Internet and Cybersecurity for Engineering Application

Final Project
Individual Report

Xuan Tuan Minh Nguyen
103819212

Table Of Contents

Introduction

1. Introduction

With the rapid rise of the network and the introduction of the Internet of Things (IoT) concept. Communicating between people and things seems to be much more simple as everything could be implemented and monitored through the internet without any physical attendance at the location where things and people reside. In addition, with more and more updates and features that are added in order to secure users from unintended attacks, the internet is considered one of the safe environments for users to connect and send information in and out without any concerns. There are lots of methods that aid users to connect, monitor and send commands to control mechanical things through the internet, however, one of the most popular methods that could be used to achieve this is by using Message Queuing Telemetry Transport, or in short, MQTT. MQTT is a lightweight, machine to machine protocol that helps connecting between devices that have resource constraints.

2. Report's objective

This report aims to focus on the MQTT protocol, which comprises the definition of the MQTT and the practice of using MQTT to send messages between devices. Moreover, this report will also point out some concerns about the MQTT in terms of security and some problems that the client could be faced with when deploying the system for public usage.

The Implementation of the project

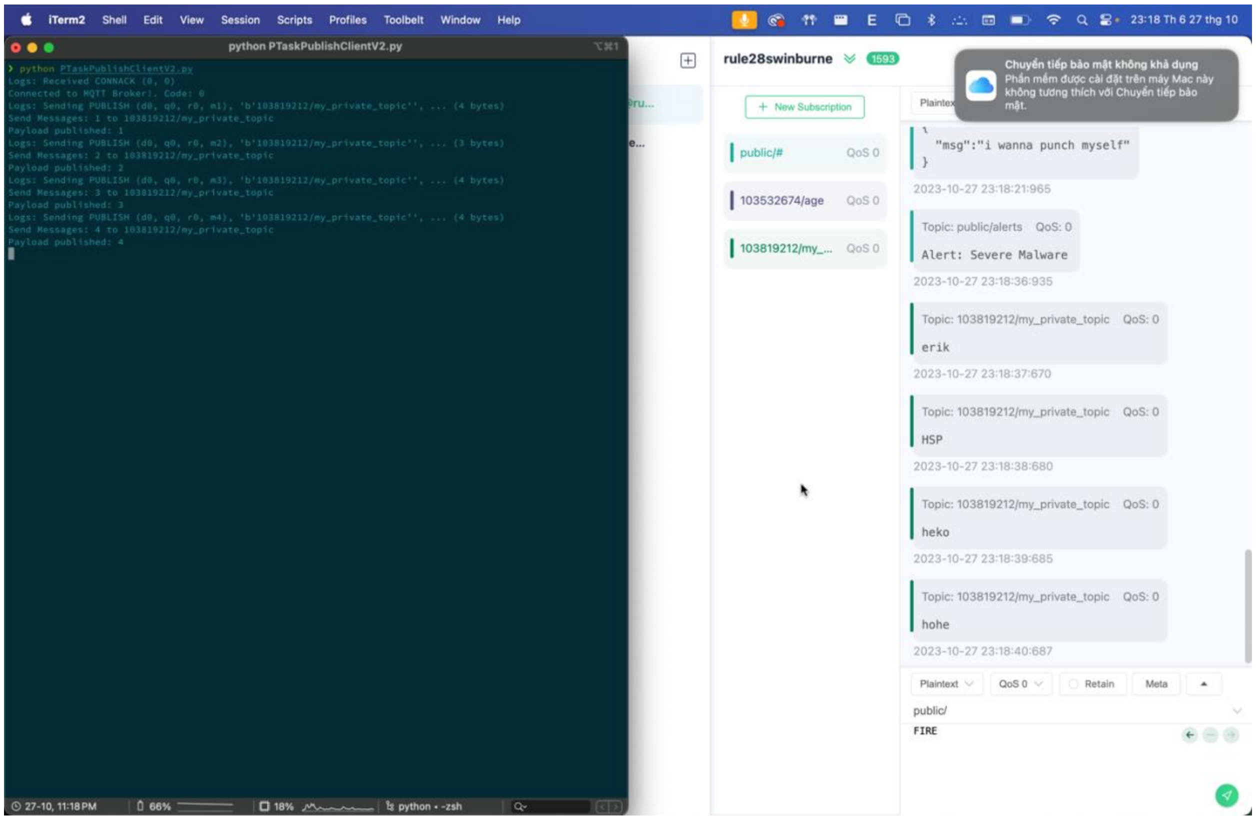
1. Message Queuing Telemetry Transport

Message Queuing Telemetry Transport, or MQTT, is a standard-based messaging protocol created by Andy Stanford-Clark (IBM) and Arlen Nipper (Eurotech). It is a lightweight, machine to machine, TCP/IP based and commonly used for sending messages and exchanging data between devices in harsh environments such as low-bandwidth, high-latency network environments and power-constrained environments. MQTT is widely used in IoT scenarios, especially for communicating between clients and the IoT devices that are located in remote locations that have limited bandwidth and lack of resources. This protocol is an event-driven protocol and uses the rule of publish/subscribe (Pub/Sub) pattern to gather devices together. The sender (Publisher) and the listener (Subscriber) will talk to each other via topics and they are decoupled from each other. Broker, or the middleware server that is used to handle the connection between publisher and subscriber, meaning that broker will be responsible for filtering all incoming messages from the publishers and distributing them to the right subscribers.

2. The implementation

2.1. Pass Task

For the implementation of the Pass task, one device must be the publisher, which generates data and posts messages to a private topic on the MQTT server while another device will act as a subscriber, meaning it will be responsible for fetching those data from both private and public topics. In order to achieve this task, I have used a Python library that helps me interact with MQTT brokers called paho-mqtt. In general, for the publisher side, I have opened a connection to the Swinburne broker server (rule28.i4t.swin.edu.au) in port 1883 with the username and password as my student ID. Then I have published the messages into the private topic (103819212/my_private_topic) using the publish() function and print the publish status to the terminal. On the client side, I also have connected to the broker server using the same implementations for the publisher side, I have subscribed to both of my private topics and all of the public topics. Finally, I print all of the retrieved messages into the terminal by modifying the built-in on_message function. In order to use the program, simply run the subscriber first (python/python3 PTaskSubscribeClient.py) and fire up the publisher (python/python3 PTaskPublishClient.py) to let it publish the messages into the private topics.



```
> python PTaskPublishClientV2.py
Logs: Received CONNACK (0, 0)
Connected to MQTT Broker!. Code: 0
Logs: Sending PUBLISH (d0, q0, r0, m1), 'b'103819212/my_private_topic'', ... (4 bytes)
Send Messages: 1 to 103819212/my_private_topic
Payload published: 1
Logs: Sending PUBLISH (d0, q0, r0, m2), 'b'103819212/my_private_topic'', ... (3 bytes)
Send Messages: 2 to 103819212/my_private_topic
Payload published: 2
Logs: Sending PUBLISH (d0, q0, r0, m3), 'b'103819212/my_private_topic'', ... (4 bytes)
Send Messages: 3 to 103819212/my_private_topic
Payload published: 3
Logs: Sending PUBLISH (d0, q0, r0, m4), 'b'103819212/my_private_topic'', ... (4 bytes)
Send Messages: 4 to 103819212/my_private_topic
Payload published: 4
```

rule28swinburne 1593

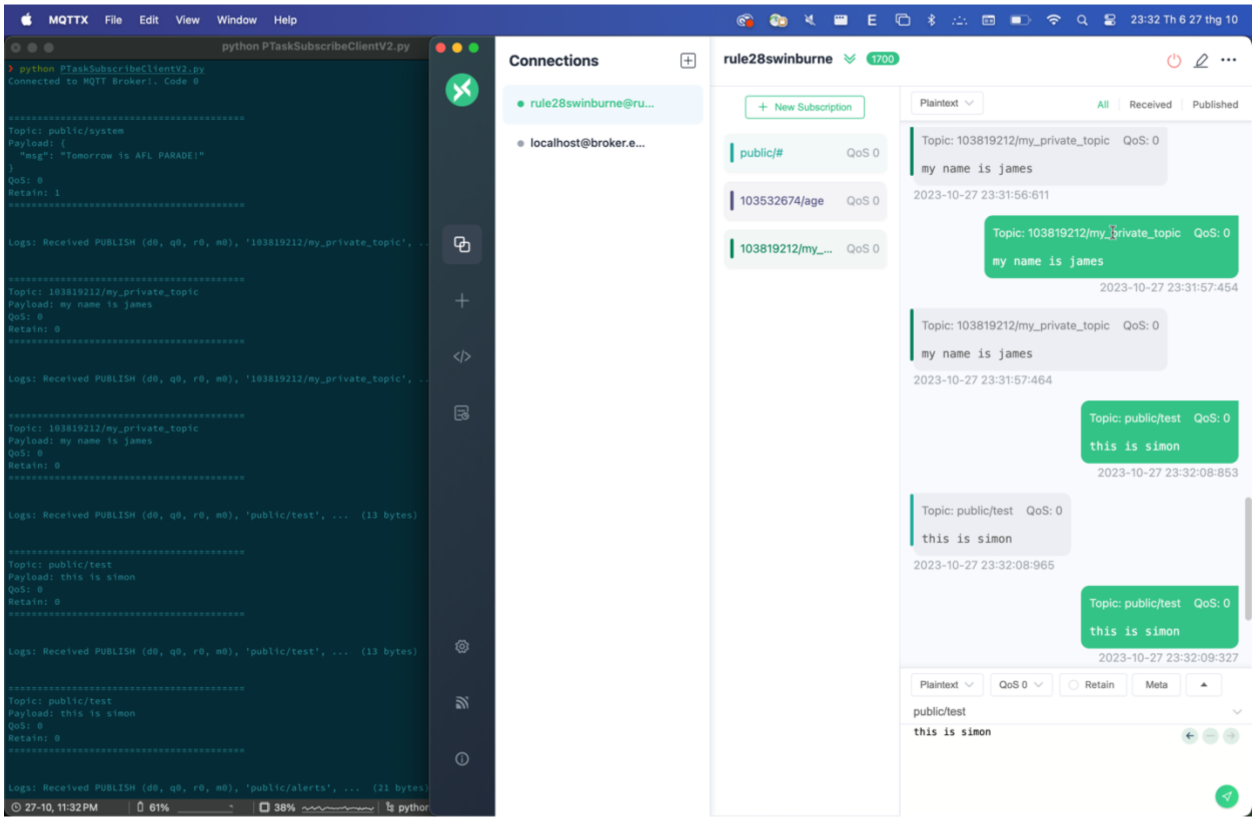
Chuyển tiếp bảo mật không khả dụng
Phần mềm được cài đặt trên máy Mac này không tương thích với Chuyển tiếp bảo mật.

public/#	QoS 0
103532674/age	QoS 0
103819212/my_...	QoS 0

```
{
  "msg":"i wanna punch myself"
}
2023-10-27 23:18:21:965

Topic: public/alerts   QoS: 0
Alert: Severe Malware
2023-10-27 23:18:36:935

Topic: 103819212/my_private_topic   QoS: 0
erik
2023-10-27 23:18:37:670

Topic: 103819212/my_private_topic   QoS: 0
HSP
2023-10-27 23:18:38:680

Topic: 103819212/my_private_topic   QoS: 0
heko
2023-10-27 23:18:39:685

Topic: 103819212/my_private_topic   QoS: 0
hohe
2023-10-27 23:18:40:687
```

Plaintext · QoS 0 · Retain · Meta
public/
FIRE

Figure 2.1.1: Graphical MQTT client received messages from device 1



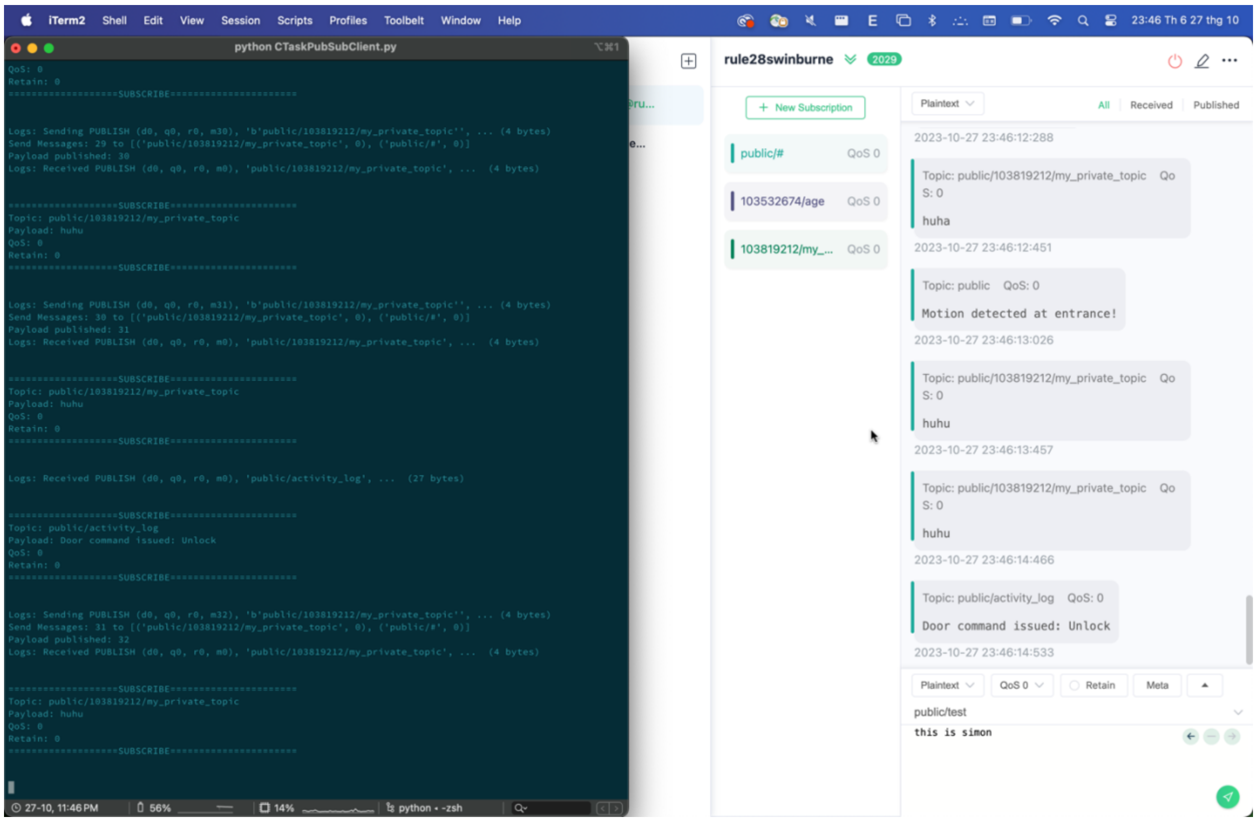
```
❯ python PTaskSubscribeClientV2.py
Connected to MQTT Broker!. Code 0


=========================================
Topic: public/system
Payload: {
  "msg": "Tomorrow is AFL PARADE!"
}
QoS: 0
Retain: 1
=========================================


Logs: Received PUBLISH (d0, q0, r0, m0), '103819212/my_private_topic', ..


=========================================
Topic: 103819212/my_private_topic
Payload: my name is james
QoS: 0
Retain: 0
=========================================


Logs: Received PUBLISH (d0, q0, r0, m0), '103819212/my_private_topic', ..


=========================================
Topic: 103819212/my_private_topic
Payload: my name is james
QoS: 0
Retain: 0
=========================================


Logs: Received PUBLISH (d0, q0, r0, m0), 'public/test', ...  (13 bytes)


=========================================
Topic: public/test
Payload: this is simon
QoS: 0
Retain: 0
=========================================


Logs: Received PUBLISH (d0, q0, r0, m0), 'public/test', ...  (13 bytes)


=========================================
Topic: public/test
Payload: this is simon
QoS: 0
Retain: 0
=========================================


Logs: Received PUBLISH (d0, q0, r0, m0), 'public/alerts', ...  (21 bytes)
27-10, 11:32PM | 61% | 38% | python
```

```
Topic: 103819212/my_private_topic   QoS: 0
my name is james
2023-10-27 23:31:56:611

Topic: 103819212/my_private_topic   QoS: 0
my name is james
2023-10-27 23:31:57:454

Topic: 103819212/my_private_topic   QoS: 0
my name is james
2023-10-27 23:31:57:464

Topic: public/test   QoS: 0
this is simon
2023-10-27 23:32:08:853

Topic: public/test   QoS: 0
this is simon
2023-10-27 23:32:08:965

Topic: public/test   QoS: 0
this is simon
2023-10-27 23:32:09:327

public/test
this is simon
```

Figure 2.1.2: Graphical MQTT client generated and send message to the device 2

2.2. Credit Task

The Credit task is more advanced compared to the Pass task, as one device must be able to act as both publisher and subscriber, meaning that it must be able to post the messages into the server and subscribe to a topic. Moreover, both must subscribe to the public topic. To implement this task, I have added the subscribe feature to the publish client, which allows it to do two jobs at the same time, which is receiving the message in the subscribed topics and generating messages to the appropriate topic. The procedure to run this program is fairly familiar with the previous task, which is fire up the subscriber first (python/python3 CTaskSubscribeClient.py) and then run the publisher/subscriber later (python/python3 CTaskPubSubClient.py)



```
QoS: 0
Retain: 0
===================SUBSCRIBE======================


Logs: Sending PUBLISH (d0, q0, r0, m30), 'b'public/103819212/my_private_topic'', ... (4 bytes)
Send Messages: 29 to [('public/103819212/my_private_topic', 0), ('public/#', 0)]
Payload published: 30
Logs: Received PUBLISH (d0, q0, r0, m0), 'public/103819212/my_private_topic', ...  (4 bytes)


===================SUBSCRIBE======================
Topic: public/103819212/my_private_topic
Payload: huhu
QoS: 0
Retain: 0
===================SUBSCRIBE======================


Logs: Sending PUBLISH (d0, q0, r0, m31), 'b'public/103819212/my_private_topic'', ... (4 bytes)
Send Messages: 30 to [('public/103819212/my_private_topic', 0), ('public/#', 0)]
Payload published: 31
Logs: Received PUBLISH (d0, q0, r0, m0), 'public/103819212/my_private_topic', ...  (4 bytes)


===================SUBSCRIBE======================
Topic: public/103819212/my_private_topic
Payload: huhu
QoS: 0
Retain: 0
===================SUBSCRIBE======================


Logs: Received PUBLISH (d0, q0, r0, m0), 'public/activity_log', ...  (27 bytes)


===================SUBSCRIBE======================
Topic: public/activity_log
Payload: Door command issued: Unlock
QoS: 0
Retain: 0
===================SUBSCRIBE======================


Logs: Sending PUBLISH (d0, q0, r0, m32), 'b'public/103819212/my_private_topic'', ... (4 bytes)
Send Messages: 31 to [('public/103819212/my_private_topic', 0), ('public/#', 0)]
Payload published: 32
Logs: Received PUBLISH (d0, q0, r0, m0), 'public/103819212/my_private_topic', ...  (4 bytes)


===================SUBSCRIBE======================
Topic: public/103819212/my_private_topic
Payload: huhu
QoS: 0
Retain: 0
===================SUBSCRIBE======================
```

```
2023-10-27 23:46:12:288
Topic: public/103819212/my_private_topic QoS: 0
huha
2023-10-27 23:46:12:451
Topic: public QoS: 0
Motion detected at entrance!
2023-10-27 23:46:13:026
Topic: public/103819212/my_private_topic QoS: 0
huhu
2023-10-27 23:46:13:457
Topic: public/103819212/my_private_topic QoS: 0
huhu
2023-10-27 23:46:14:466
Topic: public/activity_log QoS: 0
Door command issued: Unlock
2023-10-27 23:46:14:533
```

iTerm2 Shell Edit View Session Scripts Profiles Toolbelt Window Help
python CTaskPubSubClient.py
27-10, 11:46 PM 56% 14% python • -zsh
23:46 Th 6 27 thg 10
rule28swinburne 2029
New Subscription
public/# QoS 0
103532674/age QoS 0
103819212/my_... QoS 0
Plaintext All Received Published
Plaintext QoS 0 Retain Meta
public/test
this is simon

Figure 2.2.1: One device applications must publish and subscribe to a topic

Some security aspects that need to be concerned and solutions to overcome these concern aspects in the current deploying state of the application.

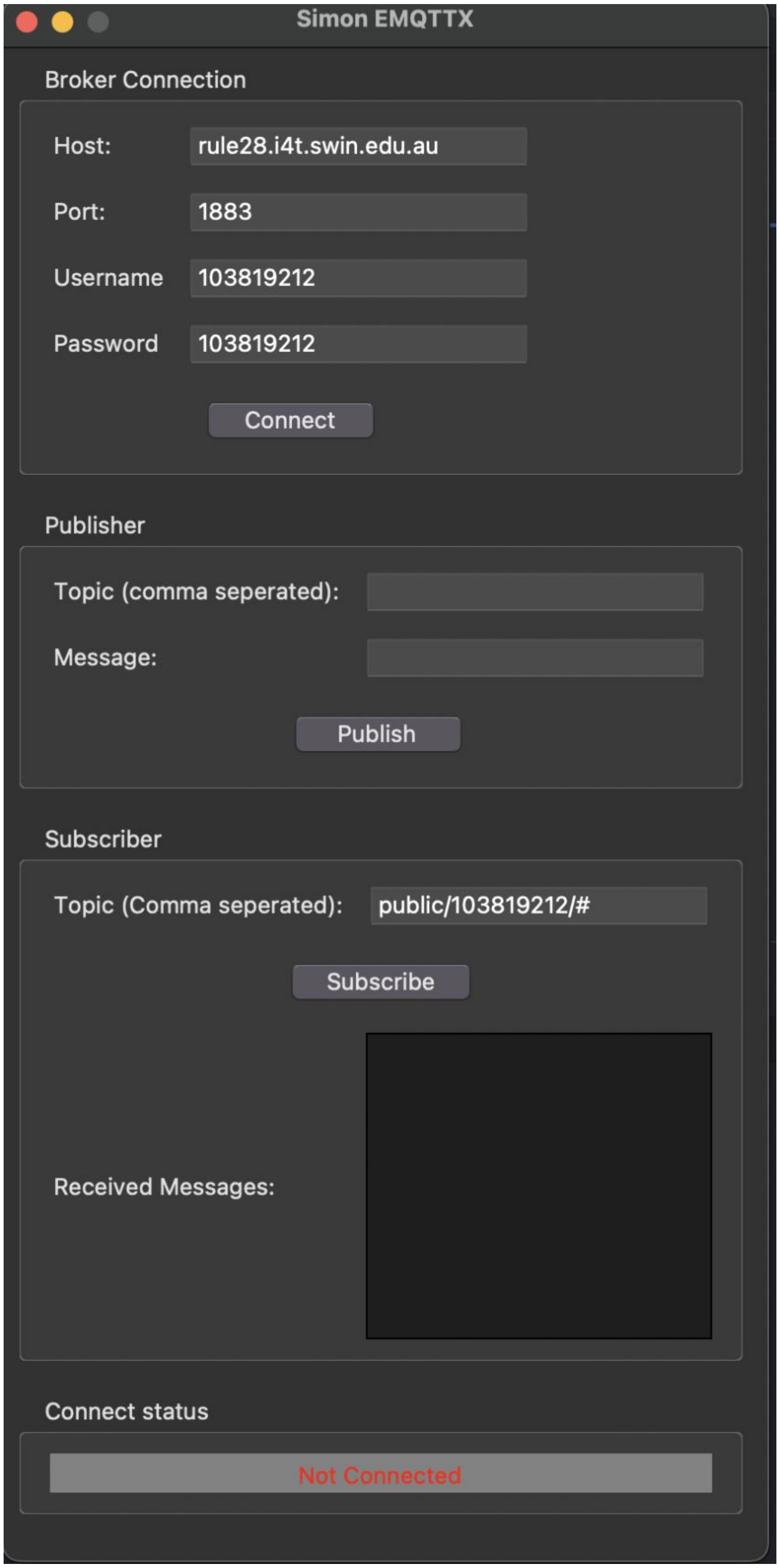
- ***The confidentiality and the integrity of the data***: At this state of the deployment, most messages are transmitted in the form of plain text or not fully encoded, which is a "rich land" for hackers to deploy eavesdropping and man-in-the-middle attack. Moreover, the routes for communicating between client and server are not securely encrypted, making it extremely vulnerable that the messages could be easily captured using some basic network sniffing tools such as: Wireshark, Bettercap, etc. To overcome this concern, using Transport Layer Security (TLS) to encode the communication between client and broker is the most efficient solution when it comes to encrypting the data in MQTT. This ensures that the data will be fully encrypted between client and server, making it difficult for hackers to decrypt the messages.
- ***Bad Authentication and Authorization:*** At this state, using the same username and password to access the broker without any authentication methods is not a practical way when it comes to authentication and authorization. As it will let hackers easily guess the password and access to the broker once they know the username. To solve this concern, using additional authentication methods, such as client certification or OAuth tokens to ensure that right clients are able to connect to the broker. Furthermore, building up the access control lists (ACLs)

on the broker to grant which clients are able to publish and subscribe to which specific topics.

- ***Broker Vulnerabilities***: Centralized brokers are the most targeted victim for the attacker, as all client devices could be threatened if the broker is compromised. The final project only relies on a single broker (rule28.i4t.swin.edu.au) for managing communication between devices, once the broker server is under control by the hacker, the hacker could drop, edit or inject messages on the server, thus able to terminate the whole system. One way to fix this is to create a redundancy mechanism to the server, which allows mitigating traffic into other brokers once there are any vulnerabilities on the broker. Moreover, the implementation for the security and the authentication of the broker is also recommended to reduce the broker's vulnerability chances.
- ***Low Quality of Service (QoS) Levels***: The Quality of Service levels must be appropriately defined based on the importance of the message. If the Quality of Service is not handled effectively, for instance using QoS level 0 in this project, could result in the loss of message during high traffic or network congestion. To overcome this problem, implementing the logic of the code to categorize the priority of the message to decide which kind of QoS level to be used for that message, particularly QoS level 2 is recommended for critical messages to ensure that the message is delivered after finishing the three-way handshake procedure.

2.3. Distinction Task

Distinction task is a whole new world, as not only it requires me to have at least two devices to subscribe to different topics and respond to various requests and posts, it also requires me to have at least one device to generate messages to more than one topic. Moreover, the most difficult part of this task is to replace the current graphical MQTT client (which is the MQTTX that I am using) with my own GUI client to monitor the system and generate or post the messages. For this task, I have used the most infamous GUI library in Python called Tkinter, which let me creates GUI in the simplest way. By using Tkinter, I have been able to create my own graphical MQTT client that has enough features to act as a GUI MQTT client, such as connect to the server, publish/subscribe messages to and from various topics and able to monitor the system.


Simon EMQTTX

Broker Connection

Host: rule28.i4t.swin.edu.au

Port: 1883

Username 103819212

Password 103819212

Publisher

Topic (comma seperated):

Message:

Subscriber

Topic (Comma seperated): public/103819212/#

Received Messages:

Connect status

Not Connected

Figure 2.3.1: Graphical MQTTX GUI Client

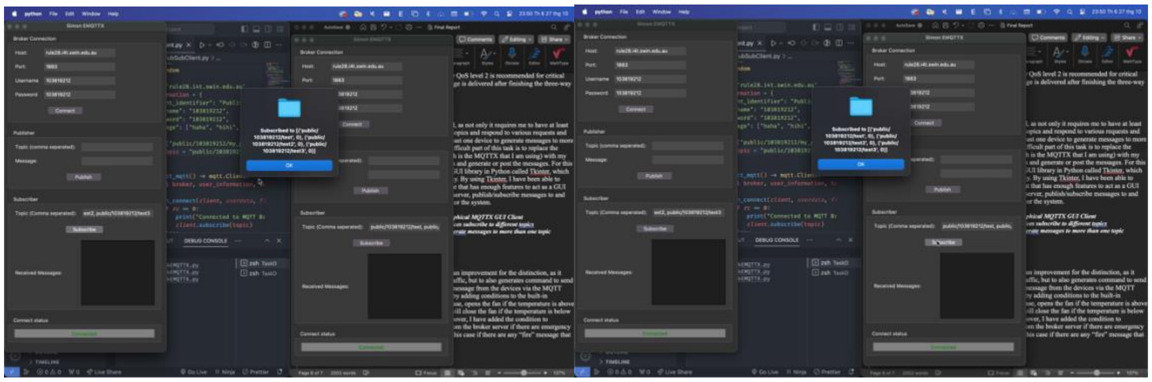
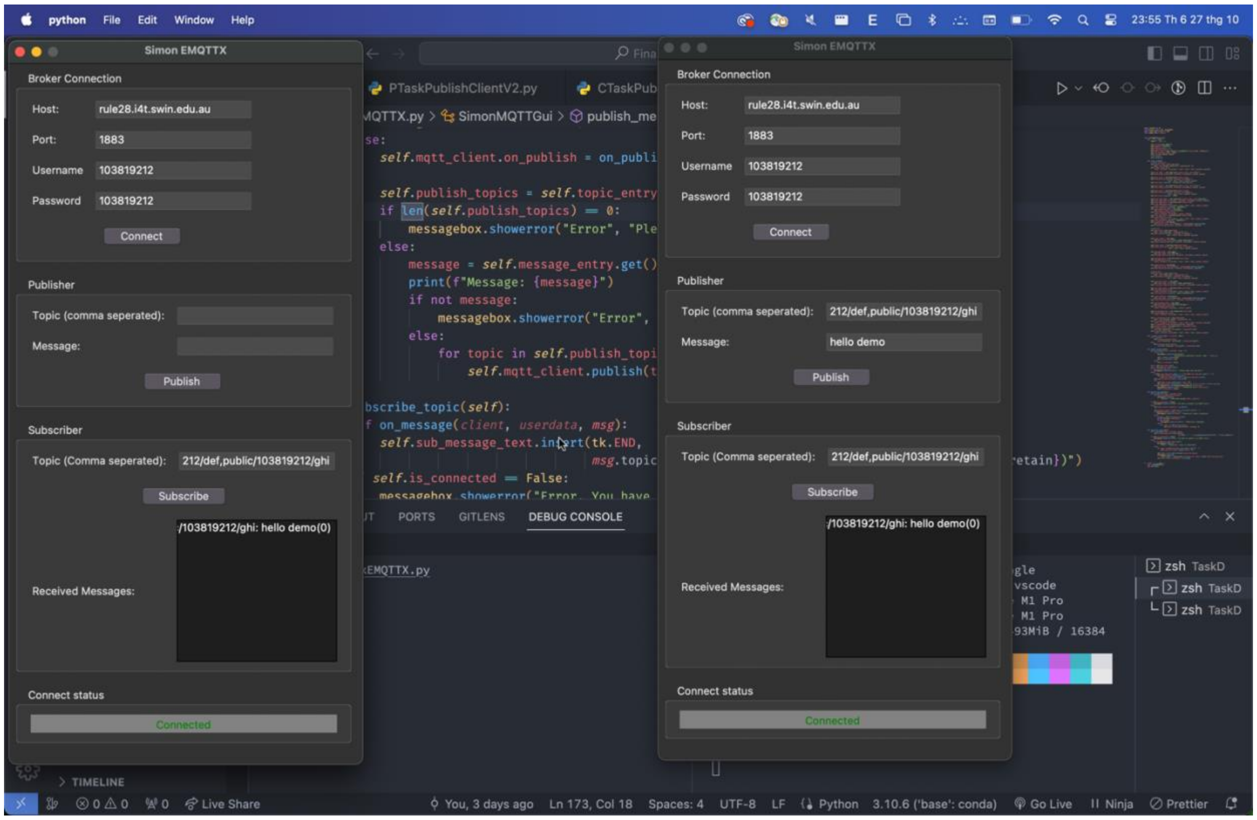


Figure 2.3.2: Two devices subscribe to different topics



Simon EMQTTX

Broker Connection
Host: rule28.i4t.swin.edu.au
Port: 1883
Username: 103819212
Password: 103819212

Publisher
Topic (comma seperated):
Message:

Subscriber
Topic (Comma seperated): 212/def,public/103819212/ghi

Received Messages: /103819212/ghi: hello demo(0)

Connect status: Connected

Simon EMQTTX

Broker Connection
Host: rule28.i4t.swin.edu.au
Port: 1883
Username: 103819212
Password: 103819212

Publisher
Topic (comma seperated): 212/def,public/103819212/ghi
Message: hello demo

Subscriber
Topic (Comma seperated): 212/def,public/103819212/ghi

Received Messages: /103819212/ghi: hello demo(0)

Connect status: Connected

```
se:
    self.mqtt_client.on_publish = on_publi
    
    self.publish_topics = self.topic_entry
    if len(self.publish_topics) == 0:
        messagebox.showerror("Error", "Ple
    else:
        message = self.message_entry.get()
        print(f"Message: {message}")
        if not message:
            messagebox.showerror("Error",
        else:
            for topic in self.publish_topi
                self.mqtt_client.publish(t

bscribe_topic(self):
f on_message(client, userdata, msg):
    self.sub_message_text.insert(tk.END,
                                 msg.topic
    self.is_connected = False:
```

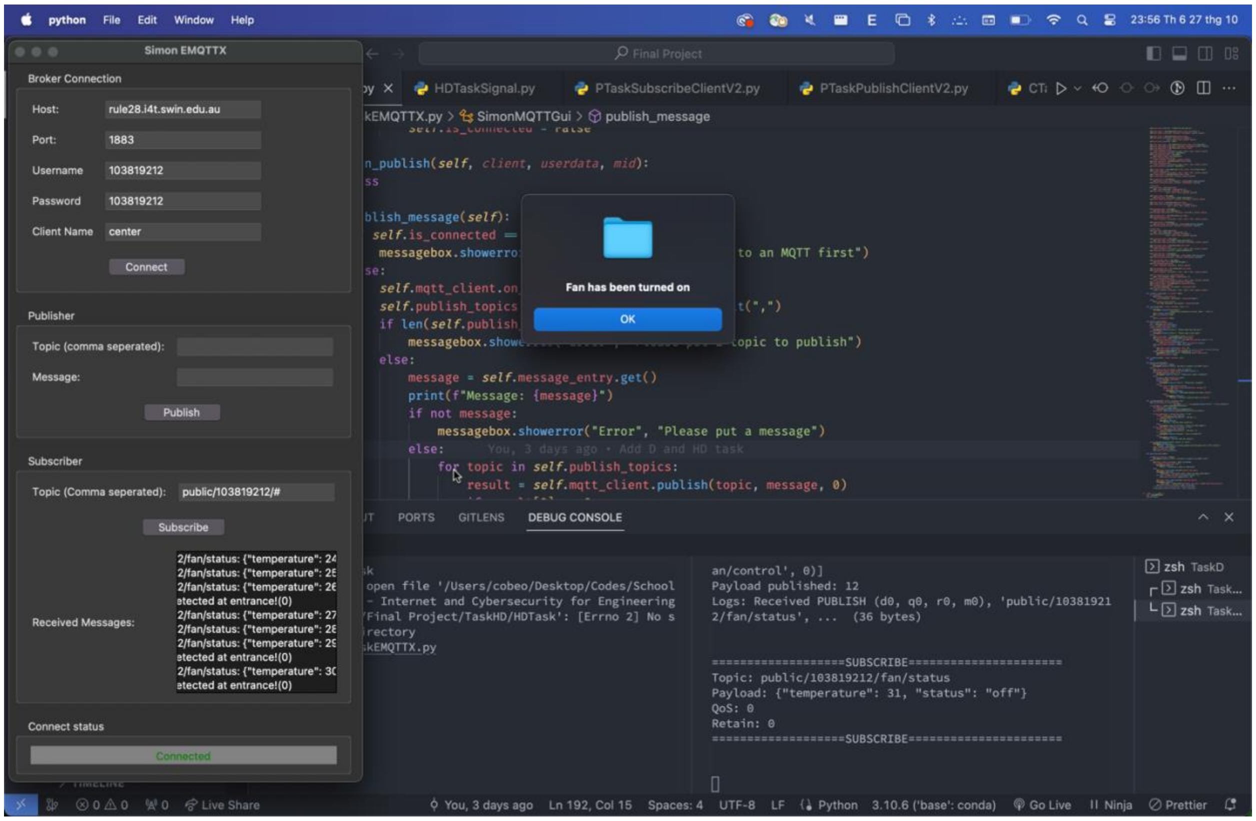
Figure 2.3.3: One device generate messages to more than one topic

2.4. High Distinction Task

The high distinction task is basically an improvement for the distinction, as it allows the client to not monitor the traffic, but to also generates command to send to the devices based on the received message from the devices via the MQTT broker. I have implemented this task by adding conditions to the built-in on_message function, which in this case, opens the fan if the temperature is above 30 and the fan is at the off state and will close the fan if the temperature is below 20 and the fan is at the on state. Moreover, I have added the condition to automatically disconnect the client from the broker server if there are emergency message generated by the broker (in this case if there are any “fire” message that are send onto the public topic)

In order to deploy the project to the public, there are some major issues that are essential to have a look and discuss some solutions to overcome those issues

- ***Denial of Service attack (DoS)***: Attackers could overload the system by sending multiple requests with tainted contents, causing the broker to slow down and crash thus interrupt the communications between linked devices. The solution for this issue is to limit the rate of sending data, monitoring traffic and using dedicated security methods to mitigate the traffic in case of having any DoS attacks.
- ***Man-in-the-Middle (MitM) attack***: The attacker could act as the “middle” user, which capture messages between client and broker. Moreover, by using MitM attack method, hacker could be able to modify the payload of the message, causing wrong commands to be executed or wrong message to be transmitted. The solution is to use the multiple authentication methods and fine-tuning the rules on the broker’s access control list (ACLs) to authenticate and providing permission to who could be able to send/receive messages from which topic to block all the unauthorized.
- ***Malware and malicious payload exposure:*** Malware or malicious code is inserted in the transferred data, these data will be send to the linked devices and the devices could get infected or compromised from those harmful payloads. In order to overcome this problem, validating and sanitizing all payloads as a measure of security. Monitor for known harmful patterns with intrusion detection systems could solve the problem.
- ***Replay attacks:*** An attacker detects valid data transmission and resends the data or "replays" it to achieve the authorized information. Those commands could be replayed to the right devices or systems, especially if the broker and client is lack of unique identification or timestamp. Adding a timestamp or a nonce in the message payload and adding logic to the code to check that each message is unique and cannot be replayed is a solution that could help to overcome this problem.


python File Edit Window Help
23:56 Th 6 27 thg 10
Final Project
HDTaskSignal.py
PTaskSubscribeClientV2.py
PTaskPublishClientV2.py
kEMQTTX.py > SimonMQTTGui > publish_message
PORTS GITLENS DEBUG CONSOLE
zsh TaskD
zsh Task...
zsh Task...
Live Share
You, 3 days ago Ln 192, Col 15 Spaces: 4 UTF-8 LF Python 3.10.6 ('base': conda) Go Live Ninja Prettier

Simon EMQTTX

Broker Connection

Host:	rule28.i4t.swin.edu.au
Port:	1883
Username	103819212
Password	103819212
Client Name	center

Connect

Publisher

Topic (comma seperated):	
Message:	

Publish

Subscriber

Topic (Comma seperated):	public/103819212/#

Subscribe

Received Messages:

```
2/fan/status: {"temperature": 24
2/fan/status: {"temperature": 25
2/fan/status: {"temperature": 26
etected at entrance!(0)
2/fan/status: {"temperature": 27
2/fan/status: {"temperature": 28
2/fan/status: {"temperature": 29
etected at entrance!(0)
2/fan/status: {"temperature": 30
etected at entrance!(0)
```

Connect status

Connected

Fan has been turned on

OK

```
n_publish(self, client, userdata, mid):
ss

blish_message(self):
 self.is_connected ==
  messagebox.showerro                      to an MQTT first")
se:
  self.mqtt_client.on
  self.publish_topics                      t(",")
  if len(self.publish
      messagebox.showe                     topic to publish")
  else:
      message = self.message_entry.get()
      print(f"Message: {message}")
      if not message:
          messagebox.showerror("Error", "Please put a message")
      else:      You, 3 days ago • Add D and HD task
          for topic in self.publish_topics:
              result = self.mqtt_client.publish(topic, message, 0)
```

```
k
open file '/Users/cobeo/Desktop/Codes/School
- Internet and Cybersecurity for Engineering
/Final Project/TaskHD/HDTask': [Errno 2] No s
rectory
kEMQTTX.py
```

```
an/control', 0)]
Payload published: 12
Logs: Received PUBLISH (d0, q0, r0, m0), 'public/10381921
2/fan/status', ...  (36 bytes)


==================SUBSCRIBE=====================
Topic: public/103819212/fan/status
Payload: {"temperature": 31, "status": "off"}
QoS: 0
Retain: 0
==================SUBSCRIBE=====================
```

Figure 2.4.1: MQTT GUI Client send command to open the fan