**Title:** Smart Lock and Hub Testing Document

**Author(s):** Group 2

**Date:** 13 September 2022

**Contents**

[Functionalities and security test cases 3](#_Toc113984562)

[Test 1: Hub Image 3](#_Toc113984563)

[Test 2: Lock Image 4](#_Toc113984564)

[Test 3: Wake Word – Correct Details 5](#_Toc113984565)

[Test 4: Wake Word – Wrong Details 6](#_Toc113984566)

[Test 5: Functionalities - Unlock 7](#_Toc113984567)

[Test 6: Functionalities - Lock 8](#_Toc113984568)

[Test 7: Functionalities - Exit 9](#_Toc113984569)

[Test 8: Functionalities – Typed Wrongly 10](#_Toc113984570)

[Test 9: Writing of the Wake Word 12](#_Toc113984571)

[Test 10: Encryption of Wake Word 13](#_Toc113984572)

[Test 11: Generation of Encryption Key 14](#_Toc113984573)

[Test 12: Incorrect Hub/Lock Username/Password 15](#_Toc113984574)

[Test 13: Two Controller Hubs Running Simultaneously 17](#_Toc113984575)

[Code Syntax, Style and Complexity Testing 18](#_Toc113984576)

[Test 14: Pylint scanner 18](#_Toc113984577)

[Test 15: Testing with Flake8 Library 20](#_Toc113984578)

[Test 16: McCabe Cyclomatic Complexity Scanner 22](#_Toc113984579)

Functionalities and security test cases

1. Hub Image

**Test case description:** The aim of this test is to ensure that the hub image is built and run successfully.

**Steps:**

1. Open Terminal.
2. Change directory to hub subfolder.
3. Run: **docker build -t mqtt-hub-test .**
4. Run the hub container: **docker run -i -t mqtt-hub-test**

**Test outcome (Pass/Fail)**

Pass

**Remediation (if applicable)**

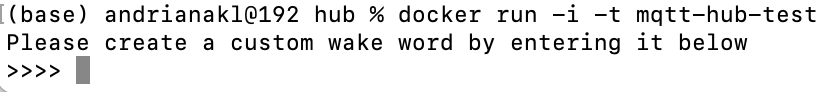
Not Applicable

**Test details**

The hub docker image was built in 12.8 seconds and ran successfully as seen below. When running the hub docker file, the output given is the very first step of the hub code.

**Text

Description automatically generated**

****

1. Lock Image

**Test case description:** The aim of this test is to ensure that the lock image is built and run successfully.

**Steps:**

1. Open Terminal.
2. Change directory to lock subfolder.
3. Run: **docker build -t mqtt-lock-test .**
4. Run the hub container: **docker run -i -t mqtt-lock-test**

**Test outcome (Pass/Fail)**

Pass

**Remediation (if applicable)**

Not Applicable

**Test details**

The lock docker image was built in 5.9 seconds and ran successfully as seen below. When running the lock docker file, the output given are messages showing the user that it has connected to the MQTT broker.

**Text

Description automatically generated**

**Text

Description automatically generated with low confidence**

1. Wake Word – Correct Details

**Test case description:** The aim of this test is to ensure that the hub asks new users to create a wake word. It then asks the user to enter the wake word. By entering the correct word it should output the three functionalities of the hub.

**Steps:**

1. Open Terminal.
2. Change directory to hub subfolder.
3. Run the hub container: **docker run -i -t mqtt-hub-test**
4. Create a custom wake word (for the sake of this test the wake word will be ‘wake’).
5. Type ‘wake’.

**Test outcome (Pass/Fail)**

Pass

**Remediation (if applicable)**

Not Applicable

**Test details**

As seen below, a custom wake word (wake) was created and correctly typed in. The hub then asks the user to choose a functionality, lock/unlock the door or exit the program.

**Graphical user interface, text

Description automatically generated**

1. Wake Word – Wrong Details

**Test case description:** The aim of this test is to ensure that the hub asks new users to create a wake word. It then asks the user to enter the wake word. By entering the wrong word it should output a denied access message.

**Steps:**

1. Open Terminal.
2. Change directory to hub subfolder.
3. Run the hub container: **docker run -i -t mqtt-hub-test**
4. Create a custom wake word (for the sake of this test the wake word will be ‘wake’).
5. Type ‘wakd’.

**Test outcome (Pass/Fail)**

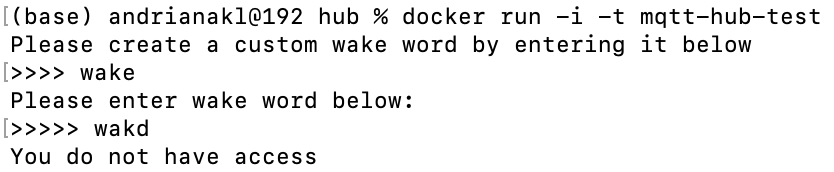
Pass

**Remediation (if applicable)**

Not Applicable

**Test details**

As seen below, a custom wake word (wake) was created and incorrectly typed in. The hub denies access to the user, due to the incorrect wake word and terminates the process.



1. Functionalities - Unlock

**Test case description:** The aim of this test is to ensure that the hub unlocks the door when asked by authorised users.

**Steps:**

1. Open Terminal
2. Change directory to subfolder hub
3. Run: **docker run -i -t mqtt-hub-test**
4. Create a custom wake word - ‘wake’.
5. Enter the wake word correctly.
6. Ask the hub to unlock by typing it correctly.

**Test outcome (Pass/Fail)**

Pass

**Remediation (if applicable)**

Not Applicable

**Test details**

The hub unlocks the door by letting the authorised user know that the door is unlocked and asks again whether the user wants to unlock, lock, or exit the program.

Text

Description automatically generated

1. Functionalities - Lock

**Test case description:** The aim of this test is to ensure that the hub locks the door when asked by authorised users.

**Steps:**

1. Open Terminal
2. Change directory to subfolder hub
3. Run: **docker run -i -t mqtt-hub-test**
4. Create a custom wake word - ‘wake’.
5. Enter the wake word correctly.
6. Ask the hub to lock by typing it correctly.

**Test outcome (Pass/Fail)**

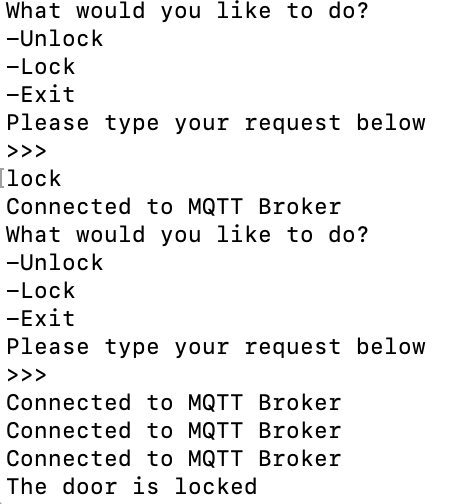
Pass

**Remediation (if applicable)**

Not Applicable

**Test details**

The hub locks the door by letting the authorised user know that the door is locked and asks again whether the user wants to unlock, lock, or exit the program.

****

1. Functionalities - Exit

**Test case description:** The aim of this test is to ensure that the hub exits the program when asked by authorised users.

**Steps:**

1. Open Terminal
2. Change directory to subfolder hub
3. Run: **docker run -i -t mqtt-hub-test**
4. Create a custom wake word - ‘wake’.
5. Enter the wake word correctly.
6. Ask the hub to exit the program by typing it correctly.

**Test outcome (Pass/Fail)**

Pass

**Remediation (if applicable)**

Not Applicable

**Test details**

The hub exits the program successfully when asked.

**Graphical user interface, text, application

Description automatically generated**

1. Functionalities – Typed Wrongly

**Test case description:** The aim of this test case is to confirm that if the user passwords do not match when trying to register, an error message will be presented.

**Steps:**

1. Open Terminal
2. Change directory to subfolder hub
3. Run: **docker run -i -t mqtt-hub-test**
4. Create a custom wake word - ‘wake’.
5. Enter the wake word correctly.
6. Ask the hub to either type incorrectly the:
   * 1. Unlock functionality.
     2. Lock functionality.
     3. Exit functionality.

**Test outcome (Pass/Fail)**

Pass

**Remediation (if applicable)**

Not Applicable

**Test details**

For all three cases, the hub prompts the user to enter a valid option when a functionality is typed in incorrectly.

1. Graphical user interface, text, application

   Description automatically generated
2. Graphical user interface, text, application

   Description automatically generated with medium confidence
3. **A picture containing graphical user interface

   Description automatically generated**
4. Writing of the Wake Word

**Test case description:** The aim of this test is to ensure that a wake word is written in a text file.

**Steps:**

1. Open Terminal
2. Change directory to subfolder hub
3. Run: **docker run -i -t mqtt-hub-test**
4. Create a custom wake word - ‘wake’.
5. Check if the wake word is in wake\_word.txt.

**Test outcome (Pass/Fail)**

Pass

**Remediation (if applicable)**

Not Applicable

**Test details**

The encryption process was omitted for this test to make sure the wake word was stored in the file. As seen below, the wake word ‘wake’ is stored in the wake\_word.txt file.

****

1. Encryption of Wake Word

**Test case description:** The aim of this test is to ensure that the wake word is encrypted.

**Steps:**

1. Open Terminal
2. Change directory to subfolder hub
3. Run: **docker run -i -t mqtt-hub-test**
4. Create a custom wake word - ‘wake’.
5. Check wake word file.

**Test outcome (Pass/Fail)**

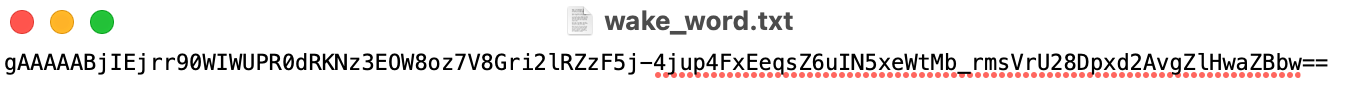
Pass

**Remediation (if applicable)**

Not Applicable

**Test details**

Wake word is encrypted as seen below.

****

1. Generation of Encryption Key

**Test case description:** The aim of this test is to ensure a successful generation of the encryption key.

**Steps:**

1. Open Terminal
2. Change directory to subfolder hub
3. Run: **docker run -i -t mqtt-hub-test**
4. Check filekey.key file.

**Test outcome (Pass/Fail)**

Pass

**Remediation (if applicable)**

Not Applicable

**Test details**

Encryption key is generated as seen below.

Graphical user interface, application

Description automatically generated

1. Incorrect Hub/Lock Username/Password

**Test case description** The aim of this test is to ensure that unauthorised users are not allowed to subscribe/publish to the broker.

**Steps:**

1. Alter one of the .txt file containing the hub/lock username and password
2. Open Terminal
3. Change directory to subfolder hub.
4. Run: **docker run -i -t mqtt-hub-test**
5. Check whether the hub is connected to the broker.

**Test outcome (Pass/Fail)**

Pass

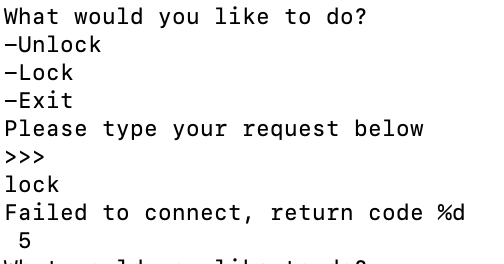
**Remediation (if applicable)**

Not Applicable

**Test details**

Cases:

1. Altered hub username:  
   Graphical user interface, text, application

   Description automatically generated
2. Altered hub password:  
   
3. Altered lock username:  
   Text

   Description automatically generated
4. Altered lock password:Text

   Description automatically generated
5. Two Controller Hubs Running Simultaneously

**Test case description:** The test aims to ensure the lock is still connected to the broker and the lock works.

**Steps:**

1. Open 3 Terminals.
2. 1st terminal: Change directory to subfolder hub
3. 2nd terminal: Change directory to subfolder hub
4. 3rd terminal: Change directory to subfolder lock
5. Run: **docker run -i -t mqtt-hub-test** on 1st and 2nd terminal
6. Run: **docker run -i -t mqtt-lock-test** on 3rd terminal
7. 1st terminal: add wake word and type lock option
8. 2nd terminal: add wake word and type unlock option

**Test outcome (Pass/Fail)**

Pass

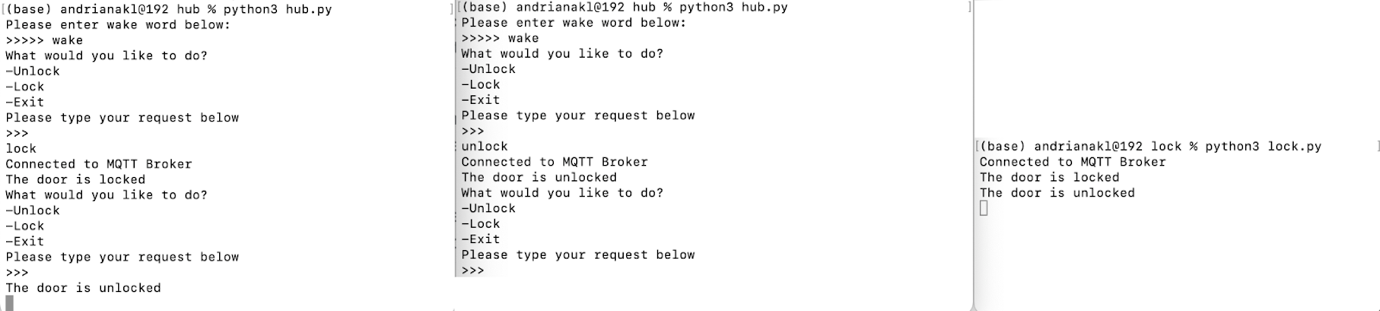
**Remediation (if applicable)**

Not Applicable

**Test details**

As seen below, when two hubs are running at the same time, the lock remains connected to the broker. When locking the door first from the one hub and unlocking it from the other, the lock locks and then unlocks.

Note: 1st terminal is on the left, 3rd on the right.



Code Syntax, Style and Complexity Testing

1. Pylint scanner

**Test case description:** The aim of this test is to identify any code styling issues by conducting a static code analysis with the help of Pylint scanner and to correct them.

**Steps:**

1. Install Python on the machine, using the command line.
2. Install Pylint on the machine, using the command line.
3. Make sure the Path in Environment variables is pointing to the scripts folder on Python.
4. Cd into the folder where src projects .py files are hosted.
5. Run the command ‘pylint file.py’ to perform the pylint scan on the file.
6. Fix the issues flagged up.

**Test outcome**

Several issues have been identified using pylint. Both files (hub.py and lock.py) were scanned with pylint.

hub.py:

**Text

Description automatically generated**

lock.py

Graphical user interface, application

Description automatically generated

**Remediation (if applicable)**

Most of the issues have been corrected by looking up the error codes on the internet. These included comments, indentations and line lengths. Some of the issues were omitted such as unused arguments in functions due to them being required for the code to work.

1. Testing with Flake8 Library

**Test case description:** The aim of this test is to scan the code base for any coding style (PEP8) and programming errors. Then, to correct any issues found.

**Steps:**

1. Download and install flake8 using the command line.
2. Navigate to the directory of Django project files.
3. Run flake8 file.py command
4. Correct mistakes flagged up by flake8.

**Test outcome**

hub.py

**Text

Description automatically generated**

lock.py

Graphical user interface, text, application

Description automatically generated

**Remediation (if applicable)**

Most of the errors found within the Flake8 tests were corrected. Errors regarding the length of comments were omitted as were needed for explanation of functions and classes

1. McCabe Cyclomatic Complexity Scanner

**Test case description:** The aim of this test is to scan the Python code base using the McCabe complexity scanner to find any overly complex code and correct it.

**Steps:**

1. Install mccabe.
2. Run mccabe against every file in the project folder.
3. Confirm that the complexity of each function is less than 10.

**Steps for macOS:**

1. Install flake8.
2. Run flake8 against .py files by setting the maximum complexity to 10 with flake8 --max-capacity 10 files.py
3. Confirm that there are no warnings/messages of a complex function.

**Test outcome (Scanner Score)**

**Hub.py:** Pass – the function complexity was less than 10 (Honglei et. al., 2009) on every Python file.

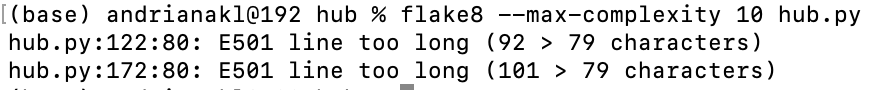
**Lock.py:** Pass- 3 warnings which were omitted

**Remediation (if applicable)**

Not applicable

**Test details**

Due to restrictions with macOS, a maximum complexity was set to 10 on the hub.py file using flake8, by running flake8 --max-complexity 10 hub.py. The results were as follow:



Note: these lines could not be separated or be simplified as they make more sense in that syntax. Separating the code would not make it easy to read.

References:

Honglei, T., Wei, S.  & Yanan, Z. (2009) ‘The Research on Software Metrics and Software Complexity Metrics’, italic: 2009 International Forum on Computer Science-Technology and Applications. Chongqing, China, 25-27 December. New York: IEEE. 131-136.