request status from device
over = allow communicator rights from the device
bye = end the session

From the connected device: Waiting for input


```

Figure 27: Ending the sessions command

### 3.7.12. Scenario 12: Client is using a fake/invalid certificate

```

Xfce Terminal *
Terminal - root@ubuntu-s-1vcpu-2gb-intel-sgpl-01:~/.group1# python3 gssl_server.py
Waiting for connection
root@ubuntu-s-1vcpu-2gb-intel-sgpl-01:~/.group1# python3 gssl_client.py
Traceback (most recent call last):
  File "gssl_client.py", line 127, in <module>
    client_program()
  File "gssl_client.py", line 25, in client_program
    client_socket = ssl.wrap_socket(client_socket,ca_certs="/root/group1/client/glsrvr.crt",cert_reqs=ssl.CERT_REQUIRED)
  File "/usr/lib/python3.8/ssl.py", line 1405, in wrap_socket
    return context.wrap_socket(
  File "/usr/lib/python3.8/ssl.py", line 500, in wrap_socket
    return self.sslsocket_class._create(
  File "/usr/lib/python3.8/ssl.py", line 1040, in _create
    self.do_handshake()
  File "/usr/lib/python3.8/ssl.py", line 1309, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLAlert: [SSL: TLSV1_ALERT_UNKNOWN_CA] tlsv1 alert unknown ca (_
root@ubuntu-s-1vcpu-2gb-intel-sgpl-01:~/.group1# 

Terminal - root@rackner-d4d095:~/.group1#
File Edit View Terminal Tabs Help
root@rackner-d4d095:~# cd group1
root@rackner-d4d095:~/group1# python3 gssl_client.py
Traceback (most recent call last):
  File "gssl_client.py", line 127, in <module>
    client_program()
  File "gssl_client.py", line 25, in client_program
    client_socket = ssl.wrap_socket(client_socket,ca_certs="/root/group1/client/glsrvr.crt",cert_reqs=ssl.CERT_REQUIRED)
  File "/usr/lib/python3.8/ssl.py", line 1405, in wrap_socket
    return context.wrap_socket(
  File "/usr/lib/python3.8/ssl.py", line 500, in wrap_socket
    return self.sslsocket_class._create(
  File "/usr/lib/python3.8/ssl.py", line 1040, in _create
    self.do_handshake()
  File "/usr/lib/python3.8/ssl.py", line 1309, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: self signed certificate (_ssl.c:1131)
root@rackner-d4d095:~/.group1#

```

Figure 28: Secure checking of certificate for use

No connection can be established

## 4. Testing - Coding Quality Check via pylint

The screenshot shows two terminal windows side-by-side. The left window is titled 'kali@kali: ~/Desktop/SSA2022/SSA2022/SSA2022' and the right window is titled '~/Desktop/SSA2022/SSA2022/SSA2022/g1nossi\_server.py - Mousepad'. Both windows display the same pylint output for the file 'g1nossi\_server.py'.

```

File Actions Edit View Help
g1nossi_server.py:50:0: C0116: Missing function or method docstring (missing-function-docstring)
g1nossi_server.py:S1:15: R1732: Consider using 'with' for resource-allocating operations (consider-using-with)
g1nossi_server.py:S1:15: W1514: Using open without explicitly specifying an encoding (unspecified-encoding)
g1nossi_server.py:58:0: C0116: Missing function or method docstring (missing-function-docstring)
g1nossi_server.py:59:15: R1732: Consider using 'with' for resource-allocating operations (consider-using-with)
g1nossi_server.py:59:15: W1514: Using open without explicitly specifying an encoding (unspecified-encoding)
g1nossi_server.py:66:0: C0116: Missing function or method docstring (missing-function-docstring)
g1nossi_server.py:97:4: C0121: Comparison 'check_username(data_userdevice) == False' should be 'check_username(data_userdevice) is False' if checking for the singleton value False, or 'not check_username(data_userdevice)' if testing for falsiness (singleton-comparison)
g1nossi_server.py:111:5: C0121: Comparison 'check_password(data_userdevice, data_password) == False' should be 'check_password(data_userdevice, data_password) is False' if checking for the singleton value False, or 'not check_password(data_use
rdevice, data_password)' if testing for falsiness (singleton-comparison)
g1nossi_server.py:126:6: C0121: Comparison 'check_device(data_userdevice, data_password, data_mac) == False' should be 'check_device(data_userdevice, data_password, data_mac)' if testing for falsiness (single
ton-comparison)
g1nossi_server.py:146:10: R1726: Boolean condition 'True and data != 'bye'' may be simplified to 'data != 'bye'' (simplifiable-condition)
g1nossi_server.py:66:0: R0915: Too many statements (64/50) (too-many-statements)
g1nossi_server.py:3:0: W0611: Unused import datetime (unused-import)
g1nossi_server.py:4:0: W0611: Unused import time (unused-import)
g1nossi_server.py:5:0: W0611: Unused import os (unused-import)
g1nossi_server.py:6:0: W0611: Unused import threading (unused-import)

Your code has been rated at -4.39/10

(kali㉿kali)-[~/Desktop/SSA2022/SSA2022/SSA2022] 30 ✘

```

The right window contains the source code for 'g1nossi\_server.py' with some annotations:

```

import socket
import ssl
import datetime
import time
import os
import threading
import hashlib
import re

from pythonping import ping

#Function to get the ip address of the remote device
def find_enclosed(s):
    # find all matches
    matches = re.findall(r"\"(.*)\"", s)
    #matches = re.findall(r'(\w+', s)
    # if there are no matches return None
    if len(matches) == 0:
        return None
    # if it is a valid number change its type to a number
    for i in range(len(matches)):
        try:
            matches[i] = int(matches[i])
        except:
            pass
    # if there is only one match return it without a list
    if len(matches) == 1:
        return matches[0]
    return matches

# Function to encrypt password
def encrypt_password (passwd):
    #Encrypt the password and return the sha digest.
    hash_passwd = (passwd)
    hash_passwd = hash_passwd.encode("utf8")
    return hashlib.sha256(hash_passwd).hexdigest()

#pylint: disable=bad-indentation
#pylint: disable= missing-function-docstring
#pylint: disable= trailing whitespace

#Function to get the ip address of the remote device
def find_enclosed(s):
    # find all matches
    matches = re.findall(r"\"(.*)\"", s)
    #matches = re.findall(r'(\w+', s)
    # if there are no matches return None
    if len(matches) == 0:
        return None
    # if it is a valid number change its type to a number
    for i in range(len(matches)):
        try:
            matches[i] = int(matches[i])
        except:
            pass
    # if there is only one match return it without a list
    if len(matches) == 1:
        return matches[0]
    return matches

# Function to encrypt password
def encrypt_password (passwd):
    #Encrypt the password and return the sha digest.
    hash_passwd = (passwd)
    hash_passwd = hash_passwd.encode("utf8")
    return hashlib.sha256(hash_passwd).hexdigest()

```

Figure 29: Pylint before g1nossi\_server.py

The screenshot shows two terminal windows side-by-side. The left window is titled 'kali@kali: ~/Desktop/SSA2022/SSA2022/SSA2022' and the right window is titled '~/Desktop/SSA2022/SSA2022/SSA2022/g1nossi\_server.py - Mousepad'. Both windows display the same pylint output for the file 'g1nossi\_server.py' after modifications.

```

File Actions Edit View Help
d, data_mac) is False' if checking for the singleton value False, or 'not check_d
evice(data_userdevice, data_password, data_mac)' if testing for falsiness (single
ton-comparison)
g1nossi_server.py:149:10: R1726: Boolean condition 'True and data != 'bye'' may b
e simplified to 'data != 'bye'' (simplifiable-condition)
g1nossi_server.py:69:0: R0915: Too many statements (64/50) (too-many-statements)
g1nossi_server.py:3:0: W0611: Unused import datetime (unused-import)
g1nossi_server.py:4:0: W0611: Unused import time (unused-import)
g1nossi_server.py:5:0: W0611: Unused import os (unused-import)
g1nossi_server.py:6:0: W0611: Unused import threading (unused-import)

Your code has been rated at 3.93/10 (previous run: -4.39/10, +8.32)

(kali㉿kali)-[~/Desktop/SSA2022/SSA2022/SSA2022] 30 ✘

```

The right window contains the source code for 'g1nossi\_server.py' with annotations removed:

```

import socket
import ssl
import datetime
import time
import os
import threading
import hashlib
import re

from pythonping import ping

#pylint: disable=bad-indentation
#pylint: disable= missing-function-docstring
#pylint: disable= trailing whitespace

#Function to get the ip address of the remote device
def find_enclosed(s):
    # find all matches
    matches = re.findall(r"\"(.*)\"", s)
    #matches = re.findall(r'(\w+', s)
    # if there are no matches return None
    if len(matches) == 0:
        return None
    # if it is a valid number change its type to a number
    for i in range(len(matches)):
        try:
            matches[i] = int(matches[i])
        except:
            pass
    # if there is only one match return it without a list
    if len(matches) == 1:
        return matches[0]
    return matches

# Function to encrypt password
def encrypt_password (passwd):
    #Encrypt the password and return the sha digest.
    hash_passwd = (passwd)
    hash_passwd = hash_passwd.encode("utf8")
    return hashlib.sha256(hash_passwd).hexdigest()

```

Figure 30: After Pylint g1nossi\_server.py

The image shows two terminal windows side-by-side. The left window is titled 'Edit personal access tok...' and shows the results of running Pylint on `g1nossi_client.py`. It lists numerous errors and warnings, such as W0311 (Bad indentation), C0303 (Trailing whitespace), and C0116 (Missing function or method docstring). The right window is titled 'SSA2022' and displays the contents of the `g1nossi_client.py` file. The code includes imports for `socket`, `pwinput`, `uuid`, `random`, and `ssl`. It defines a `get_mac()` function to get the MAC address and a `client_program()` function to handle client connections and send/receive data.

```

File Actions Edit View Help
-indentation)
g1nossi_client.py:112:0: W0311: Bad indentation. Found 6 spaces, expected 24 (bad
-indentation)
g1nossi_client.py:114:0: C0303: Trailing whitespace (trailing-whitespace)
g1nossi_client.py:115:0: W0311: Bad indentation. Found 5 spaces, expected 20 (bad
-indentation)
g1nossi_client.py:116:0: W0311: Bad indentation. Found 6 spaces, expected 24 (bad
-indentation)
g1nossi_client.py:117:0: W0311: Bad indentation. Found 6 spaces, expected 24 (bad
-indentation)
g1nossi_client.py:118:53: C0303: Trailing whitespace (trailing-whitespace)
g1nossi_client.py:118:0: W0311: Bad indentation. Found 6 spaces, expected 24 (bad
-indentation)
g1nossi_client.py:119:0: C0303: Trailing whitespace (trailing-whitespace)
g1nossi_client.py:120:0: C0303: Trailing whitespace (trailing-whitespace)
g1nossi_client.py:121:0: W0311: Bad indentation. Found 4 spaces, expected 16 (bad
-indentation)
g1nossi_client.py:123:0: C0303: Trailing whitespace (trailing-whitespace)
g1nossi_client.py:1:0: C0114: Missing module docstring (missing-module-docstring)
g1nossi_client.py:9:0: C0116: Missing function or method docstring (missing-func
tion-docstring)
g1nossi_client.py:16:0: C0116: Missing function or method docstring (missing-func
tion-docstring)
g1nossi_client.py:32:1: R1702: Too many nested blocks (6/5) (too-many-nested-bloc
ks)
g1nossi_client.py:32:1: R1702: Too many nested blocks (6/5) (too-many-nested-bloc
ks)
g1nossi_client.py:93:7: W0612: Unused variable 'device_temp' (unused-variable)
g1nossi_client.py:16:0: R0912: Too many branches (16/12) (too-many-branches)
g1nossi_client.py:16:0: R0915: Too many statements (68/50) (too-many-statements)
g1nossi_client.py:3:0: C0411: standard import "import uuid" should be placed befo
re "import pwinput" (wrong-import-order)
g1nossi_client.py:4:0: C0411: standard import "import random" should be placed be
fore "import pwinput" (wrong-import-order)

Your code has been rated at -5.79/10

(kali㉿kali)-[~/Desktop/SSA2022/SSA2022/SSA2022]
$ 28 ×
```

```

File Edit Search View Document Help
import socket
import pwinput
import uuid
import random
#import ssl

#Function to get the MAC Address
def get_mac():
    mac_num = hex(uuid.getnode()).replace('0x', '').replace('L', '').upper()
    mac_num = mac_num.zfill(12)
    mac = '-'.join(mac_num[i:i+2] for i in range(0, 11, 2))
    return mac

def client_program():
    mylist = ['reset','over','latency','bye']
    #host = socket.gethostname() # Both Client and Server running on same machi
    #host='127.0.0.1'
    host='128.199.141.149'
    port = 1236 # socket server port number

    client_socket = socket.socket() # instantiate the socket
    client_socket.connect((host, port)) # connect to the server
    #client_socket = ssl.wrap_socket(client_socket,ca_certs="/root/group1/client

    message = input("Enter Device Name: ") # take input
    client_socket.send(message.encode())
    data_devicename = client_socket.recv(1024).decode()
    print(data_devicename+'\n') # show in terminal

    if data_devicename=='Device Name not found':
        print('Session will be closed') # show in terminal
        client_socket.close()

    else:
        message = pwinput.pwinput(prompt='Enter Password: ')
```

Figure 31: Pylint before `g1nossi_client.py`

The image shows two terminal windows side-by-side. The left window is titled 'Edit personal acc...' and shows the results of running Pylint on `g1nossi_client.py` again. The error count has significantly decreased, with only a few remaining: C0116 (Missing function or method docstring), C0301 (Line too long), and C0114 (Missing module docstring). The right window is titled 'SSA2022' and displays the updated `g1nossi_client.py` code. The changes include fixing the line length in the `get_mac()` function and adding missing docstrings for the `client_program()` and `get_mac()` functions.

```

File Actions Edit View Help
g1nossi_client.py:16:0: C0116: Missing function or method docstring (missing-func
tion-docstring)
g1nossi_client.py:32:1: R1702: Too many nested blocks (6/5) (too-many-nested-bloc
ks)
g1nossi_client.py:32:1: R1702: Too many nested blocks (6/5) (too-many-nested-bloc
ks)
g1nossi_client.py:93:7: W0612: Unused variable 'device_temp' (unused-variable)
g1nossi_client.py:16:0: R0912: Too many branches (16/12) (too-many-branches)
g1nossi_client.py:16:0: R0915: Too many statements (68/50) (too-many-statements)
g1nossi_client.py:3:0: C0411: standard import "import uuid" should be placed befo
re "import pwinput" (wrong-import-order)
g1nossi_client.py:4:0: C0411: standard import "import random" should be placed be
fore "import pwinput" (wrong-import-order)

Your code has been rated at 3.82/10 (previous run: -5.79/10, +9.61)

(kali㉿kali)-[~/Desktop/SSA2022/SSA2022/SSA2022]
$ pylint g1nossi_client.py
***** Module g1nossi_client
g1nossi_client.py:29:0: C0301: Line too long (120/100) (line-too-long)
g1nossi_client.py:1:0: C0114: Missing module docstring (missing-module-docstring)
g1nossi_client.py:13:0: C0116: Missing function or method docstring (missing-func
tion-docstring)
g1nossi_client.py:20:0: C0116: Missing function or method docstring (missing-func
tion-docstring)
g1nossi_client.py:97:7: W0612: Unused variable 'device_temp' (unused-variable)
g1nossi_client.py:20:0: R0912: Too many branches (16/12) (too-many-branches)
g1nossi_client.py:20:0: R0915: Too many statements (68/50) (too-many-statements)
g1nossi_client.py:3:0: C0411: standard import "import uuid" should be placed befo
re "import pwinput" (wrong-import-order)
g1nossi_client.py:4:0: C0411: standard import "import random" should be placed be
fore "import pwinput" (wrong-import-order)

Your code has been rated at 8.82/10 (previous run: 3.82/10, +5.00)

(kali㉿kali)-[~/Desktop/SSA2022/SSA2022/SSA2022]
$ 28 ×
```

```

File Edit Search View Document Help
import socket
import pwinput
import uuid
import random
# import ssl

#pylint: disable=bad-indentation
#pylint: disable=trailing-whitespace
#pylint: disable=too-many-nested-blocks

#Function to get the MAC Address
def get_mac():
    mac_num = hex(uuid.getnode()).replace('0x', '').replace('L', '').upper()
    mac_num = mac_num.zfill(12)
    mac = '-'.join(mac_num[i:i+2] for i in range(0, 11, 2))
    return mac

def client_program():
    mylist = ['reset','over','latency','bye']
    #host = socket.gethostname() # Both Client and Server running on same machi
    #host='127.0.0.1'
    host='128.199.141.149'
    port = 1236 # socket server port number

    client_socket = socket.socket() # instantiate the socket
    client_socket.connect((host, port)) # connect to the server
    #client_socket = ssl.wrap_socket(client_socket,ca_certs="/root/group1/client

    message = input("Enter Device Name: ") # take input
    client_socket.send(message.encode())
    data_devicename = client_socket.recv(1024).decode()
    print(data_devicename+'\n') # show in terminal

    if data_devicename=='Device Name not found':
        print('Session will be closed') # show in terminal
        client_socket.close()
```

Figure 32: After Pylint `g1nossi_client.py`

```

File Actions Edit View Help
g)
g1ssl_reg_server.py:9:0: C0103: Constant name "host" doesn't conform to UPPER_CAS
E naming style (invalid-name)
g1ssl_reg_server.py:10:0: C0103: Constant name "port" doesn't conform to UPPER_CA
SE naming style (invalid-name)
g1ssl_reg_server.py:11:0: C0103: Constant name "ThreadCount" doesn't conform to U
PPER_CASE naming style (invalid-name)
g1ssl_reg_server.py:19:19: W1505: Using deprecated method wrap_socket() (deprecat
ed-method)
g1ssl_reg_server.py:30:0: C0116: Missing function or method docstring (missing-fu
nction-docstring)
g1ssl_reg_server.py:37:0: C0116: Missing function or method docstring (missing-fu
nction-docstring)
g1ssl_reg_server.py:38:15: R1732: Consider using 'with' for resource-allocating o
perations (consider-using-with)
g1ssl_reg_server.py:47:15: W1514: Using open without explicitly specifying an enc
oding (unspecified-encoding)
g1ssl_reg_server.py:46:0: C0116: Missing function or method docstring (missing-fu
nction-docstring)
g1ssl_reg_server.py:47:15: R1732: Consider using 'with' for resource-allocating o
perations (consider-using-with)
g1ssl_reg_server.py:54:0: C0116: Missing function or method docstring (missing-fu
nction-docstring)
g1ssl_reg_server.py:70:7: C0121: Comparison 'check_device(name, password, mac_addr
) == False' should be 'check_device(name, password, mac_addr) is False' if check
ing for the singleton value False, or 'not check_device(name, password, mac_addr)
' if testing for falsiness (singleton-comparison)
g1ssl_reg_server.py:74:22: R1732: Consider using 'with' for resource-allocating o
perations (consider-using-with)
g1ssl_reg_server.py:74:22: R1732: Consider using 'with' for resource-allocating o
perations (consider-using-with)
g1ssl_reg_server.py:2:0: W0611: Unused import os (unused-import)

Your code has been rated at 1.90/10

(kali㉿kali)-[~/Desktop/SSA2022/SSA2022/SSA2022] 28 ✘

```

```

File Edit Search View Document Help
import socket
import os
import threading
import hashlib
import ssl

#
#host = '127.0.0.1'
host = '128.199.141.149'
port = 1238
ThreadCount = 0

# Create Socket (TCP) Connection
ServerSocket = socket.socket(family = socket.AF_INET, type = socket.SOCK_STREAM)

try:
    ServerSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) # allow reuse
    ServerSocket = ssl.wrap_socket(ServerSocket,server_side=True,certfile="/root/grc
    ServerSocket.bind((host, port))

except socket.error as e:
    print(str(e))

print('Waiting for a Connection..')
ServerSocket.listen(1)

# Function to encrypt password
def encrypt_password (passwd):
    #Encrypt the password and return the sha digest.
    hash_passwd = (passwd)
    hash_passwd = hash_passwd.encode("utf8")
    return hashlib.sha256(hash_passwd).hexdigest()

# Function to check if an username existed in asmis.txt
def check_username(username):

```

Figure 33: Pylint before g1ssl\_server.py

```

File Actions Edit View Help
g)
g1ssl_reg_server.py:14:0: C0103: Constant name "host" doesn't conform to UPPER_CAS
E naming style (invalid-name)
g1ssl_reg_server.py:15:0: C0103: Constant name "port" doesn't conform to UPPER_CAS
E naming style (invalid-name)
g1ssl_reg_server.py:16:0: C0103: Constant name "ThreadCount" doesn't conform to U
PPER_CASE naming style (invalid-name)
g1ssl_reg_server.py:24:19: W1505: Using deprecated method wrap_socket() (deprecate
d-method)
g1ssl_reg_server.py:35:0: C0116: Missing function or method docstring (missing-fu
nction-docstring)
g1ssl_reg_server.py:42:0: C0116: Missing function or method docstring (missing-fu
nction-docstring)
g1ssl_reg_server.py:43:15: R1732: Consider using 'with' for resource-allocating o
perations (consider-using-with)
g1ssl_reg_server.py:43:15: W1514: Using open without explicitly specifying an enc
oding (unspecified-encoding)
g1ssl_reg_server.py:51:0: C0116: Missing function or method docstring (missing-fu
nction-docstring)
g1ssl_reg_server.py:52:15: R1732: Consider using 'with' for resource-allocating o
perations (consider-using-with)
g1ssl_reg_server.py:52:15: W1514: Using open without explicitly specifying an enc
oding (unspecified-encoding)
g1ssl_reg_server.py:59:0: C0116: Missing function or method docstring (missing-fu
nction-docstring)
g1ssl_reg_server.py:75:7: C0121: Comparison 'check_device(name, password, mac_addr
) == False' should be 'check_device(name, password, mac_addr) is False' if check
ing for the singleton value False, or 'not check_device(name, password, mac_addr)
' if testing for falsiness (singleton-comparison)
g1ssl_reg_server.py:79:22: R1732: Consider using 'with' for resource-allocating o
perations (consider-using-with)
g1ssl_reg_server.py:79:22: R1732: Consider using 'with' for resource-allocating o
perations (consider-using-with)
g1ssl_reg_server.py:2:0: W0611: Unused import os (unused-import)

Your code has been rated at 7.30/10 (previous run: 1.90/10, +5.40)

(kali㉿kali)-[~/Desktop/SSA2022/SSA2022/SSA2022] 28 ✘

```

```

File Edit Search View Document Help
import socket
import os
import threading
import hashlib
import ssl

#pylint: disable = trailing whitespace
#pylint: disable = bad-indentation
#pylint: disable = line-too-long

#host = '127.0.0.1'
host = '128.199.141.149'
port = 1238
ThreadCount = 0

# Create Socket (TCP) Connection
ServerSocket = socket.socket(family = socket.AF_INET, type = socket.SOCK_STREAM)

try:
    ServerSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) # allow reuse
    ServerSocket = ssl.wrap_socket(ServerSocket,server_side=True,certfile="/root/grc
    ServerSocket.bind((host, port))

except socket.error as e:
    print(str(e))

print('Waiting for a Connection..')
ServerSocket.listen(1)

# Function to encrypt password
def encrypt_password (passwd):
    #Encrypt the password and return the sha digest.
    hash_passwd = (passwd)
    hash_passwd = hash_passwd.encode("utf8")
    return hashlib.sha256(hash_passwd).hexdigest()

```

Figure 34: After Pylint g1ssl\_client.py

The screenshot shows two terminal windows side-by-side. The left window is titled 'Edit personal account' and shows the output of running Pylint on `g1ssl_reg_client.py`. It lists numerous errors related to whitespace, indentation, and naming conventions. The right window is titled 'Screenshot\_2022...' and shows the source code for `g1ssl_reg_client.py`, which includes imports for socket, pwinput, uuid, and ssl, and defines a `get_mac` function to handle device connection and password input.

```

***** Module g1ssl_reg_client
g1ssl_reg_client.py:10:0: W0311: Bad indentation. Found 2 spaces, expected 4 (bad-indentation)
g1ssl_reg_client.py:11:0: W0311: Bad indentation. Found 2 spaces, expected 4 (bad-indentation)
g1ssl_reg_client.py:12:0: W0311: Bad indentation. Found 2 spaces, expected 4 (bad-indentation)
g1ssl_reg_client.py:13:0: W0311: Bad indentation. Found 2 spaces, expected 4 (bad-indentation)
g1ssl_reg_client.py:14:0: C0303: Trailing whitespace (trailing-whitespace)
g1ssl_reg_client.py:17:0: C0301: Line too long (104/100) (line-too-long)
g1ssl_reg_client.py:25:31: C0303: Trailing whitespace (trailing-whitespace)
g1ssl_reg_client.py:30:37: C0303: Trailing whitespace (trailing-whitespace)
g1ssl_reg_client.py:35:29: C0303: Trailing whitespace (trailing-whitespace)
g1ssl_reg_client.py:42:37: C0303: Trailing whitespace (trailing-whitespace)
g1ssl_reg_client.py:48:18: C0303: Trailing whitespace (trailing-whitespace)
g1ssl_reg_client.py:1:0: C0114: Missing module docstring (missing-module-docstring)
g1ssl_reg_client.py:6:0: C0103: Constant name "host" doesn't conform to UPPER_CAS_E naming style (invalid-name)
g1ssl_reg_client.py:7:0: C0103: Constant name "port" doesn't conform to UPPER_CAS_E naming style (invalid-name)
g1ssl_reg_client.py:9:0: C0116: Missing function or method docstring (missing-function-docstring)
g1ssl_reg_client.py:17:9: W1505: Using deprecated method wrap_socket() (deprecated-d-method)
g1ssl_reg_client.py:34:0: W0105: String statement has no effect (pointless-string-statement)
g1ssl_reg_client.py:43:0: C0103: Constant name "mac_addr" doesn't conform to UPPE_R_CASE naming style (invalid-name)
g1ssl_reg_client.py:3:0: C0411: standard import "import uuid" should be placed before "import pwinput" (wrong-import-order)
g1ssl_reg_client.py:4:0: C0411: standard import "import ssl" should be placed before "import pwinput" (wrong-import-order)

Your code has been rated at 2.86/10

(kali㉿kali)-[~/Desktop/SSA2022/SSA2022/SSA2022] 20 ×

```

```

File Edit Search View Document Help
import socket
import pwinput
import uuid
import ssl

host = '128.199.141.149'
port = 1238

def get_mac():
    mac_num = hex(uuid.getnode()).replace('0x', '').replace('L', '').upper()
    mac_num = mac_num.zfill(12)
    mac = '-'.join(mac_num[i:i+2] for i in range(0, 11, 2))
    return mac

# create an ipv4 (AF_INET) socket object using the tcp protocol (SOCK_STREAM)
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client = ssl.wrap_socket(client, ca_certs="/root/group1/client/g1server.crt", cert_reqs=ssl.CERT_REQUIRED)

# connect the client
# client.connect((target, port))
client.connect((host, port))
response = client.recv(2048)

# Input Device Name
name = input(response.decode())
client.send(str.encode(name))
response = client.recv(2048)

# Input Password
# password = input(response.decode())
password = pwinput.pwinput(response.decode(), mask='*')

client.send(str.encode(password))
''' Response : Status of Connection :
    1 : Registration successful
    2 : Connection Successful
    3 : Login Failed
...

```

Figure 35: Pylint before `g1ssl_reg_client.py`

The screenshot shows two terminal windows side-by-side. The left window is titled 'Edit personal account' and shows the output of running Pylint on `g1ssl_reg_client.py` again. The number of errors has significantly decreased, with only a few remaining related to whitespace and imports. The right window is titled 'Screenshot\_2022...' and shows the source code for `g1ssl_reg_client.py`, which has been updated to fix the identified issues.

```

***** Module g1ssl_reg_client
g1ssl_reg_client.py:43:0: C0103: Constant name "mac_addr" doesn't conform to UPPE_R_CASE naming style (invalid-name)
g1ssl_reg_client.py:38:0: C0411: standard import "import uuid" should be placed before "import pwinput" (wrong-import-order)
g1ssl_reg_client.py:4:0: C0411: standard import "import ssl" should be placed before "import pwinput" (wrong-import-order)

Your code has been rated at 2.86/10

(kali㉿kali)-[~/Desktop/SSA2022/SSA2022/SSA2022] 20 ×
$ pylint g1ssl_reg_client.py
***** Module g1ssl_reg_client
g1ssl_reg_client.py:22:0: C0301: Line too long (104/100) (line-too-long)
g1ssl_reg_client.py:1:0: C0114: Missing module docstring (missing-module-docstring)
g1ssl_reg_client.py:11:0: C0103: Constant name "host" doesn't conform to UPPER_CAS_E naming style (invalid-name)
g1ssl_reg_client.py:12:0: C0103: Constant name "port" doesn't conform to UPPER_CAS_E naming style (invalid-name)
g1ssl_reg_client.py:14:0: C0116: Missing function or method docstring (missing-function-docstring)
g1ssl_reg_client.py:22:9: W1505: Using deprecated method wrap_socket() (deprecated-d-method)
g1ssl_reg_client.py:39:0: W0105: String statement has no effect (pointless-string-statement)
g1ssl_reg_client.py:48:0: C0103: Constant name "mac_addr" doesn't conform to UPPE_R_CASE naming style (invalid-name)
g1ssl_reg_client.py:3:0: C0411: standard import "import uuid" should be placed before "import pwinput" (wrong-import-order)
g1ssl_reg_client.py:4:0: C0411: standard import "import ssl" should be placed before "import pwinput" (wrong-import-order)

Your code has been rated at 6.43/10 (previous run: 2.86/10, +3.57)

(kali㉿kali)-[~/Desktop/SSA2022/SSA2022/SSA2022] 20 ×

```

```

File Edit Search View Document Help
import socket
import pwinput
import uuid
import ssl

#pylint: disable= bad-indentation
#pylint: disable= trailing-whitespace

host = '128.199.141.149'
port = 1238

def get_mac():
    mac_num = hex(uuid.getnode()).replace('0x', '').replace('L', '').upper()
    mac_num = mac_num.zfill(12)
    mac = '-'.join(mac_num[i:i+2] for i in range(0, 11, 2))
    return mac

# create an ipv4 (AF_INET) socket object using the tcp protocol (SOCK_STREAM)
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client = ssl.wrap_socket(client, ca_certs="/root/group1/client/g1server.crt", cert_reqs=ssl.CERT_REQUIRED)

# connect the client
# client.connect((target, port))
client.connect((host, port))
response = client.recv(2048)

# Input Device Name
name = input(response.decode())
client.send(str.encode(name))
response = client.recv(2048)

# Input Password
# password = input(response.decode())
password = pwinput.pwinput(response.decode(), mask='*')

client.send(str.encode(password))

```

Figure 36: After Pylint `g1ssl_reg_client.py`

The screenshot shows two terminal windows. The left window displays Pylint errors for `g1ssl_client.py`, while the right window shows the source code for `g1ssl_client.py`.

```

File Actions Edit View Help
g1ssl_client.py:117:0: W0311: Bad indentation. Found 6 spaces, expected 24 (bad-indentation)
g1ssl_client.py:118:53: C0303: Trailing whitespace (trailing-whitespace)
g1ssl_client.py:118:0: W0311: Bad indentation. Found 6 spaces, expected 24 (bad-indentation)
g1ssl_client.py:119:0: C0303: Trailing whitespace (trailing-whitespace)
g1ssl_client.py:120:0: C0303: Trailing whitespace (trailing-whitespace)
g1ssl_client.py:121:0: W0311: Bad indentation. Found 4 spaces, expected 16 (bad-indentation)
g1ssl_client.py:123:0: C0303: Trailing whitespace (trailing-whitespace)
g1ssl_client.py:110:0: C0114: Missing module docstring (missing-module-docstring)
g1ssl_client.py:90:0: C0103: Constant name "host" doesn't conform to UPPER_CASE naming style (invalid-name)
g1ssl_client.py:90:0: C0103: Constant name "port" doesn't conform to UPPER_CASE naming style (invalid-name)
g1ssl_client.py:12:0: C0116: Missing function or method docstring (missing-function-docstring)
g1ssl_client.py:19:0: C0116: Missing function or method docstring (missing-function-docstring)
g1ssl_client.py:25:17: W1505: Using deprecated method wrap_socket() (deprecated-method)
g1ssl_client.py:32:1: R1702: Too many nested blocks (6/5) (too-many-nested-blocks)
g1ssl_client.py:32:1: R1702: Too many nested blocks (6/5) (too-many-nested-blocks)
g1ssl_client.py:93:7: W0612: Unused variable 'device temp' (unused-variable)
g1ssl_client.py:19:0: R0912: Too many branches (16/12) (too-many-branches)
g1ssl_client.py:19:0: R0915: Too many statements (67/50) (too-many-statements)
g1ssl_client.py:3:0: C0411: standard import "import uuid" should be placed before "import pwinput" (wrong-import-order)
g1ssl_client.py:4:0: C0411: standard import "import random" should be placed before "import pwinput" (wrong-import-order)
g1ssl_client.py:5:0: C0411: standard import "import ssl" should be placed before "import pwinput" (wrong-import-order)

Your code has been rated at -5.64/10

(kali㉿kali)-[~/Desktop/SSA2022/SSA2022/SSA2022]
```

The right window contains the `g1ssl_client.py` code:

```

import socket
import pwinput
import uuid
import random
import ssl

#host='127.0.0.1'
host = '128.199.141.149'
port = 1237 # socket server port number

#Function to get the MAC Address
def get_mac():
    mac_num = hex(uuid.getnode()).replace('0x', '').replace('L', '').upper()
    mac_num = mac_num.zfill(12)
    mac = '-'.join(mac_num[i:i+2] for i in range(0, 11, 2))
    return mac

def client_program():
    mylist = ['reset', 'over', 'latency', 'bye']
    #host = socket.gethostname() # Both Client and Server running on same machine

    client_socket = socket.socket() # instantiate the socket
    client_socket.connect((host, port)) # connect to the server
    client_socket = ssl.wrap_socket(client_socket, ca_certs='/root/group1/client.pem')

    message = input("Enter Device Name: ") # take input
    client_socket.send(message.encode())
    data_devicename = client_socket.recv(1024).decode()
    print(data_devicename+'\n') # show in terminal

    if data_devicename=='Device Name not found':
        print('Session will be closed') # show in terminal
        client_socket.close()

    else:
        message = pwinput.pwinput(prompt='Enter Password: ')
```

Figure 37: Pylint before `g1ssl_client.py`

The screenshot shows two terminal windows. The left window displays Pylint errors for `g1ssl_client.py`, while the right window shows the source code for `g1ssl_client.py`.

```

File Actions Edit View Help
"import pwinput" (wrong-import-order)
g1ssl_client.py:4:0: C0411: standard import "import random" should be placed before "import pwinput" (wrong-import-order)
g1ssl_client.py:5:0: C0411: standard import "import ssl" should be placed before "import pwinput" (wrong-import-order)

Your code has been rated at -5.64/10

(kali㉿kali)-[~/Desktop/SSA2022/SSA2022/SSA2022]
$ pylint g1ssl_client.py
***** Module g1ssl_client
g1ssl_client.py:29:0: C0301: Line too long (119/100) (line-too-long)
g1ssl_client.py:1:0: C0114: Missing module docstring (missing-module-docstring)
g1ssl_client.py:8:0: C0103: Constant name "host" doesn't conform to UPPER_CASE naming style (invalid-name)
g1ssl_client.py:9:0: C0103: Constant name "port" doesn't conform to UPPER_CASE naming style (invalid-name)
g1ssl_client.py:29:17: W1505: Using deprecated method wrap_socket() (deprecated-method)
g1ssl_client.py:36:1: R1702: Too many nested blocks (6/5) (too-many-nested-blocks)
g1ssl_client.py:36:1: R1702: Too many nested blocks (6/5) (too-many-nested-blocks)
g1ssl_client.py:97:7: W0612: Unused variable 'device temp' (unused-variable)
g1ssl_client.py:23:0: R0912: Too many branches (16/12) (too-many-branches)
g1ssl_client.py:23:0: R0915: Too many statements (67/50) (too-many-statements)
g1ssl_client.py:3:0: C0411: standard import "import uuid" should be placed before "import pwinput" (wrong-import-order)
g1ssl_client.py:4:0: C0411: standard import "import random" should be placed before "import pwinput" (wrong-import-order)
g1ssl_client.py:5:0: C0411: standard import "import ssl" should be placed before "import pwinput" (wrong-import-order)

Your code has been rated at 8.33/10 (previous run: -5.64/10, +13.97)

(kali㉿kali)-[~/Desktop/SSA2022/SSA2022/SSA2022]
```

The right window contains the `g1ssl_client.py` code, showing the changes made to fix the Pylint errors:

```

import socket
import pwinput
import uuid
import random
import ssl

#host='127.0.0.1'
host = '128.199.141.149'
port = 1237 # socket server port number

#pylint: disable=trailing-whitespace
#pylint: disable=missing-function-docstring
#pylint: disable=bad-indentation

#Function to get the MAC Address
def get_mac():
    mac_num = hex(uuid.getnode()).replace('0x', '').replace('L', '').upper()
    mac_num = mac_num.zfill(12)
    mac = '-'.join(mac_num[i:i+2] for i in range(0, 11, 2))
    return mac

def client_program():
    mylist = ['reset', 'over', 'latency', 'bye']
    #host = socket.gethostname() # Both Client and Server running on same machine

    client_socket = socket.socket() # instantiate the socket
    client_socket.connect((host, port)) # connect to the server
    client_socket = ssl.wrap_socket(client_socket, ca_certs='/root/group1/client.pem')

    message = input("Enter Device Name: ") # take input
    client_socket.send(message.encode())
    data_devicename = client_socket.recv(1024).decode()
    print(data_devicename+'\n') # show in terminal

    if data_devicename=='Device Name not found':
        print('Session will be closed') # show in terminal
        client_socket.close()
```

Figure 38: After Pylint `g1ssl_client.py`

## **5. Vulnerabilities identified and mitigations deployed (202 words)**

The development process initially indicated Transmission Control Protocol (6.1.) which could breach confidentiality through spoofing and cryptography failures. As designed in the attack-defence tree (Figure 42), we assumed spoofing would be the highest probability of a successful network attack (0.238 rating) and measures of incorporating Transport Layer Security were initiated as mitigation.

In (6.2), we identified that a malicious user might employ spoofing, information disclosure through social engineering, or breaching elevation of privilege may obtain the device name and password of lock. To authenticate a genuine user, the mitigation would be to cross-check the device's MAC address as the original one and refuse access if not.

In the attack-defence tree (Figure 42), we suggested a physical attack would pose the highest chance (0.75 rating) of the compromised door lock. Other vulnerabilities included stolen passwords (6.3.). We identified that the physical threat of a malicious user could shoulder surf a genuine user and obtain the password related to the smart lock device. Therefore masking the password would support mitigation.

### 5.1. Risk: No SSL encryption

5.1.1. It could lead to Man in the Middle (MITM) attack and result in confidential data stole

### 5.1.2. Running g1nossi\_server.py at the Server's end g1nossi\_client at the Client's end:

```

Terminal - root@ubuntu-s-1vcpu-2gb-intel-sgpl-01:~/group1# python3 glnos
Waiting for connection
Connection from: ('173.82.56.198', 44414)
Received from connected: iron
Received harsh password from client: 07b6b098fc0cddd5b6854e07
Received from client MAC: 00:16:3C:CB:B1:51

Available options:
lock = lock the device
unlock = unlock the device
status = request status from device
over = allow communicator rights from the device
bye = end the session

From the connected device: Waiting for input
Input-> lock
From the connected device: Device is locked
Input-> unlock
From the connected device: Device is Unlocked
Input-> over
From the connected device: latency
avg_latency:169.25ms
min_latency:168.71ms
max_latency:169.79ms
packet_loss:0.0
Input-> 

root@racknerd-d4d095:~/group1# python3 glnossi_client.py
Enter Device Name: iron
Device:iron is found

Enter Password: *****
Password is correct

Send MAC Address: 00:16:3C:CB:B1:51 to Server
Available options:
reset = Reset temperature sensor
latency = Query latency
bye = end the session

Wait for input...
From the Controller: lock
Device is locked
From the Controller: unlock
Device is Unlocked
From the Controller: over
Assigned with communication role
Input-> latency

```

Figure 39: No Secure Sockets Layer and insecure connection

### 5.1.3. Using Wireshark to view the connection between client and server via port: 1236 and 44414

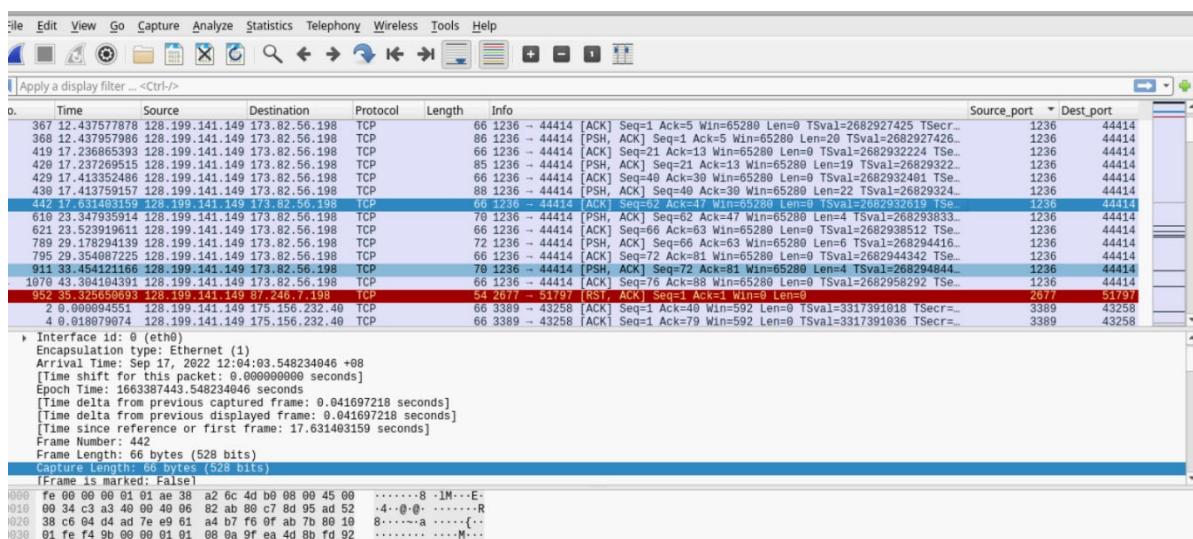


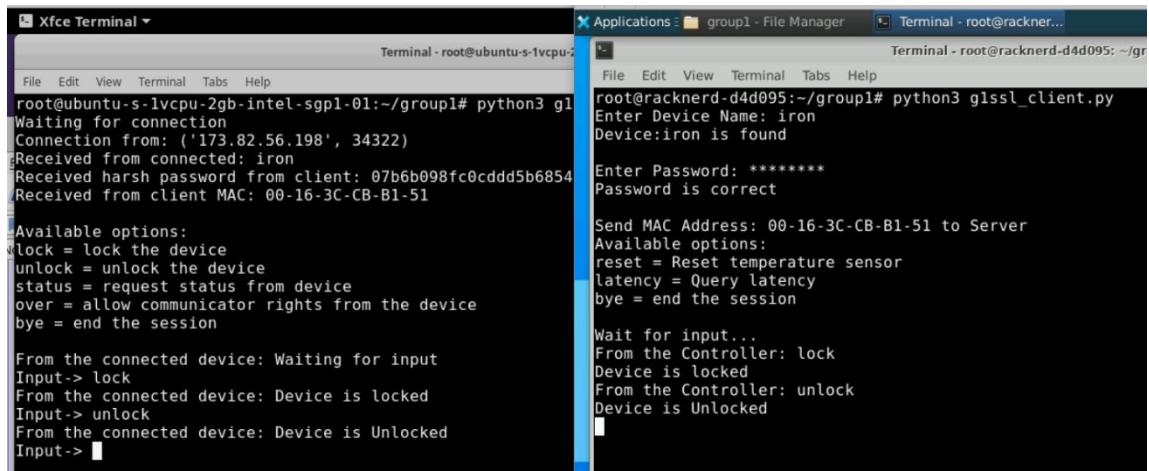
Figure 36: Wireshark confirmation of insecure connection

The protocol is just TCP

#### 5.1.4. Mitigation: Implement SSL encryption to mitigate the risk

#### 5.1.5. Running g1ssl\_server.py at the Server's end g1ssl\_client at the

Client's end:



The image shows two terminal windows side-by-side. The left terminal window, titled 'Xfce Terminal', is running on a Ubuntu system (root@ubuntu-s-1vcpu-2gb-intel-sgpl-01). It displays the output of the command 'python3 g1ssl\_server.py'. The server is waiting for a connection from an IP address (173.82.56.198) and port 34322. It receives a harsh password from the client and a MAC address (00-16-3C-CB-B1-51). It then lists available options: lock, unlock, status, over, and bye. The server also logs messages from the connected device indicating it is waiting for input, locked, and unlocked.

```
root@ubuntu-s-1vcpu-2gb-intel-sgpl-01:~/group1# python3 g1ssl_server.py
Waiting for connection
Connection from: ('173.82.56.198', 34322)
Received from connected: iron
Received harsh password from client: 07b6b098fc0cddd5b6854
Received from client MAC: 00-16-3C-CB-B1-51

Available options:
lock = lock the device
unlock = unlock the device
status = request status from device
over = allow communicator rights from the device
bye = end the session

From the connected device: Waiting for input
Input-> lock
From the connected device: Device is locked
Input-> unlock
From the connected device: Device is Unlocked
Input->
```

The right terminal window, titled 'Terminal - root@rackner...', is running on a racknerd system (root@racknerd-d4d095). It displays the output of the command 'python3 g1ssl\_client.py'. The client connects to the server using the IP address 173.82.56.198. It enters the device name 'iron' and finds it. It then enters a password, which is confirmed as correct. The client sends the MAC address 00-16-3C-CB-B1-51 to the server. It lists available options: reset, latency, and bye. The client then waits for input, receives a lock command from the controller, and then an unlock command, resulting in the device being unlocked.

```
root@racknerd-d4d095:~/group1# python3 g1ssl_client.py
Enter Device Name: iron
Device:iron is found

Enter Password: *****
Password is correct

Send MAC Address: 00-16-3C-CB-B1-51 to Server
Available options:
reset = Reset temperature sensor
latency = Query latency
bye = end the session

Wait for input...
From the Controller: lock
Device is locked
From the Controller: unlock
Device is Unlocked
```

Figure 40: Secure Sockets Layer and secure connection

### 5.1.6. Using Wireshark to view the connection between client and server via port: 1237 and 34322

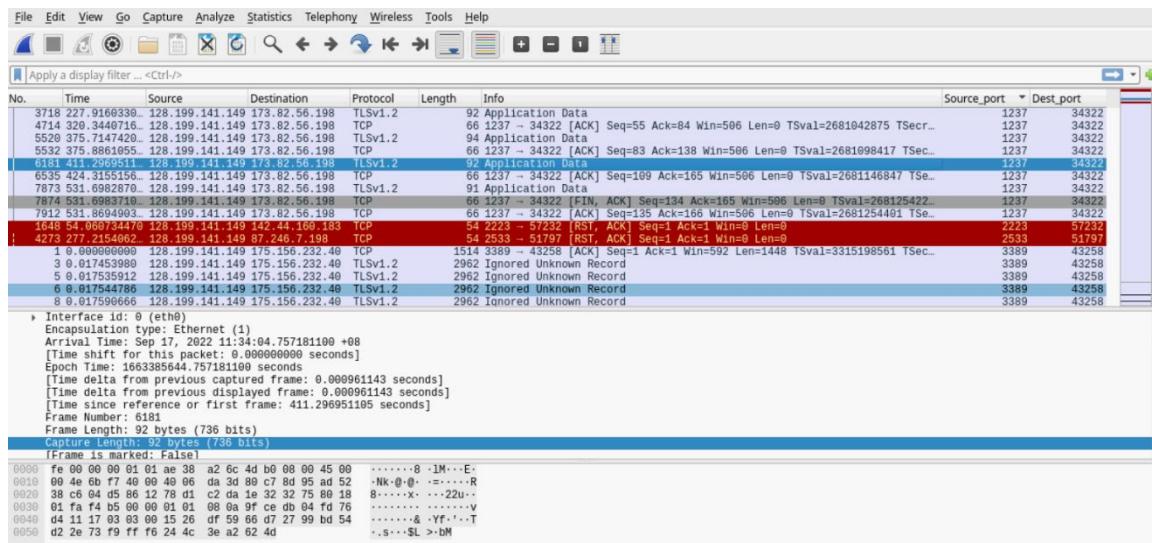


Figure 41: Transport Layer Security protocol enabled

The protocol is now TLS

## 5.2. Risk: Credentials

### 5.2.1. Device name and Password may be stolen

5.2.2. Mitigation: To check the client's MAC address as part of the authentication process.

5.3. Risk: Password input is not masked

5.3.1. Password entered by the user may be stolen via shoulder surfing

5.3.2. Mitigation: Password input is masked with \*

## **6. Discussion of any omissions/ Lack of mitigations (Word count 167)**

The development process provided many constraints on our initial design document.

Time was a critical factor which meant numerous aspects could not be completed.

However, we followed many OWASP security best practices and incorporated TLS/SSL communication between devices as planned. Below are some of the areas which were started in development but incomplete. A critical aspect would consider introducing web sockets and a web-based application to include these omissions.

### **6.1. Lack of certificate from Certificate Authority (CA)**

6.1.1. The self-signed certificate needs to be installed in the device to communicate with the server.

### **6.2. Lack of multi-threading**

6.2.1. Only a single device can be established with the server at any point in time

### **6.3. No Server's timeout**

6.3.1. Communication should be disconnected gracefully when there is no activity between the server and client.

6.3.2. This reduces the chances of unauthorised or hacking when the connection is permanently established

#### 6.4. Credential management

6.4.1. Lock account after three invalid attempts

6.4.2. Enforce complex password requirements

6.4.3. Enforce Multi-Factor Authentication

## 7. References

- 7.1. AB Hossain (2022), Measuring ping latency of a server - Python  
<https://www.anycodings.com/1questions/1359093/measuring-ping-latency-of-a-server-python> [Last accessed: 12-Sep-2022]
- 7.2. Adam Bavosa (2020), Python Socket Programming: Client, Server and Peer Libraries  
<https://www.pubnub.com/blog/socket-programming-in-python-client-server-p2p/>  
[Last accessed: 09-Sep-2022]
- 7.3. Carlo Hääläinen (2013) Python SSL socket echo test with self-signed certificate  
<https://carlo-hamalainen.net/2013/01/24/python-ssl-socket-echo-test-with-self-signed-certificate/> [Last accessed: 15-Sep-2022]
- 7.4. Conex. 2021. *How is the Philips smart lock resistant to moisture and high temperature?* [Online]. Available: <https://www.conex-global.com/blog/moisture-high-temperature.html> [Last Accessed: 10-Sep-2022].
- 7.5. De Win, B., Piessens, F., Joosen, W. & Verhanneman, T. On the importance of the separation-of-concerns principle in secure software engineering. Workshop on the Application of Engineering Principles to System Security Design, 2002. Citeseer, 1-10.
- 7.6. Fedingo (2021), How to Install OpenSSL in Ubuntu  
<https://fedingo.com/how-to-install-openssl-in-ubuntu/>  
[Last accessed: 10-Sep-2022]
- 7.7. Greg Mattes (2008), Getting MAC Address

<https://stackoverflow.com/questions/159137/getting-mac-address>

[Last Accessed 10-Sep-2022]

- 7.8. *Group 1 Secure Systems Architecture August 2022. Development Team Project: Design Document* [Online]. University of Essex Online. Available: <https://www.my-course.co.uk/mod/assign/view.php?id=681617> [Last accessed 18-Sep- 2022].

- 7.9. Jason Brownlee (2018) How to Generate Random Numbers in Python

<https://machinelearningmastery.com/how-to-generate-random-numbers-in-python/> [Last Accessed 12-Sep- 2022]

- 7.10. Jose Manuel Ortega, Dr. M. O. Faruque Sarker, Sam Washington (2019) Learning Python Networking - Second Edition. Available via the O'reilly [Accessed 09-Sep- 2022]

- 7.11. Panka (2022), Python Socket Programming - Server, Client Example

<https://www.digitalocean.com/community/tutorials/python-socket-programming-server-client> [Last accessed: 10-Sep-2022]

- 7.12. Pavelić, M., Lončarić, Z., Vuković, M. & Kušek, M. Internet of Things Cyber Security: Smart Door Lock System. 2018 International Conference on Smart Systems and Technologies (SST), 10-12 Oct. 2018 2018. 227-232.

- 7.13. Shahriar Shovon (2022), How to Install and Use Wireshark on Ubuntu [https://linuxhint.com/install\\_wireshark\\_ubuntu/](https://linuxhint.com/install_wireshark_ubuntu/) [Last accessed: 15-Sep-2022]

- 7.14. Tomerikoo (2021), Getting a hidden password input

<https://stackoverflow.com/questions/9202224/getting-a-hidden-password-input> [Last Accessed 10-Sep- 2022]

## 7.15. Two-Bit Alchemist (2016), Extract string within parentheses - PYTHON

<https://stackoverflow.com/questions/38999344/extract-string-within-parentheses-python> [Last Accessed 10-Sep- 2022]

## 8. Appendices

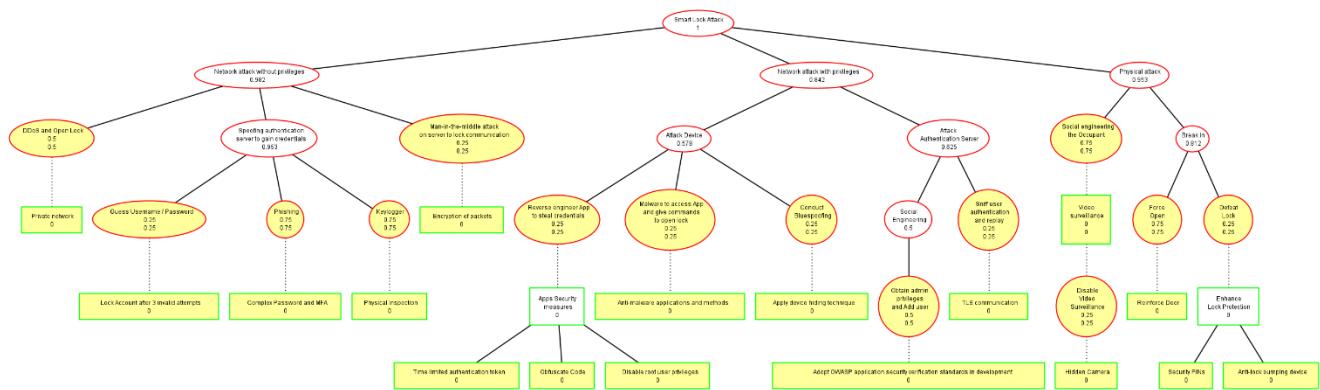


Figure 42: Attack-Defence Tree on Smart Door Lock from Design Document