# SSA\_PCOM7E | DEVELOPMENT TEAM PROJECT - CODE DEVELOPMENT

----

The following should be installed already before setup.

- Python3 ([Download](https://www.python.org/) 3.6 or later.)

```

pip install -r requirements.txt

```

**\*\*\*Running in Linux or Windows from the downloaded directory\*\*\***

1. Run the server.py (server) file to begin:

```

python3 /PATH/server.py

```

1. Authenticate to the server, entering the password stored in readpwd.txt file. Python checks the contents of this file when authenticating the user.
   1. If the incorrect password is entered, the program displays a message and terminates cleanly
   2. If the correct password is entered, the server:
      1. authenticates the user
      2. connects to the broker (broker.emqx.io)
      3. pulls down the last message that the broker had stored, sent to it by the smart meter before disconnection, and displays it. This is possible as the smart meter sends its messages with the retain flag set to True and all messages being sent with QoS 1. Note that it also provides a calculation to pounds sterling to make it easier for the user to track expenditure.
      4. The server then sends a confirmation of this figure to the smart meter using topic UNIT1222, waits and listens for more messages.
2. To begin sending more data to the server, run the smart\_meter.py file:

```

python3 /PATH/smart\_meter.py

```

1. Once open, the smart meter:
   1. Connects to the broker
   2. Pulls the last confirmation message from the broker that was sent from the server and displays it both in organic encrypted payload form, and in decrypted human-readable form
   3. immediately begins transmitting (simulated energy usage) data using topic UNIT1221. It does this by generating a random number and adding it to the current running total received from the server in the previous step
   4. This data is sent to the broker, ready for the server to retrieve and repeat the process, starting at 2.b.iii again, setting the sequence in a continuous loop
2. To end the program, press Enter on the server screen

**Additional Notes**

* All messages are encrypted before being sent using a pre-shared key generated by fernet and inserted into both device scripts to enable encryption and decryption at both ends. This ensures the broker never knows the value of the plain text. It also means that it is impossible for the communications to be a victim of eavesdropping attacks due to the encryption key only being known by the server and the smart meter.
* Payload encryption is necessary due to the fact a public broker is used. If a private broker was available, then PKI would have been a better method to protect the data in transit using port 8883 and certificates. However, this is impossible due to the broker not being accessible. This is also a limitation for authenticating to the broker, and so authentication has been placed on the front end instead.
* Some advantages however, are that PKI would also add to a greater overhead in terms of packet size, and battery life will consequently be reduced. Additionally, by implementing authentication locally means that credentials are not sent over untrusted public communication.
* QOS is set to 1, this guarantees that all packets are sent (at least once). The fact that they are sent at least once is of no concern due to the retain flag only keeping the LAST message. Therefore, QOS level 2 would have meant unnecessary overhead. If QOS was set to 0, then the application may lose messages and therefore an unwanted incorrect reading may be given.
* The retain flag is not set for the confirmation communication from the server on topic UNIT1222 as it is important that the smart meter controls the running total alone on UNIT1221, this guarantees the loop is unidirectional. This is an attempt to resolve the consumer / producer problem (Tech Monger, 2018)
* If the server gets disconnected, the smart\_meter.py will wait until it is back up.
* The transmission loop will also be paused if the smart meter is disconnected. However, following reconnection, the server will need to be rebooted as the communications loop starts when the server launches and picks up the last stored message on topic UNITS1221 from the broker.
* Because the smart meter waits for the server, it is inconsequential in which order the devices are booted to begin the program again.

Decrypted meter output will be displayed on when successfully receiving unit data:

```

Total Units = 42

Total Cost (@£0.00039 per unit): £ 0.0164

Total Units = 48

Total Cost (@£0.00039 per unit): £ 0.0187

Total Units = 48

Total Cost (@£0.00039 per unit): £ 0.0187

Total Units = 52

Total Cost (@£0.00039 per unit): £ 0.0203

```

**Threats and Vulnerabilities**

See below threats, how they apply to the IoT network, and mitigation techniques.

| \*\*Threats\*\* | \*\*Vulnerable (Y/N)\*\* | \*\*Mitigation Details\*\* |

| -------------------------------- | -------------------- |------------------------------------------------------------------------------------------------------------------------------------|

| SQL Injection | N | \- No database is being used |

| Cross Site Scripting (XSS) | N | \- No front-end development |

| Broken Authentication | Y | \- User is prompted to enter a password when connecting to the broker, however it could be further stored in hashed or encrypted form<br/>\- Cipher key being used to encrypt and decrypt messages|

| Privilege Escalation | N | \- User is prompted to enter a password when connecting to the broker<br/>\- Minimal commands are allowable to the user (least privilege) |

| Session Hijack | Y | \- Public brokers do not support security certificates for authorisation |

| Man in the Middle | Y | \- Public brokers don’t support security certificates for authorisation |

| Wireless Key Compromise | N | \- Not applicable for this project |

| Eavesdropping | N | \- Although on an unsecured port (1833), cipher key being used to encrypt and decrypt messages provide confidentiality |

| Resource Exhaustion | N | \- If an incorrect password is entered, the program ends<br/>\- The broker only stored maximum 1 message |

| Port Scanning | Y | \- Using unsecured MQTT port 1833 for transmission as MQTT over SSL on port 8883 is not supported on public brokers |

| Malicious Code Scanning | Y | \- Malware tools are not being used |

| Connections from unknown devices | N | \- User is prompted to enter a password when connecting to the broker<br/>\- Cipher key being used to encrypt and decrypt messages |

| Access and Compromise of Devices | N | \- No physical devices |

A key consideration for this project was where the MQTT broker should be hosted publicly or privately. This determined which technical controls could be implemented to stemming from the MQTT broker’s inherited vulnerabilities. Private brokers can restrict connection to specific IoT devices. This can be used for both production and prototyping. On a public broker, any device can publish and subscribe to those topics. Because there is no privacy, public brokers should not be used for production, and instead be used for testing / prototyping (Chen, Huo, Zhu and Fan, 2020).

A level of confidentiality and availability was achieved by the project, however using a public MQTT broker meant leveraging preferred security controls such as certificates to ensure integrity, and additionally broker authentication was not possible. Because MQTT is a simple protocol designed for low-powered devices, it seeks to minimise the processing required to send messages, resulting in significant security issues. (Hernández Ramos et al., 2018). Most flaws can be fixed with proper protocol settings on the client and broker. On a private broker, additional security measures could mitigate threats to which the project is susceptible: For example, TLS with certificate credentials for all connections (providing secure authentication and authorisation), only TCP/IP ports 8883 would be open on the server (secure MQTT protocol) and limiting MQTT publishing and subscribing (prevent resource exhaustion and ensuring availability) (Kotak et al., 2019).

**References**

Cope, S. (N.D.) Practical MQTT with Steve. Available from: <http://www.steves-internet-guide.com> [Accessed 16 September 2022].

Chen, F., Huo, Y., Zhu, J., & Fan, D. (2020) ‘A Review on the Study on MQTT Security Challenge’, 2020 IEEE International Conference on Smart Cloud (SmartCloud). Washington DC, 6-8 November. IEEE. 128-133.

Hernández Ramos, S., Lacuesta, R. & Villalba, M. T. (2018) MQTT Security: A Novel Fuzzing Approach. *Wireless communications and mobile computing* 2018: 1-11. Available from: https://doi.org/10.1155/2018/8261746  
  
Kotak, J., Shah, A., Shah, A., & Rajdev, P. (2019) ‘A Comparative Analysis on Security of MQTT Brokers’, *2nd Smart Cities Symposium (SCS 2019)*. Bahrain, 24-26 March. Stevenage, UK: IET. p. 1-5.

Tech Monger. (2018) Solution to Producer Consumer Problem in Python. Available from: <https://techmonger.github.io/55/producer-consumer-python> [Accessed 19 September 2022].