Contents: **Page:**

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 ommissions 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.

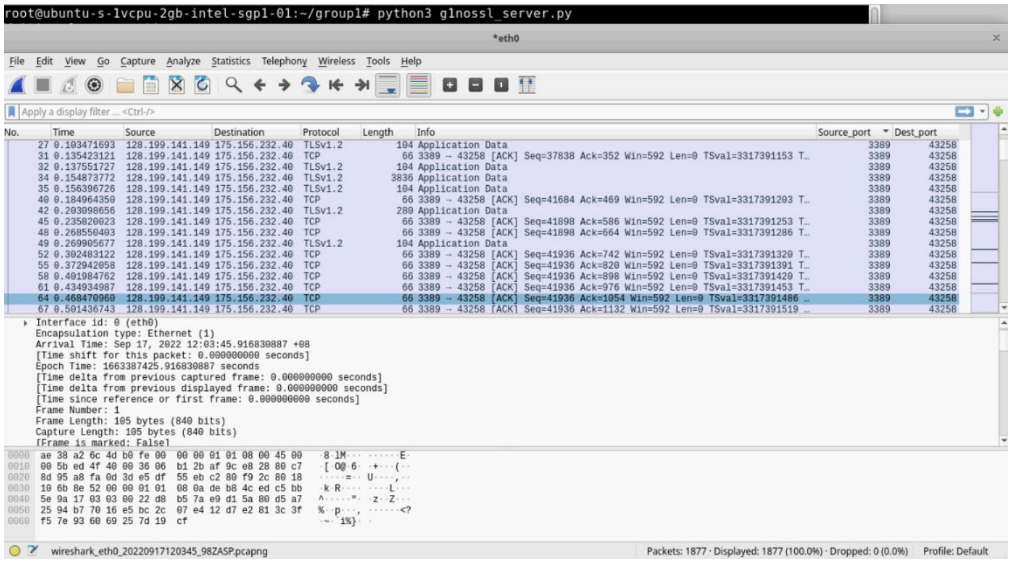


Figure 2: Wireshark report example from g1nossl_server.py

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-sgp1-01:/# 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)

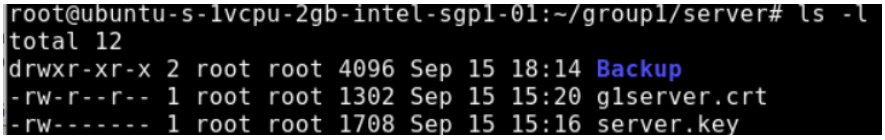


Figure 4: Moving certificates and keys

2.2.16.5. Since we are using self-signed certificates, copy g1server.crt to the 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

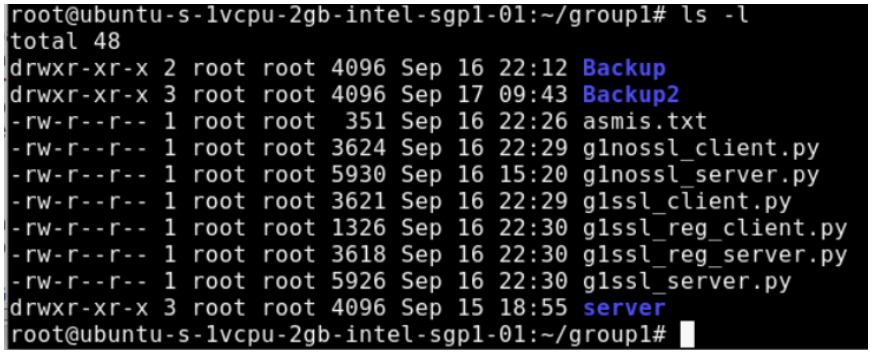


Figure 6: Creating directory and storing scripts

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. g1nossl_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. g1nossl_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

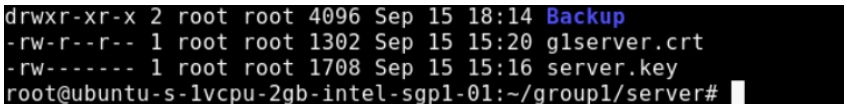


Figure 7: Creating a server directory

3. Output from demonstrations

Registration process

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

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

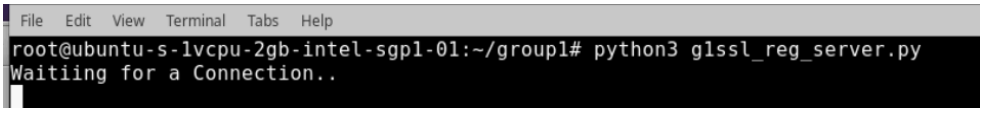


Figure 8: Server connection

3.3. At the Client end:

3.3.1. Enter: g1ssl_reg_client.py

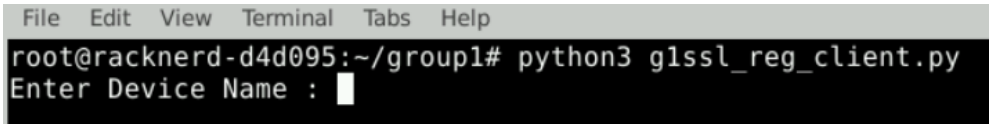


Figure 9: Choosing a device name in client

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

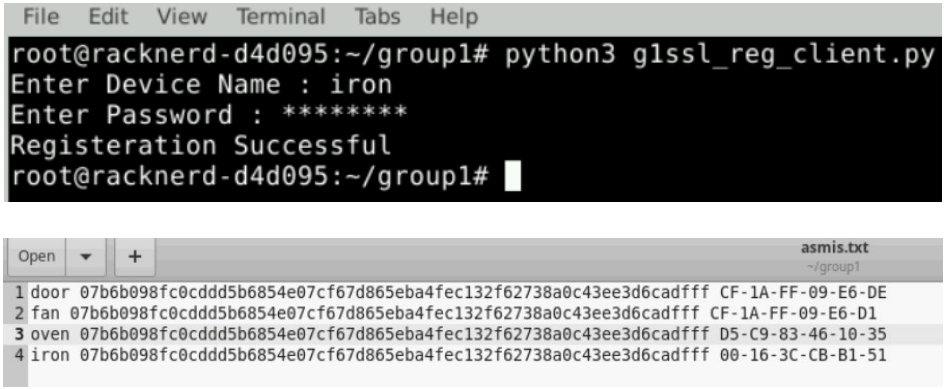


Figure 10: Confirmation of communication

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:

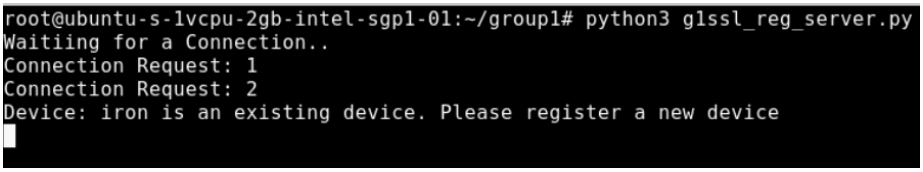
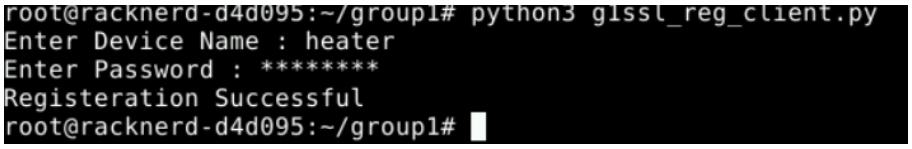


Figure 11: Checking for existing devices

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.



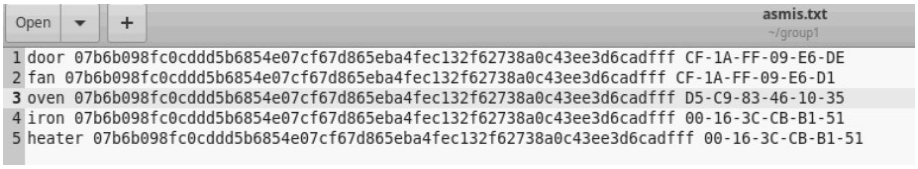


Figure 12: MAC address confirmation

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

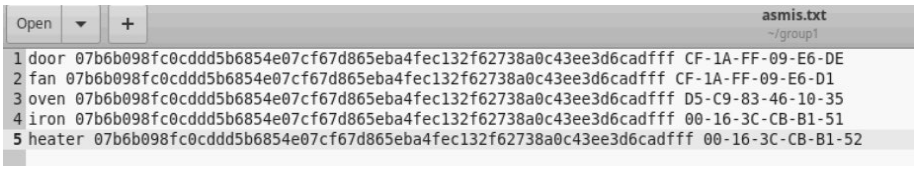


Figure 13: MAC address change

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

Figure 14: Server connection

3.6. At the Client's end, enter: g1ssl_client.py

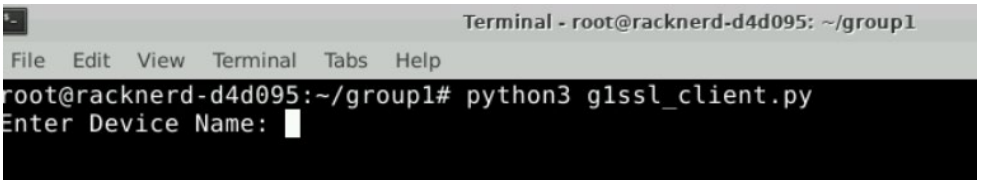


Figure 15: Client connection

3.7. Communication between Server and Client:

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

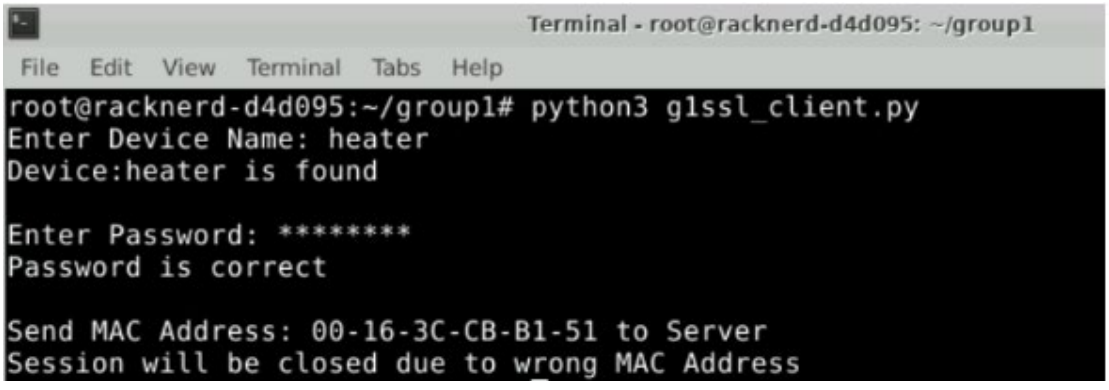


Figure 16: Communicating errors

3.7.2. Scenario 2: Device is correct but wrong password

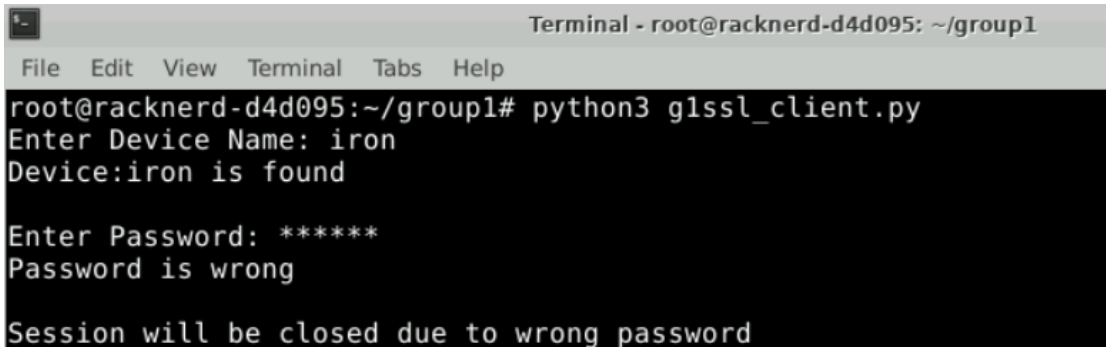


Figure 17: Communication incorrect password

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

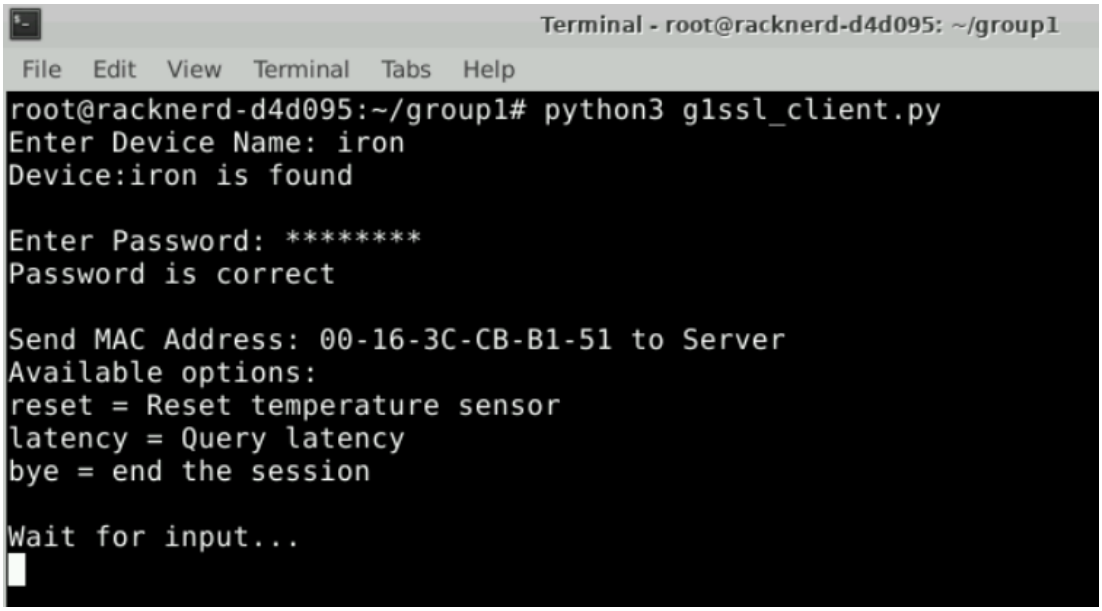


Figure 18: Communicating correct credentials

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

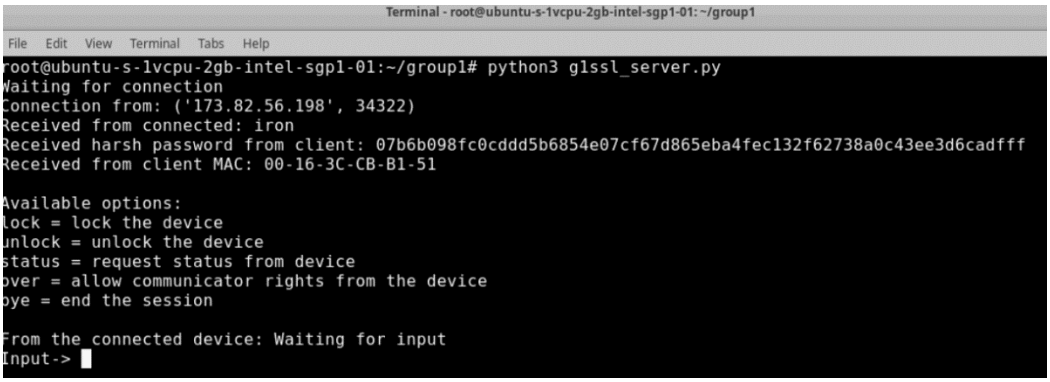


Figure 19: Server input

At the Server end, showing available command

3.7.4. Scenario 4: lock the device via lock command

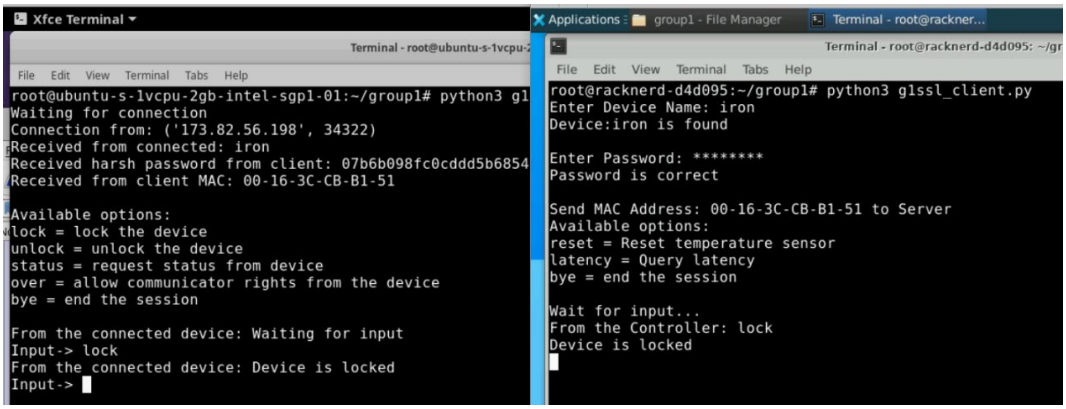


Figure 20: Locking the device

3.7.5. Scenario 5: Unlock the device via unlock command

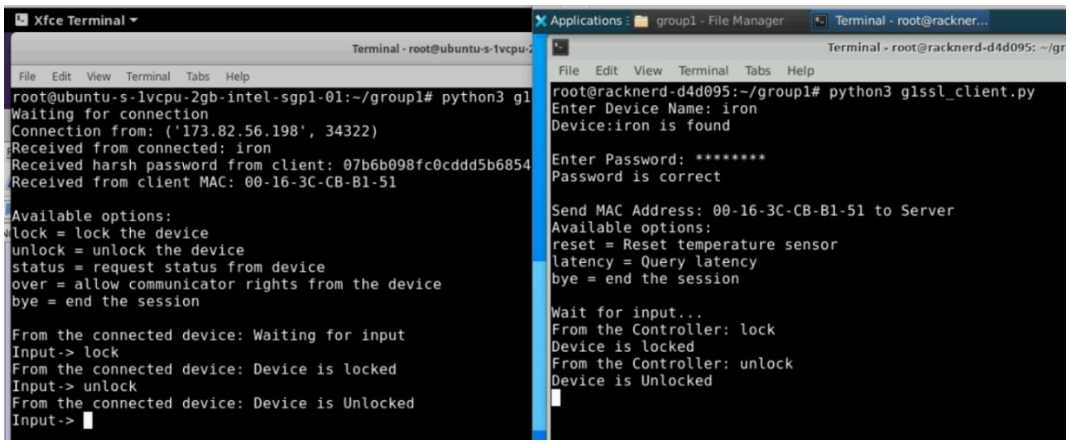


Figure 21: Unlocking the device

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

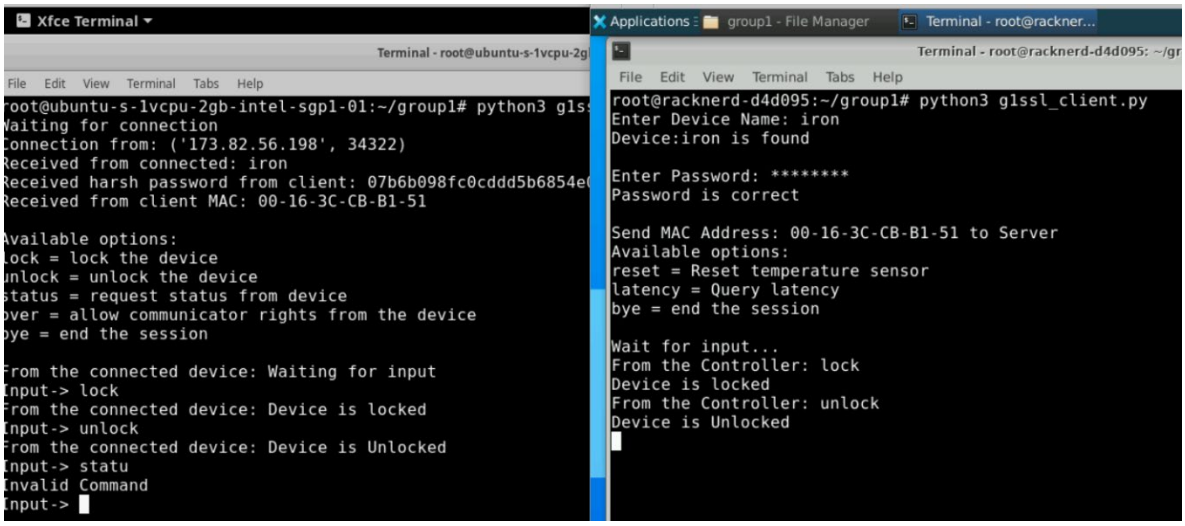


Figure 22: Incorrect input communication

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

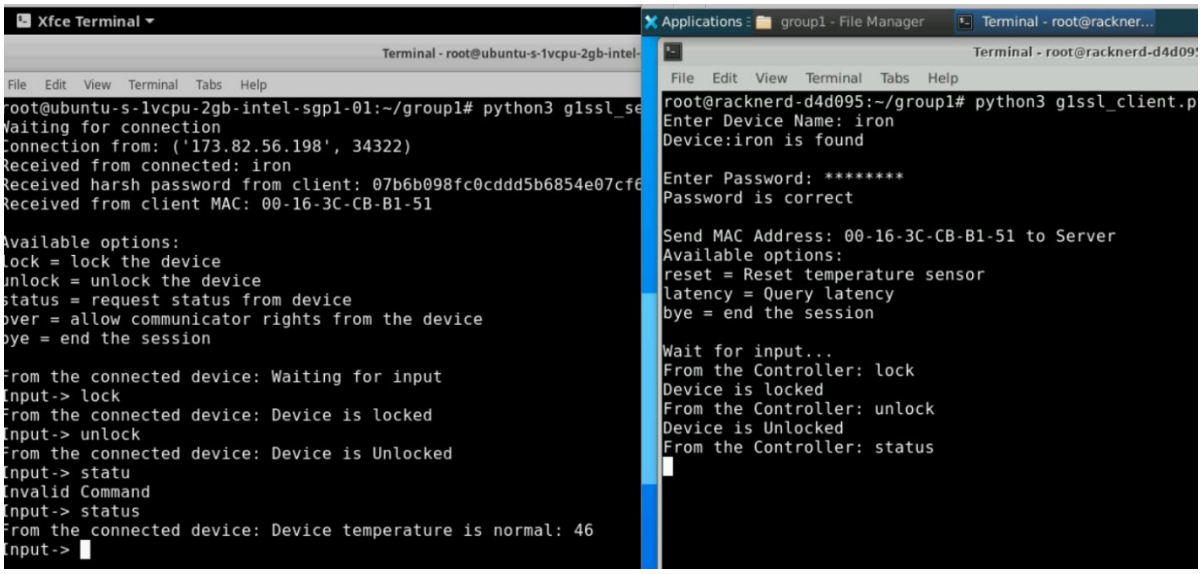


Figure 23: Communication status updates such as temperature from the lock device

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

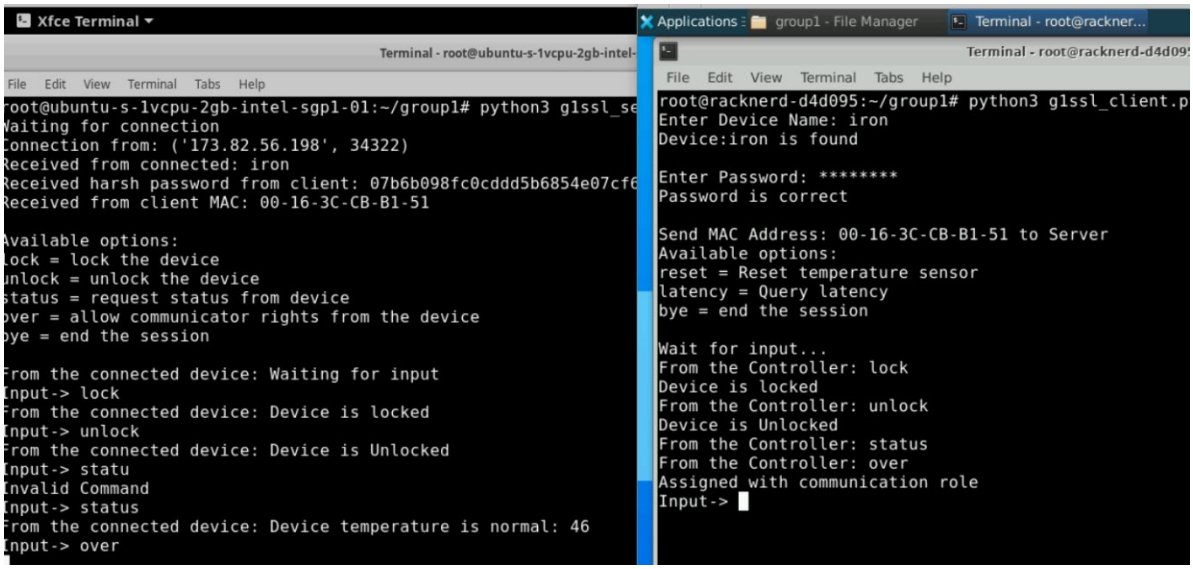


Figure 24: Assigning to client

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

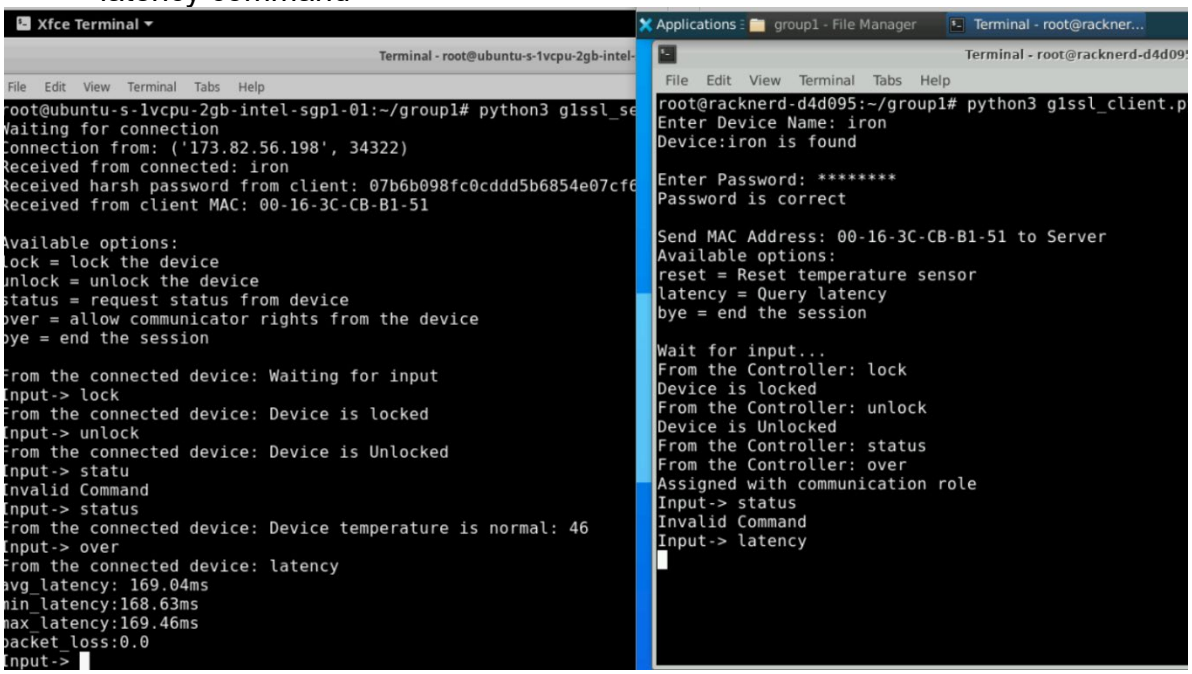


Figure 25: Communicating latency check

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

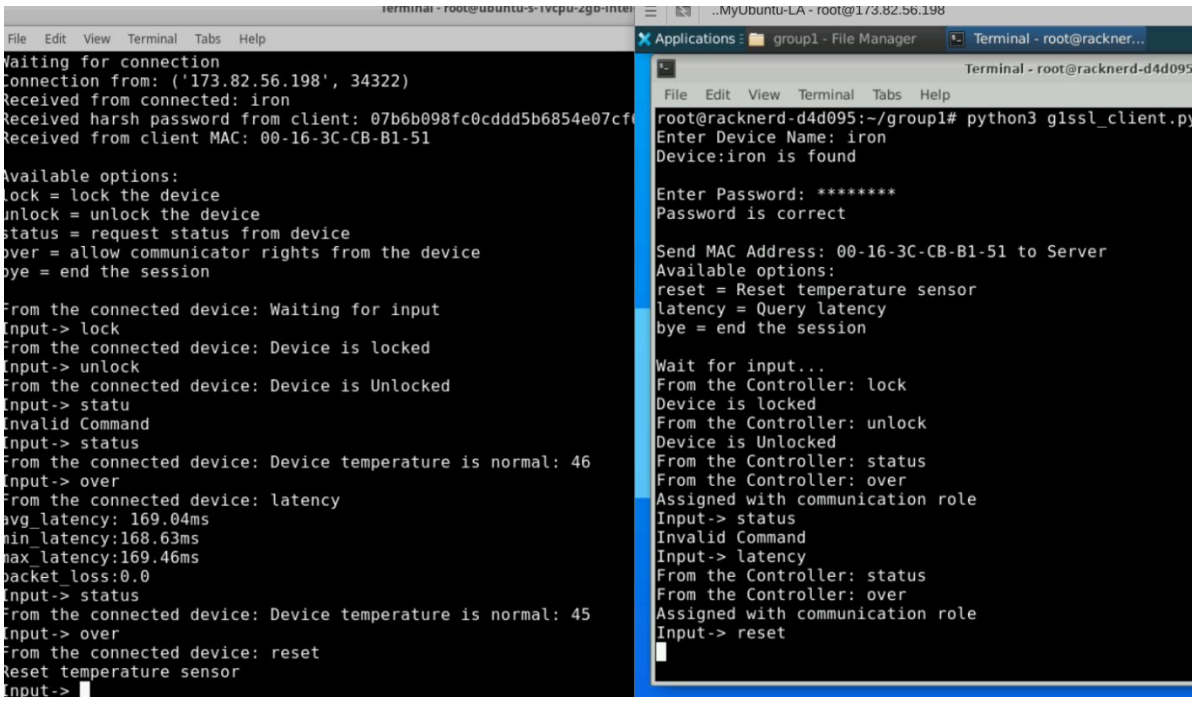


Figure 26: Reset the temperature sensor

3.7.11. Scenario 11: ending the session via bye command

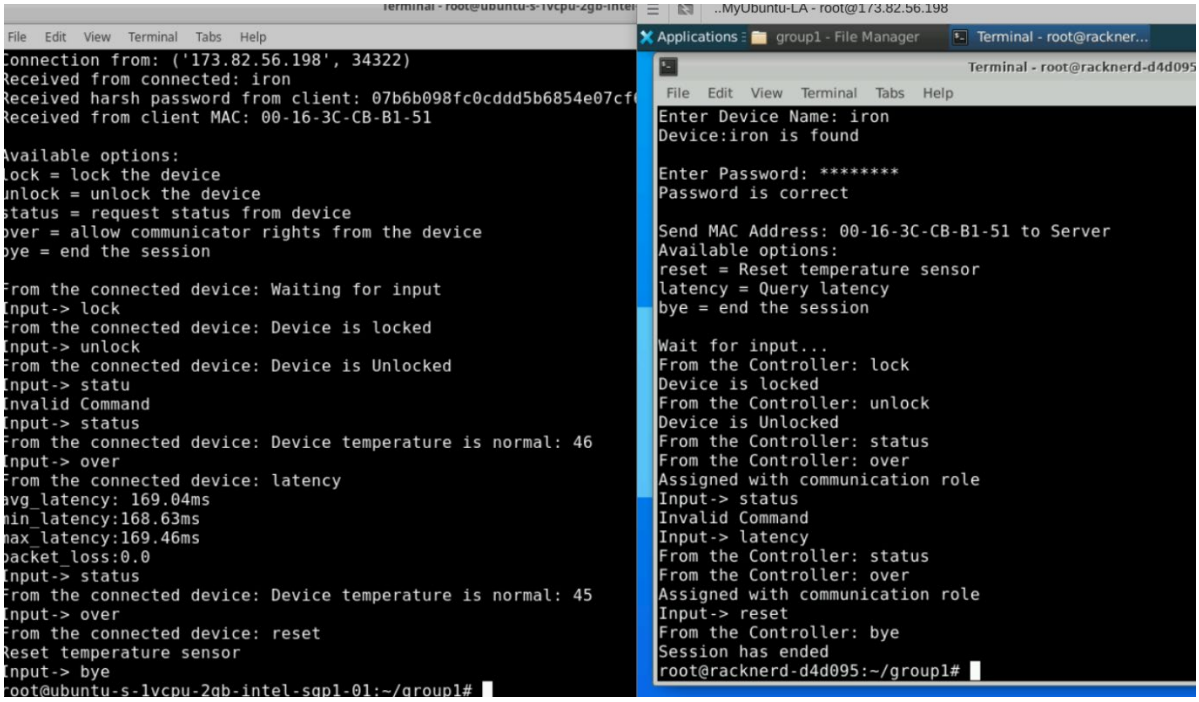


Figure 27: Ending the sessions command

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

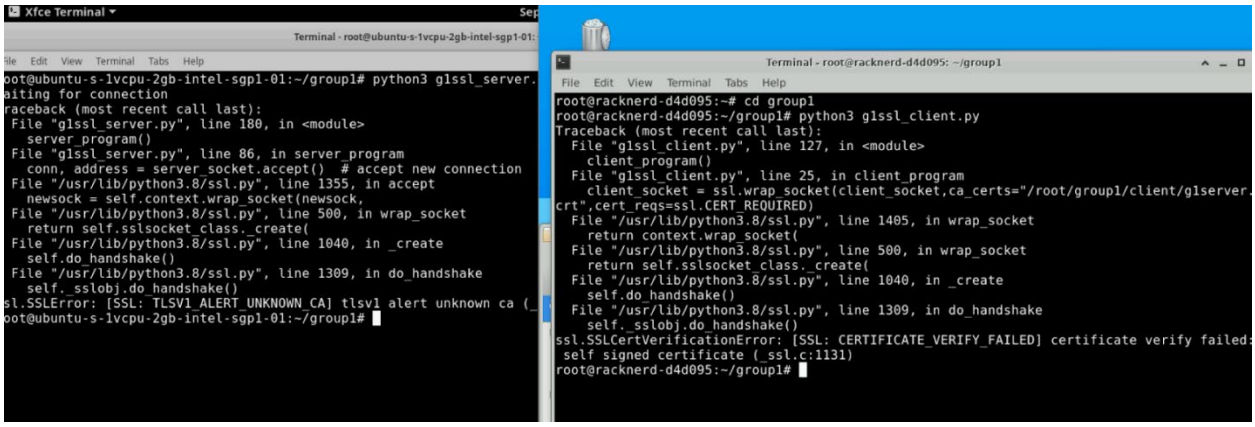


Figure 28: Secure checking of certificate for use

No connection can be established

4. Testing - Coding Quality Check via pylint

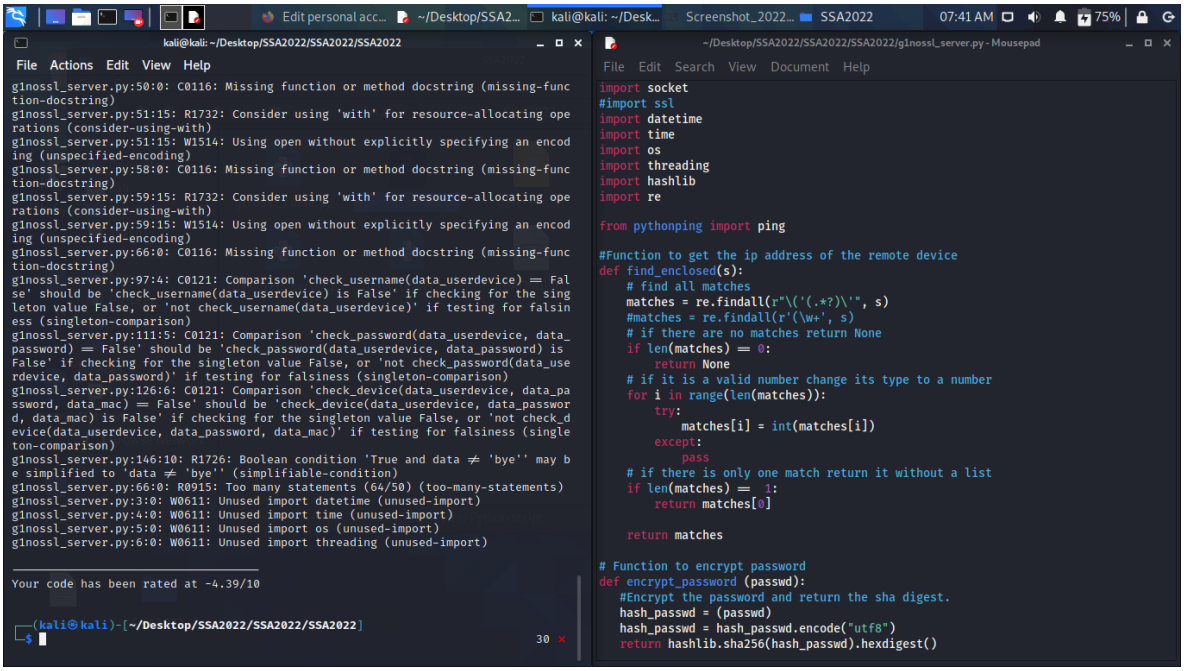


Figure 29: Pylint before g1nossl_server.py

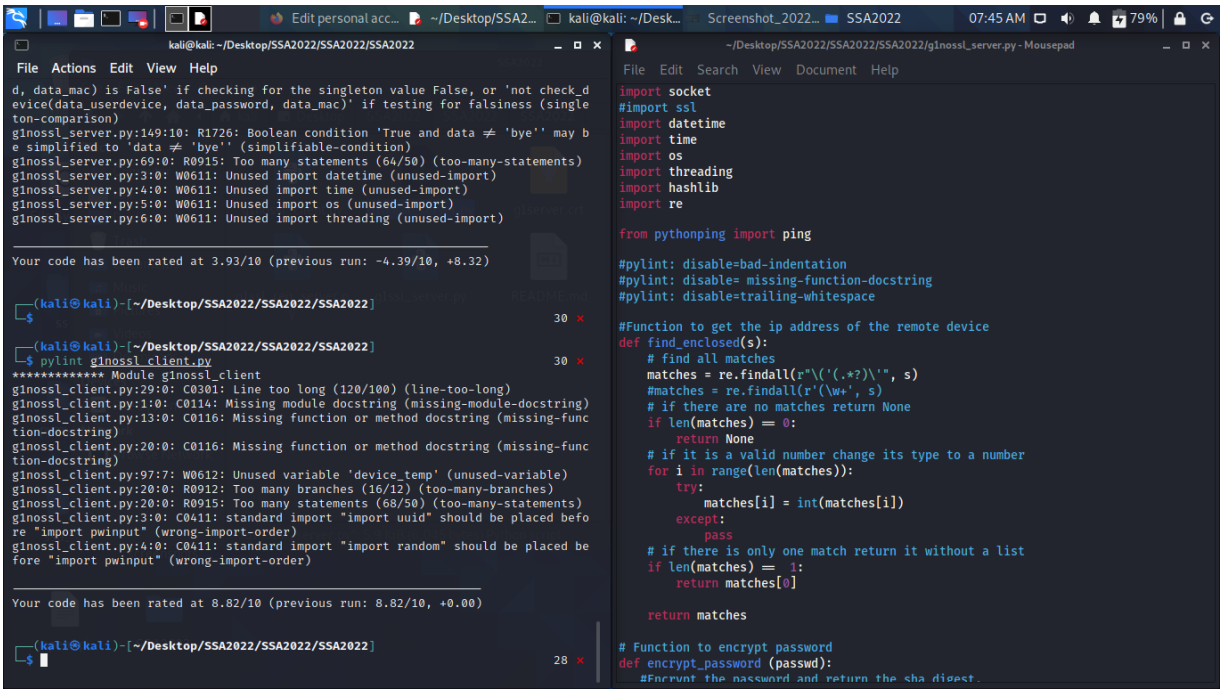


Figure 30: After Pylint g1nossl_server.py

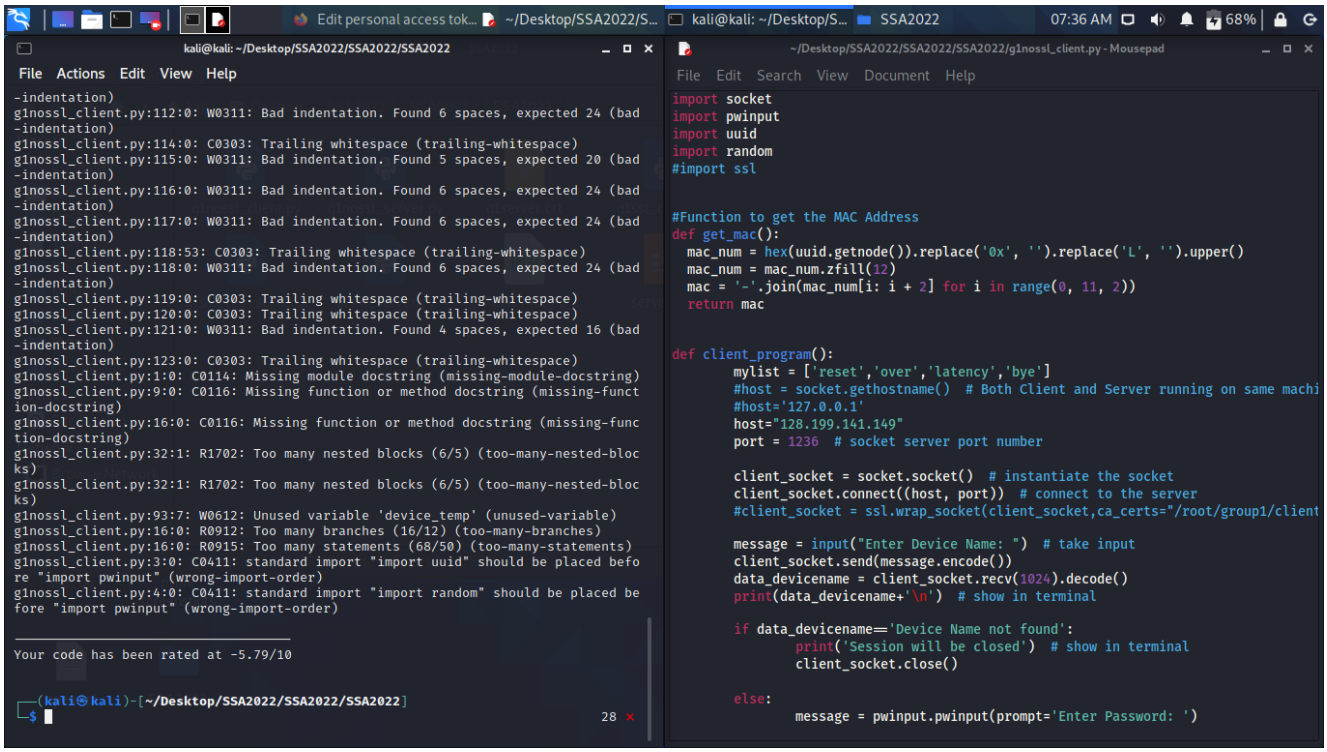


Figure 31: Pylint before g1nossl_client.py

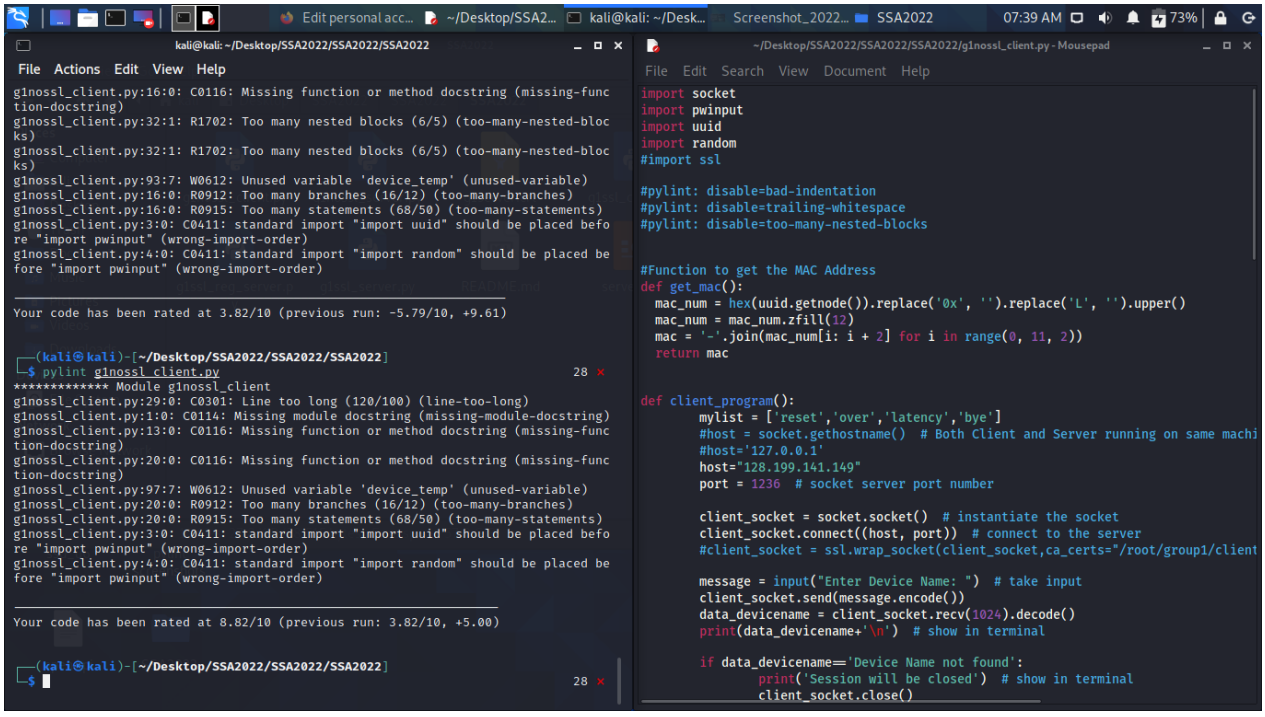


Figure 32: After Pylint g1nossl_client.py

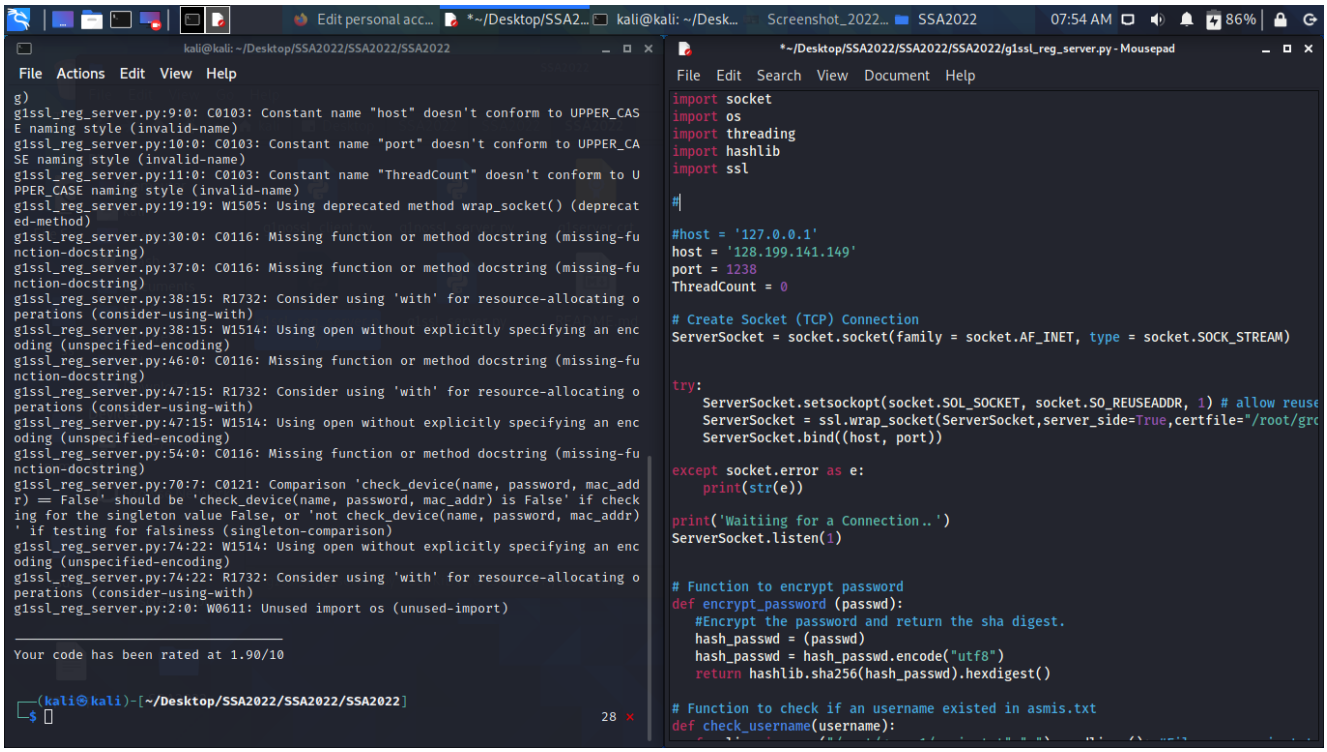


Figure 33: Pylint before g1ssl_server.py

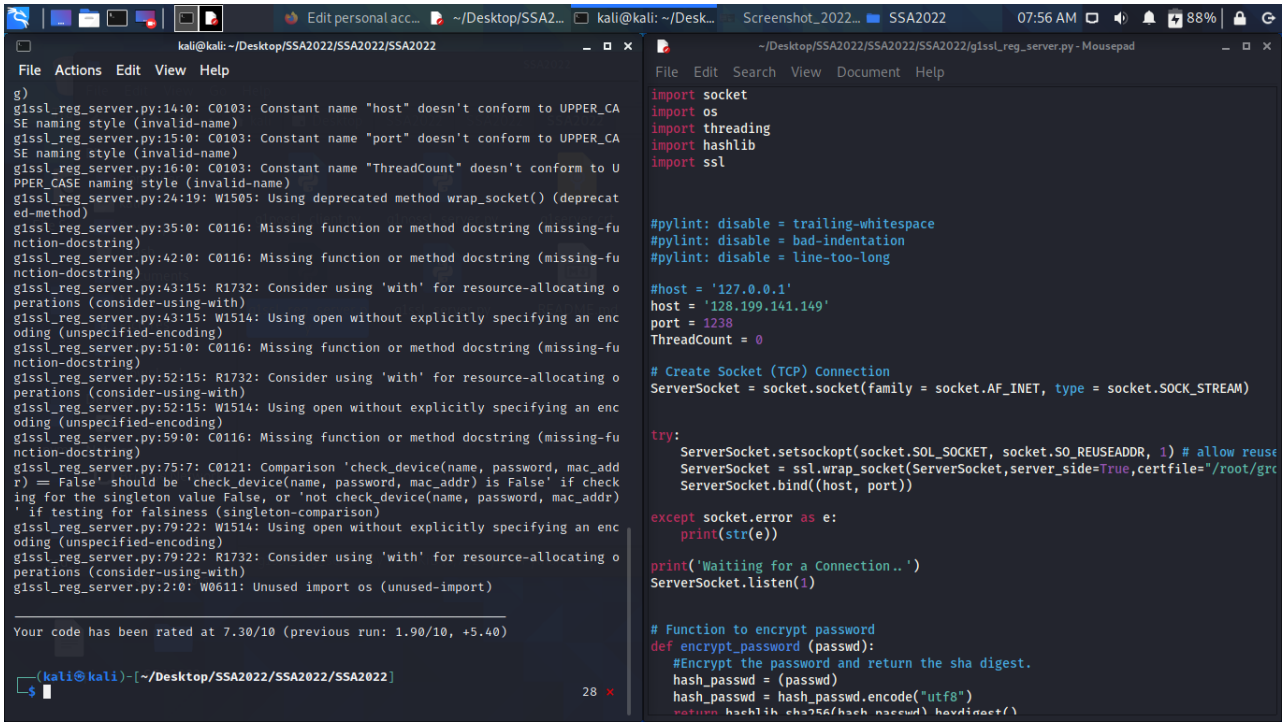


Figure 34: After Pylint g1ssl_client.py

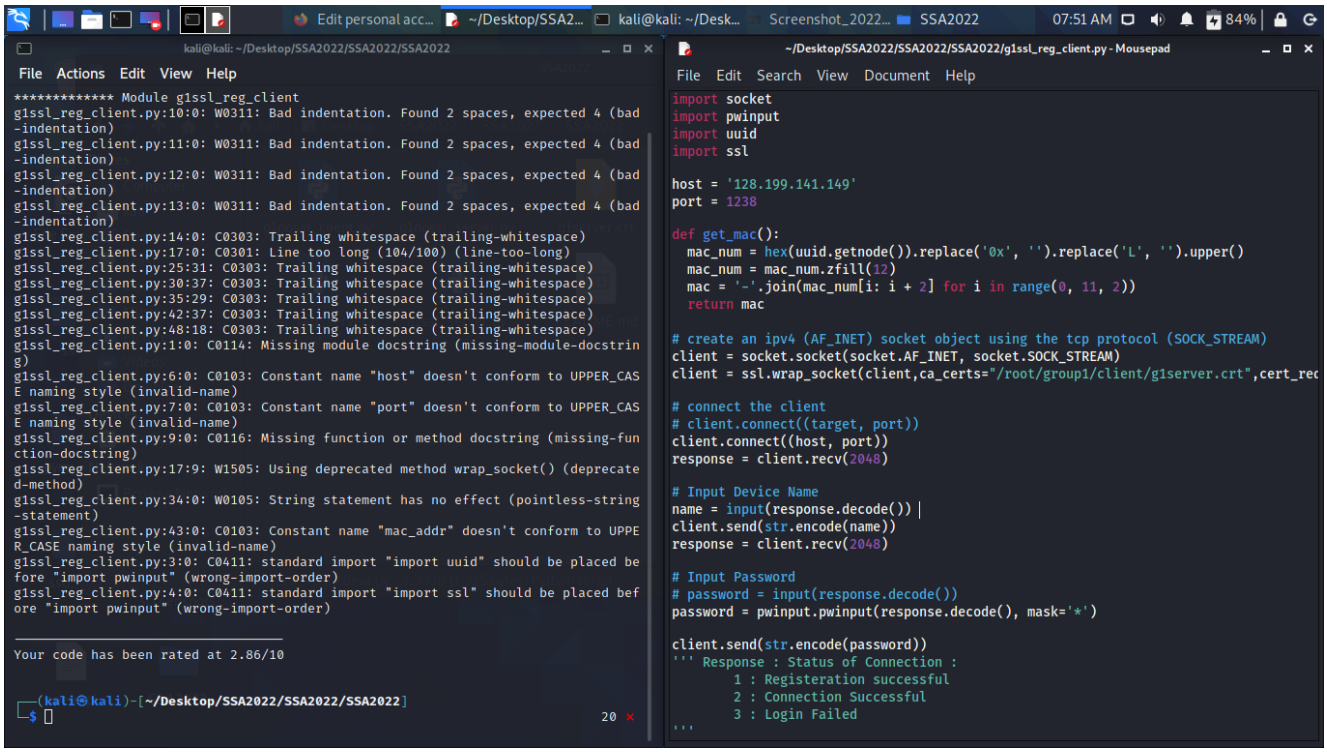


Figure 35: Pylint before g1ssl_reg_client.py

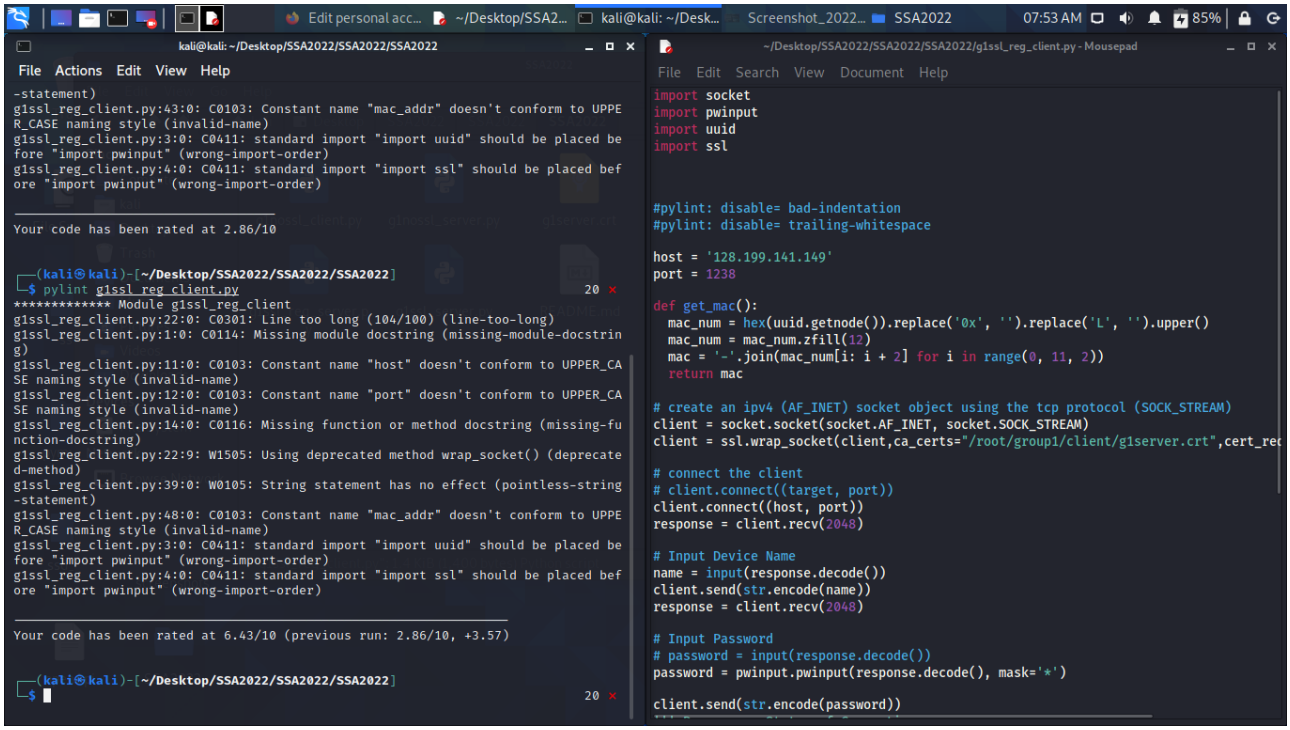


Figure 36: After Pylint g1ssl_reg_client.py

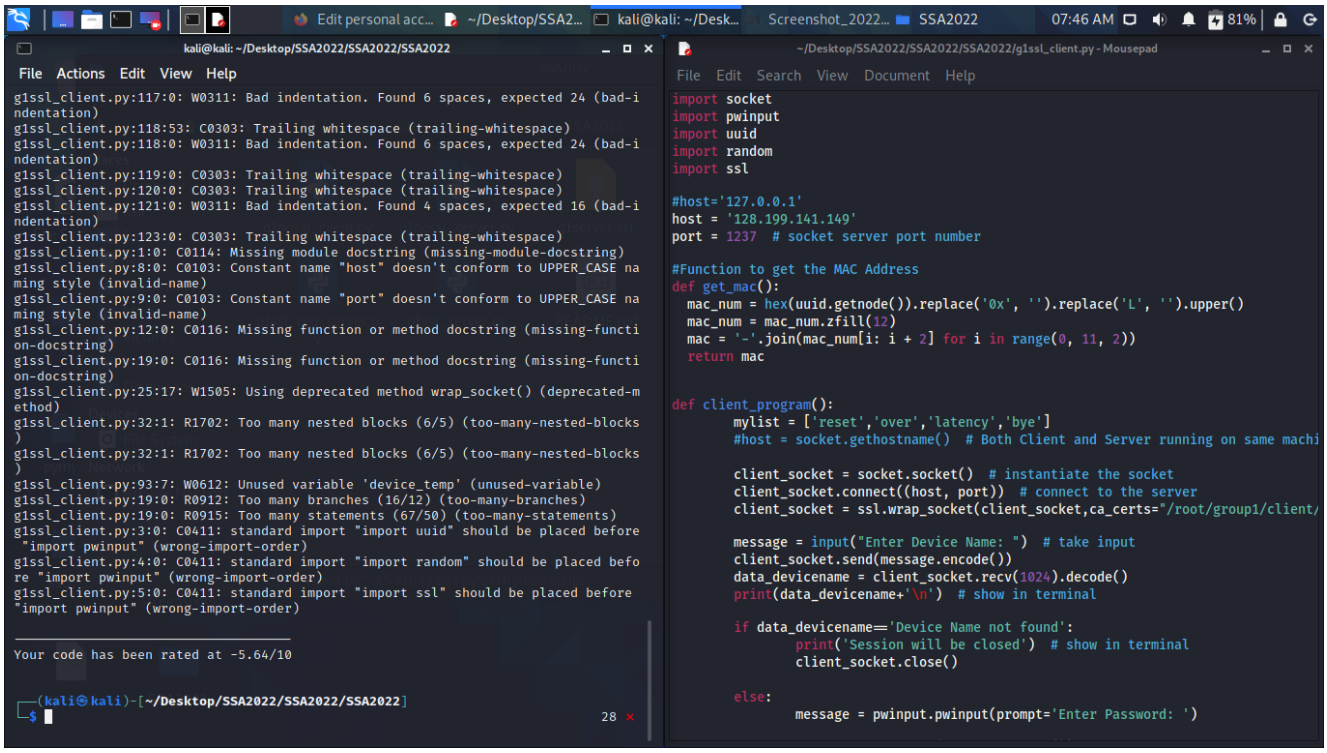


Figure 37: Pylint before g1ssl_client.py

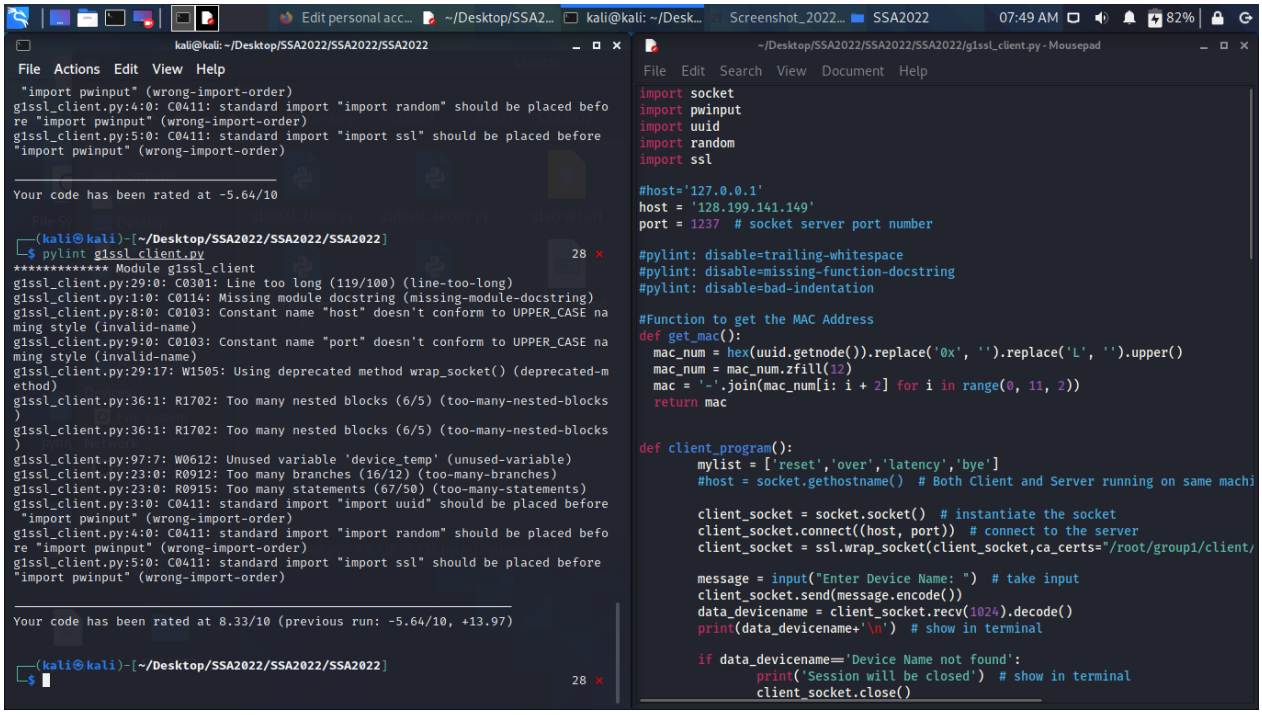


Figure 38: After Pylint g1ssl_client.py

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 g1nossl_server.py at the Server's end g1nossl_client at the Client's end:

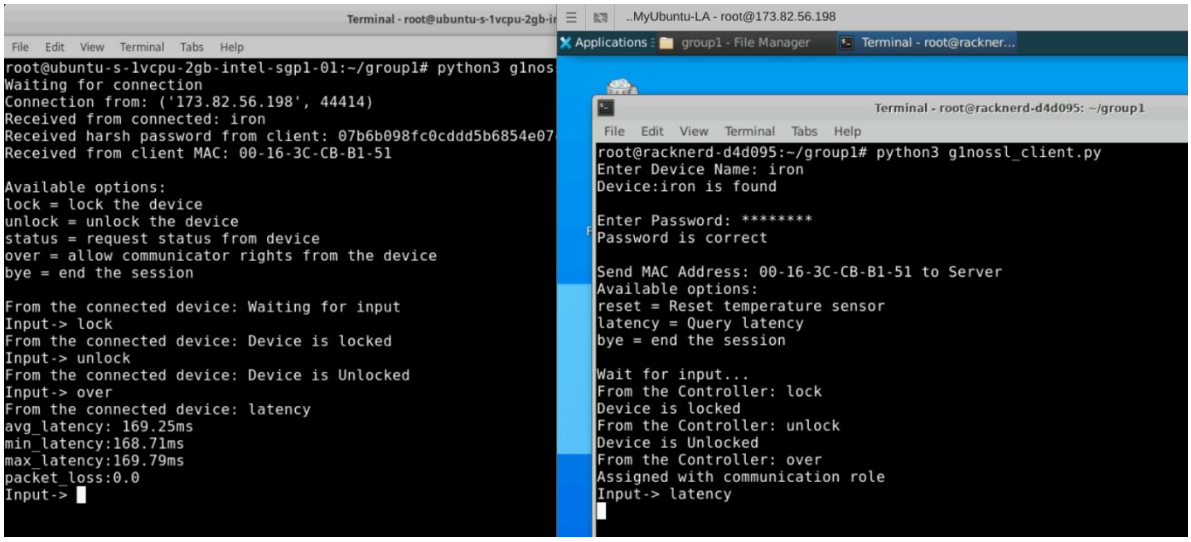


Figure 39: No Secure Sockets Layer and insecure connection

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

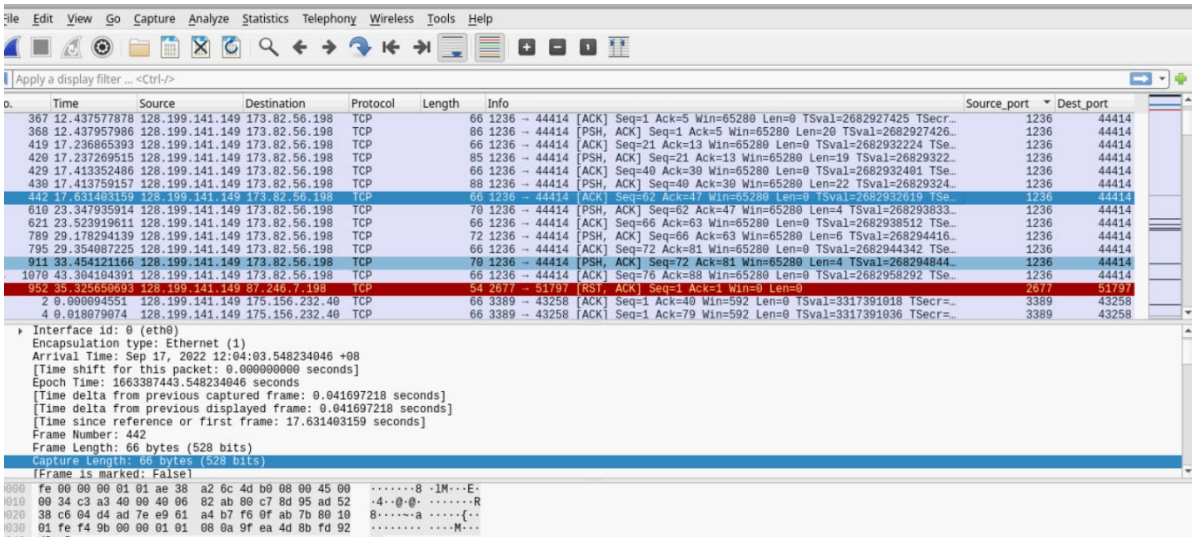


Figure 36: Wireshark confirmation of insecure connection

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:

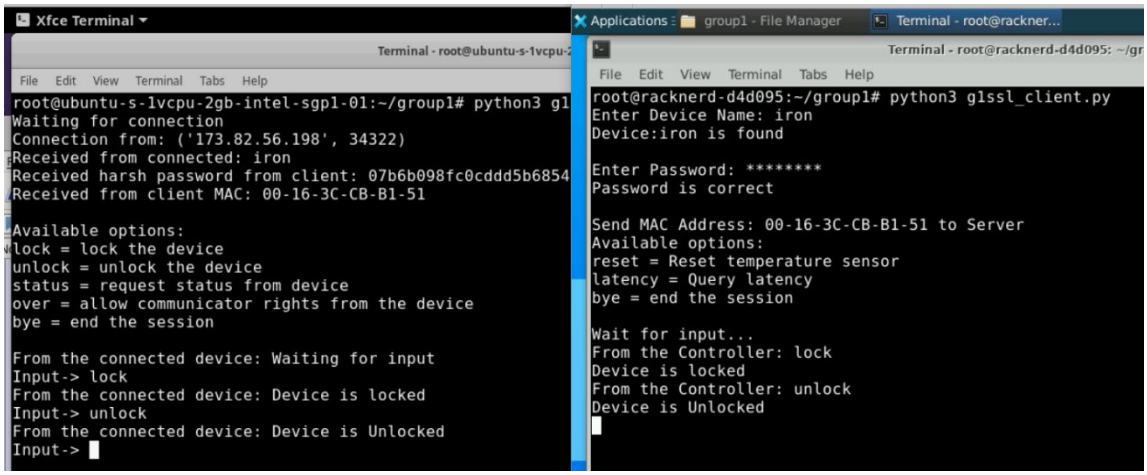


Figure 40: Secure Sockets Layer and secure connection

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

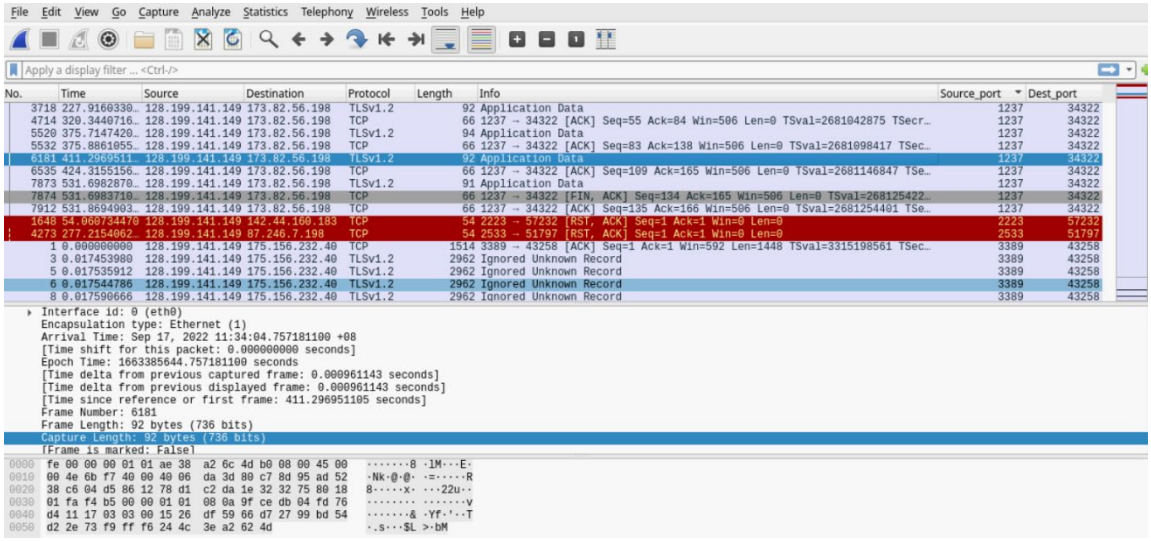


Figure 41: Transport Layer Security protocol enabled

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 incompleted. 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 Hämäläinen (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/ [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

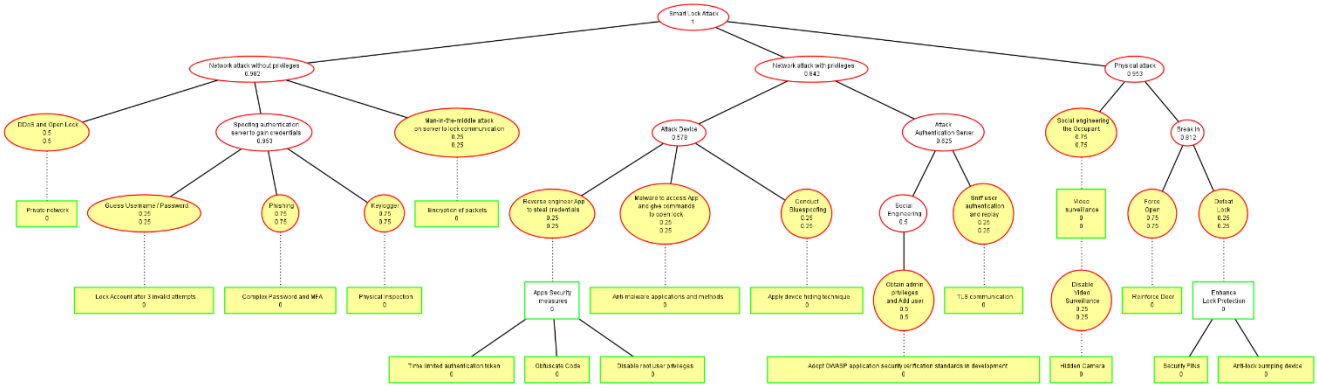


Figure 42: Attack-Defence Tree on Smart Door Lock from Design Document