README FILE

SmartHome Controller/Light

1. DISCUSSION - VULNERABILITY MANAGEMENT

This document discusses a prototype that will demonstrate the operation of a smart home system, by focusing only on the operation and interaction between a simulated controller and simulated lighting and motion sensors. It will also discuss the security vulnerabilities associated with the system, and explore and propose mitigation steps to produce a secured IoT smart home infrastructure.

The solution uses a MQTT protocol in Python (paho client) which uses a broker to facilitate communication between subscribers and publishes. MQTT will be used for sharing and reacting to sensor information like motion and light levels.

Hintaw et al. (2021) states that:

MQTT uses a publisher/subscriber model to facilitate messaging between devices making messaging lightweight. Nevertheless, there are a number of security issues due to the design of the protocol itself. Some of the issues are denial of service, identity spoofing, information disclosure, the elevation of privileges, and data tampering (Hintaw et al, 2021).

Table 1.1 below, as from (Mlambo et al., 2023), shows a STRIDE table depicting attack vectors and vulnerabilities identified within the MQTT protocol used in the system and provides mitigation techniques to remedy the identified vulnerabilities.

Threat Type	Type of Attack or Vulnerability	Mitigation Techniques
Spoofing Identity	• Control or unauthorized access (Janes et al, 2020) • Escalation of privileges (Rizvi et al, 2020)	• Implement authorized access with multi factor authentication • Enable audit trials

Tampering with Data	• Data exfiltration (Vaccari et al, 2021)
	• Data Manipulation (Bhattacharjee et al, 2017)
	• Control over database (Cooper, J and James, A. 2009)
		• Access control
		• Input validation
		• Encryption of Data
		  ◦ At rest
		  ◦ In transit
		  ◦ upon access
		• apply a defence in depth approach
		• Define security requirements

Repudiation	• Validate system owner/user (Cruz-Piris et al, 2018) • Validate input (Redini et al, 2021)	• Apply a form control list to system access • Apply Validation of output data owner • Apply Secure Socket layer (SSL) Certificate

Information disclosure	• System providing Following type of info :
	  ◦ Operation system in use (Abomhara, M and Koien, G. 2015)
	  ◦ IP address
	• SQL injection (Tweneboah et al, 2017)
	• Data breach
	• Insecure data storage (Ahmad, J and Rajan A.V. 2016)
	• insecure data transfer communication (Shin, S. and Seto, Y. 2020)
		• Limit the amount of information that the system can provide when scanned
		• Limit displaying the output where not needed to
		• Define system security requirements

Denial of Service	• UDP ,ICMP, SYN and HTTP Flood (Gupta et al, 2022) • DDos Attack (Kolias et al, 2017) • DNS Amplification (Arthi, R. and Krishnaveni, S. 2021) • Application layer control	• Implement appropriate authentication and authorisation mechanisms in the solution • Implement proper Access Control
Elevation of privileges	• Exploiting software vulnerabilities (Cam-winget, N et all 2016) • Bypassing authentication methods (Jiang et al, 2018) • Social engineering (Ghasemi et al, 2016)	• Implement least privilege • Apply appropriate patch management practices while adhering to regular patch cycle. • Apply Logging and monitoring controls. • Utilise proper Network Segmentation • Apply proper encryption

One of the characteristics of the smart home system is that of a Distributed system and employs a microservices architecture, it is imperative to have a holistic view of the vulnerabilities that will have an impact on the system and it is important to look at the challenges associated with distributed systems and provide solutions to the challenges.

A distributed system contains multiple nodes that are physically separate but linked together using the network. All the nodes in this system communicate with each other and handle processes in tandem. Each of these nodes contains a small part of the distributed operating system software (Meador, 2020)

Table 1.2 below outlines the challenges that are associated with distributed systems and in turn, how MQTT helps in addressing these challenges.

LIMITATIONS	REMEDIATION
SECURITY	MQTT support TLS/SSL to encrypt between device and broker and control access for each device. In addition to this, not much security is built-in MQTT, positive side it allows to build security on top of it to cater for evolving and different IoT devices (Hübschmann, 2021).
USABILITY- LOW MEMORY & PROCESSOR DEVICES	MQTT uses lightweight protocol and has been able to support the increasing number of small, cheap, and low-powered IoT devices on the market that have low memory and low processing power (Hübschmann, 2021)
RELIABILITY + SCALABILITY	MQTT protocols are reliable because they employ QoS that guarantees delivery of messages and the publish-subscribe model helps scale well as there need not be direct communication between the publisher and the subscriber (Samarth, 2021).
POWER CONSUMPTION	MQTT protocol is lightweight and boasts a limited code footprint. This makes it ideal for low-power devices and those with limited battery lives, including now-ubiquitous smartphones and ever-growing numbers of sensors and connected devices (Matthews, 2020)
BANDWIDTH + LATENCY	MQTT protocol consists of publishing and subscription operation, it is a very simple and ultra-lightweight designed specifically for devices with limited resources and low bandwidth and need a high-latency and work in unreliable networks (Yudidharmaa et al, 2022)
TIMING	MQTT allows real-time data sharing between devices, this is perfect for applications where timing is crucial (Asim, 2022).

Table 1.2: Distributed Systems Challenges and Remediations

Although there is no standard or framework relating to security smart homes, The OWASP IoT top 10 is scrutinized to further provide guidance on identifying and remediation vulnerabilities in the smart home system.

The system uses an IoT networking model, so it is fitting to use the OWASP IoT Top 10 standard for developers and web application security as it represents a broad consensus about the most critical security risks to web applications.

Table 1.3. below lists the top 10 IoT vulnerabilities and mitigation actions associated with them.

OWASP IoT Top 10 for Proactive Security	
1	Weak, Guessable, Or Hardcoded Passwords
Mitigation	Have a unique set of credentials for each device
2	Insecure Network Services
Mitigation	Disallow connection with high risks networks like public Wi-Fi
3	Insecure Ecosystem Interfaces
Mitigation	Authenticate and authorize IoT endpoints
4	Lack of Secure Update Mechanism
Mitigation	Verify the source and integrity of updates with digital signature
5	Use of Insecure or Outdated Components
Mitigation	Replace legacy technologies
6	Insufficient Privacy protection
Mitigation	Store only necessary information and ensure end-to-end security
7	Insecure Data Transfer and Storage
Mitigation	Encryption of data when at rest, in transit or during processing
8	Insecure Default Settings
Mitigation	Secure decommissioning and monitoring of assets
9	Lack of Device Management
Mitigation	Compel users to change default passwords after device installation
10	Lack of Physical Hardening
Mitigation	Validate firmware with secure boot

Table 1.3: OWASP IoT Top Ten for proactive security (Basatwar, 2021)

Table 1.4 - 1.7, as from (Mlambo, et al., 2023), shows the current features of the system that makes it to be vulnerable and the mitigations that can be applied also taking in consideration vulnerabilities associated to the MQTT protocol, Distributed system

challenges, and IoT OWASP top 10 (as referenced from (Touqeer, et al., 2021), (Borgini, 2021), (Apriorit, 2022), (Anand, et al., 2020), (Abdullah, et al., 2019)).

Features of the Current System	Risks Accompanied	Potential Vulnerabilities	Possible Mitigations
It relies solely on digits on the phone's keypad to access the security system	• Unauthorized access. • Spoofing • Man-in-the-middle Attacks • Installation of malicious software • Fines and lawsuits that could lead to damaged reputations, bankruptcy and losses	• Lack of Multi-Factor Authentication • Lack of authorization • Unencrypted communication • Not enough security enforcing features • Lack of data privacy and certified compliances like GDPR, ISO 27001, ISO 27017, ISO 27018, etc	• Multi-Factor Authentication • Implement changing of passwords • Implement complex passwords • Limit number of log-in attempts • User Access controls • Authorizations • Session management • Implement data privacy
The system's functionality is dependent on the	• Wi-Fi dependency • Network attack • Denial-of-Service	• System is down and security is compromised	• Set-up other system connectivity e.g.,

Wi-Fi connection only,	(DoS) and Denial-of -Sleep (DoSL) attacks	once Wi-Fi connection is lost or weak • Insecure network • Unencrypted communication	Local Area Connection • Firewalls like Next-generation firewall • Limit device or network bandwidth • Backup connectivity options like 4G or 3G, to ensure that the system remains operational even if the Wi-Fi connection is lost. • Intrusion Detection and Prevention Systems • Implementation of secure socket layer (SSL) Certificates, • Data Encryption

			• Network segmentation
Lack of security tests that make room for the system's improvements	• More prone to breaches	• Lack of security tests and scanning	• Regular security and backup testing, and scanning for threats helps in reinforcing the system
Lack of data storage security	• Injection attacks • Tampering	• Unsecure data storage	• Secure databases • Antivirus • Data encryption
Lack of Security Updates	• More prone to breaches	• Lack of Security Updates and patches	• Regular and automatic System and hardware updates
Unsecured device management	• Unauthorised factory-resetting of devices • Installation of malicious software and updates	• Malicious software updates • Device breaches • Weak firmware or software, servers, backend	• Use of secure updating mechanisms like digital signatures • Practising secure Programming

	• Software and firmware risks and attacks	application	practices • System centralization • Implementing secure device management protocols • Limiting the number of device management access points • Ensure tamper-resistant hardware
Human Error	• Breaches • Social engineering	• Human errors	• Cybersecurity training on users

Table 1.4: SmartHome Vulnerabilities and Mitigations

In conclusion, a number of vulnerabilities have been identified on the Smart Home system from the design phase and illustrated in the ADTrees, further explorations of the vulnerabilities related to the system were also taken into consideration and also identified during testing of the system.

By utilising the scrum agile approach, this sprint will only prioritise in the mitigation of Authentication and Encryption Vulnerabilities as they are the basic pillars in securing the

system. Further security controls will be planned and incorporated in future development of upcoming sprints of the system.

---

2. HYPOTHESIS INVESTIGATION

2.1 Description

The team's hypothesis question focused on achieving the basic cybersecurity requirement of authentication and authorisation within a microservice. can authentication and authorisation be handled within the code or should it be carried out and achieved as a different microservice on a separate container? (de Almeida, M.G. and Canedo, E.D., 2022.) discussed heavily that microservice should focus on dividing the application into small services, each running separately and connected through an (API) application programming interface gateway only when required. our hypothesis attempted to confirm if the authentication and authorisation processes can be added into the code without impacting the efficiency of the code. as MQTT connects the publisher and subscribers via a trusted broker in practice. which is often the case since IoT device owners often also are provided control over the broker and still require to validate the ownership via a username and password in order to login to the platform or pbe rovided with different commands. the hypothesis question attempts to find a direct answer to service discovery limitations as microservice architecture make service

discovery hard especially when inter-service communication is required (Yarygina, T. and Bagge, A.H., 2018).

2.2 The Model

The current model proposed attempt to simplify the microservices function and code management, by integrating the authentication and authorisation function within the MQTT application running within docker as a container. This would remove the requirement of having a complex infrastructure such as integrating the docker simulation code with a (RADIUS) remote authentication dial in server or an Accesses control list. This model is meant to overcome the complexity of adding different containers as computing resources tend to be limited and due to several hardware constraints the team wanted to simplify the approach and utilise of containers with a secure code depyloment and simplified access through the use of a basic username and password that is secure, functional and practical.

2.3 Experiments and Analysis

Manually testing the application

we attempted to login to the application without going into the signup process and it failed which is the desired result. this proved that the code was an efficient approach to achieve the result of authentication and authorisation processes. (Al-Naji, F.H. and Zagrouba, R., 2020) focused on the importance of authentication and authorisation within IoT devices as a continuous process.

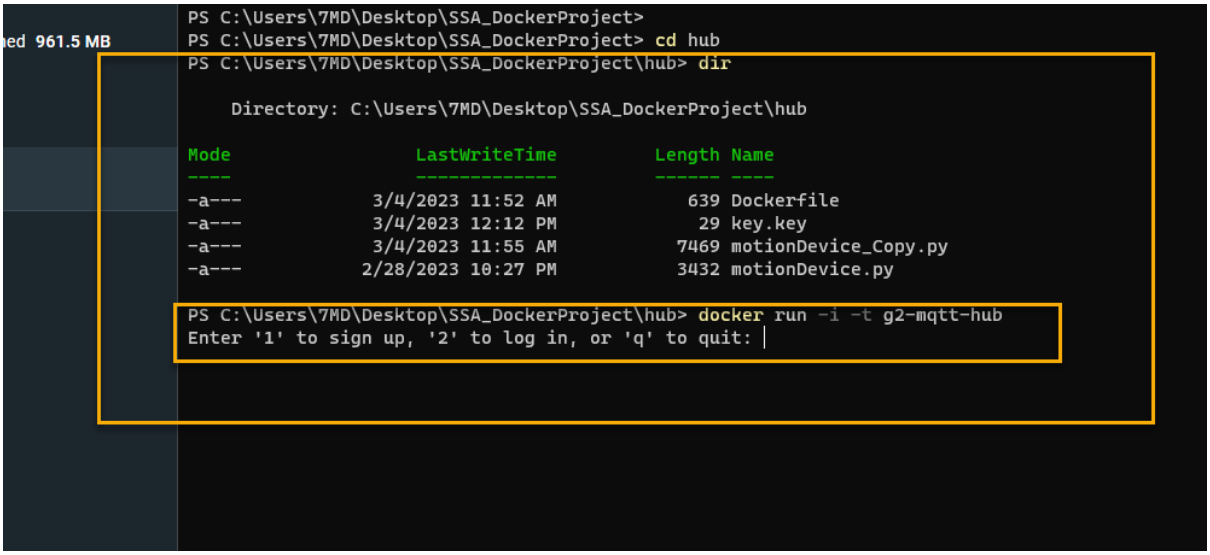

Figure 2.1: Example of running the docker container with the MQTT code

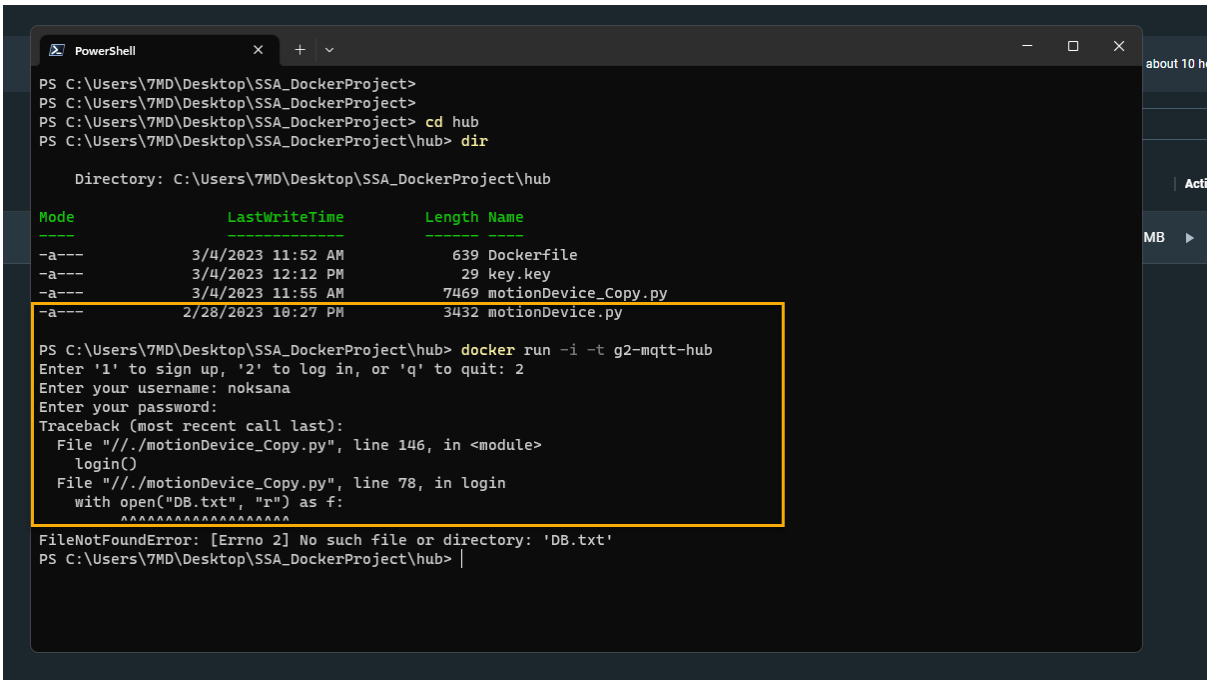

Figure 2.2: Attempting to bypass the built-in authentication, which resulted in failure

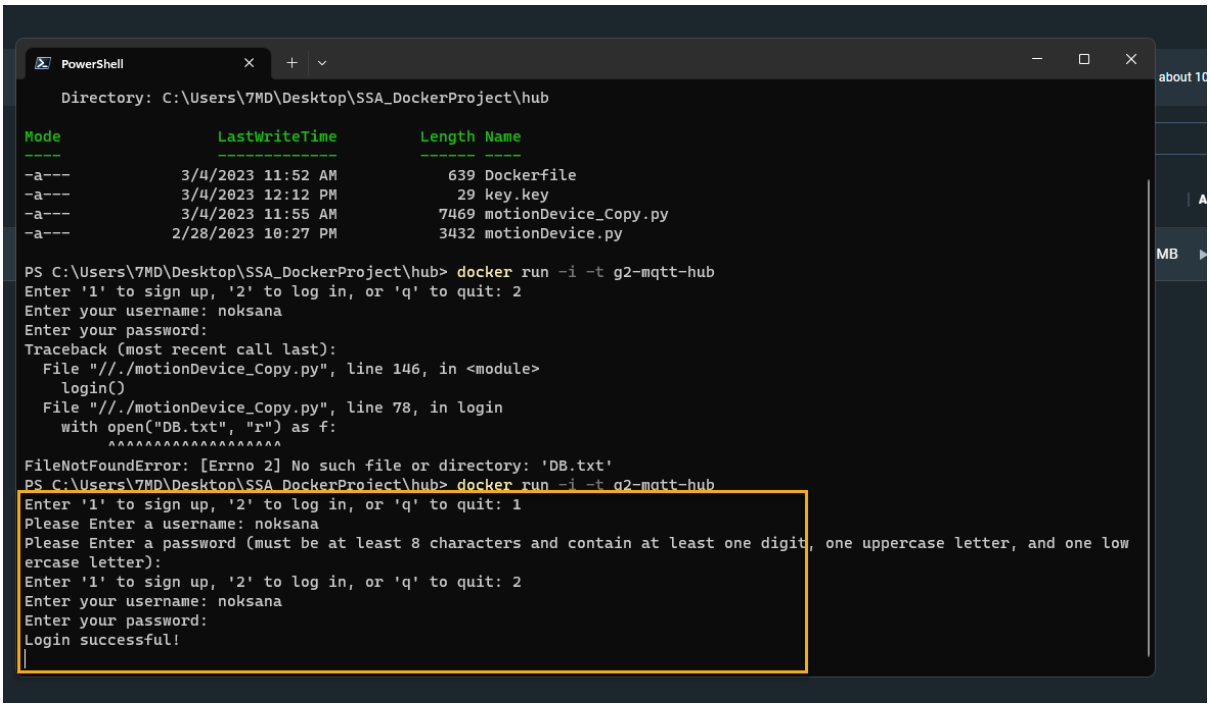

Figure 2.3: successfully loggin into the MQTT code after signing up to the application

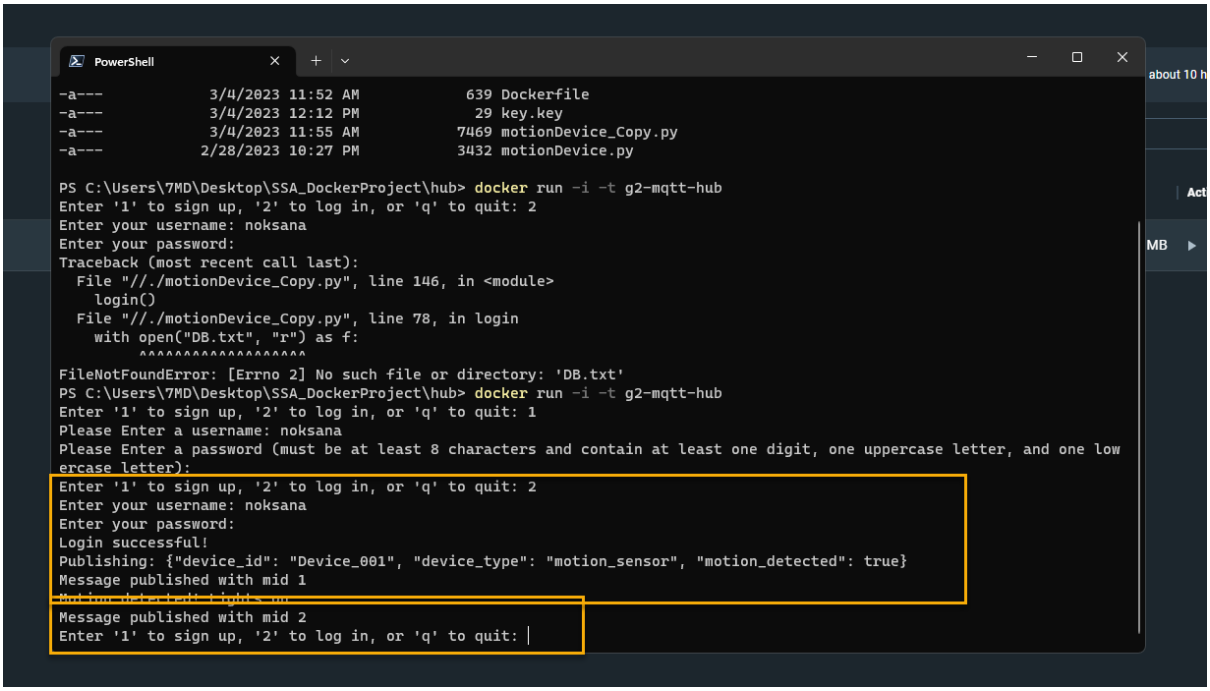

Figure 2.4: The code successfully running and simulating the motion sensor IoT

2.4 Python Script Testing with pylint

Pylint is an analysis tool that searches for errors, bugs, and other issues within Python code (*Dasgupta & Hooshangi, 2017).* To ensure our team's Python code was secure, Pylint was used to test the code and improve each script. The initial code tests started with a score of 2.24 and was improved to reach the maximum score of 9.6

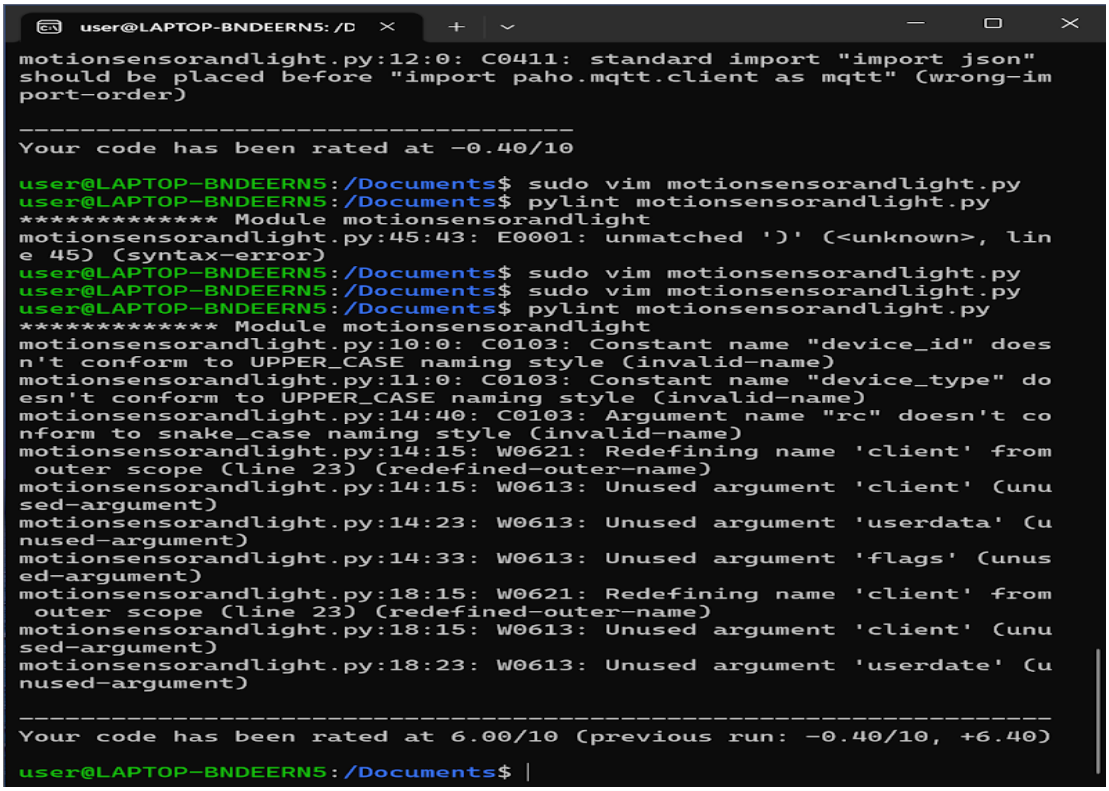

Figure 2.5: Pylint score for motion sensor and light clients before improvements.

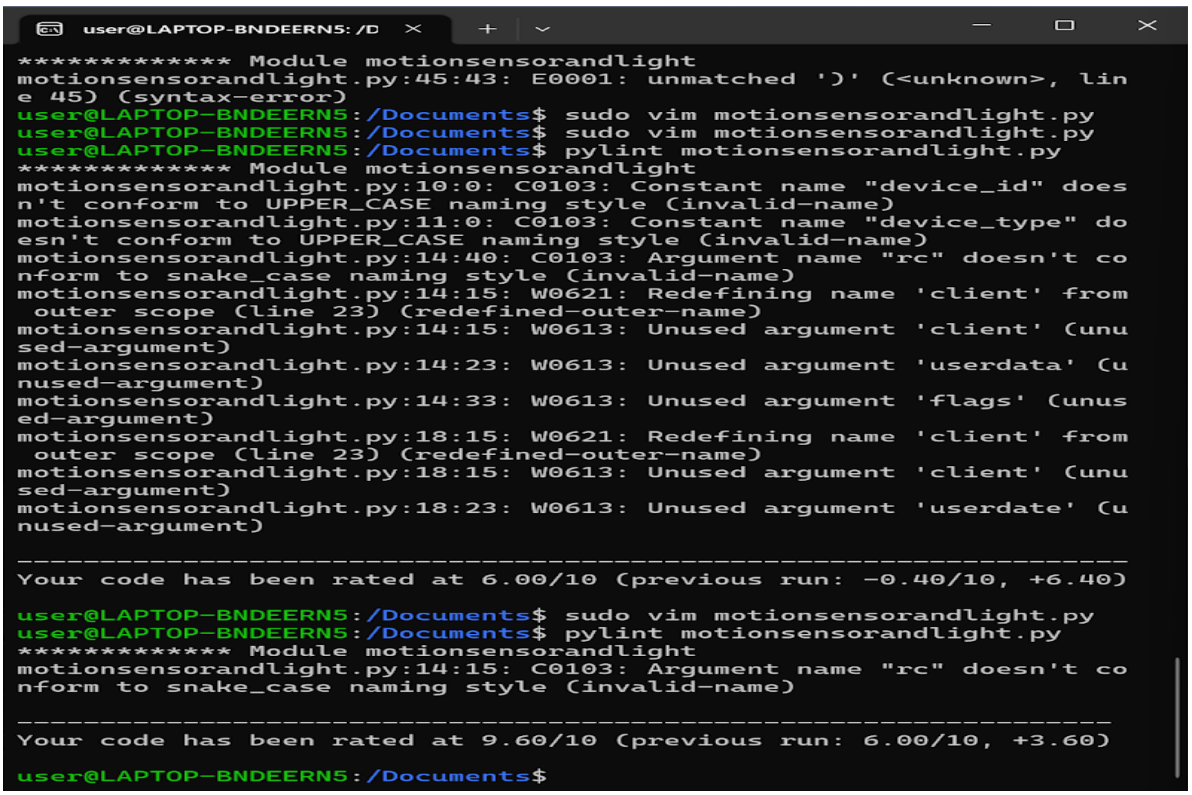

Figure 2.6: Pylint score for motion sensor and light clients after improvements.

The pylint score increased from 6.00 to 9.60. The variable “rc” was left as it was as this is commonly used in other scripts.

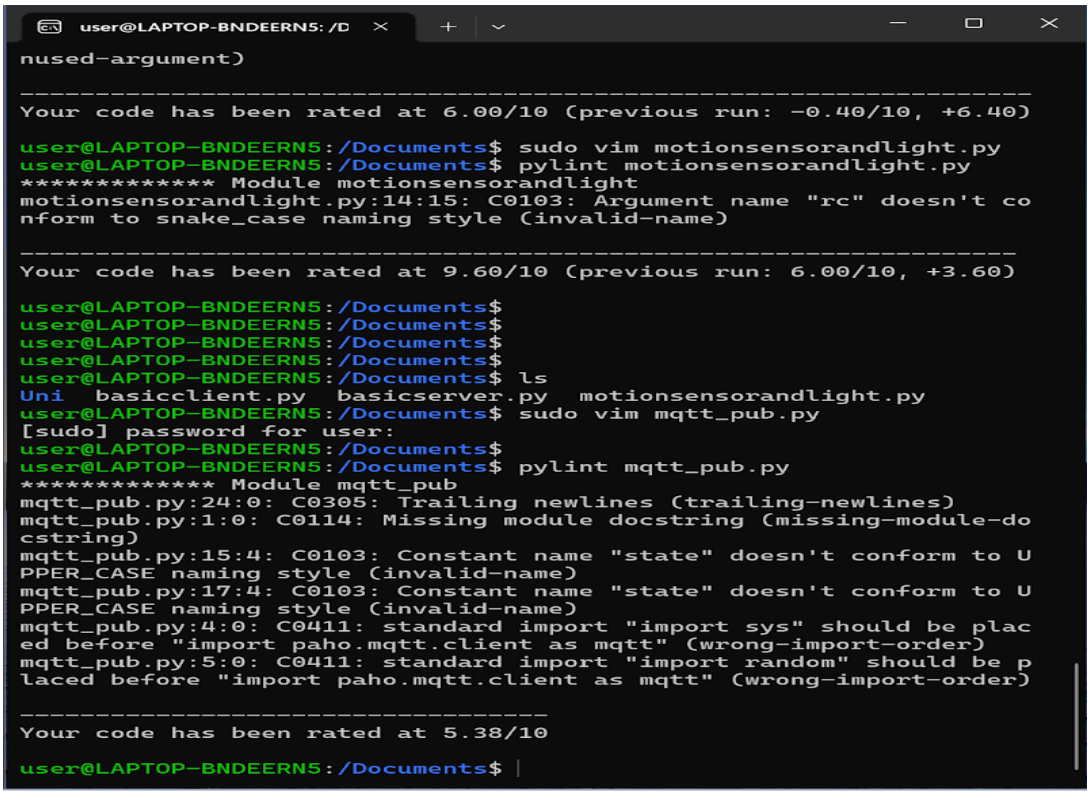

Figure 2.7: Light client simulator pylint score before improvements.

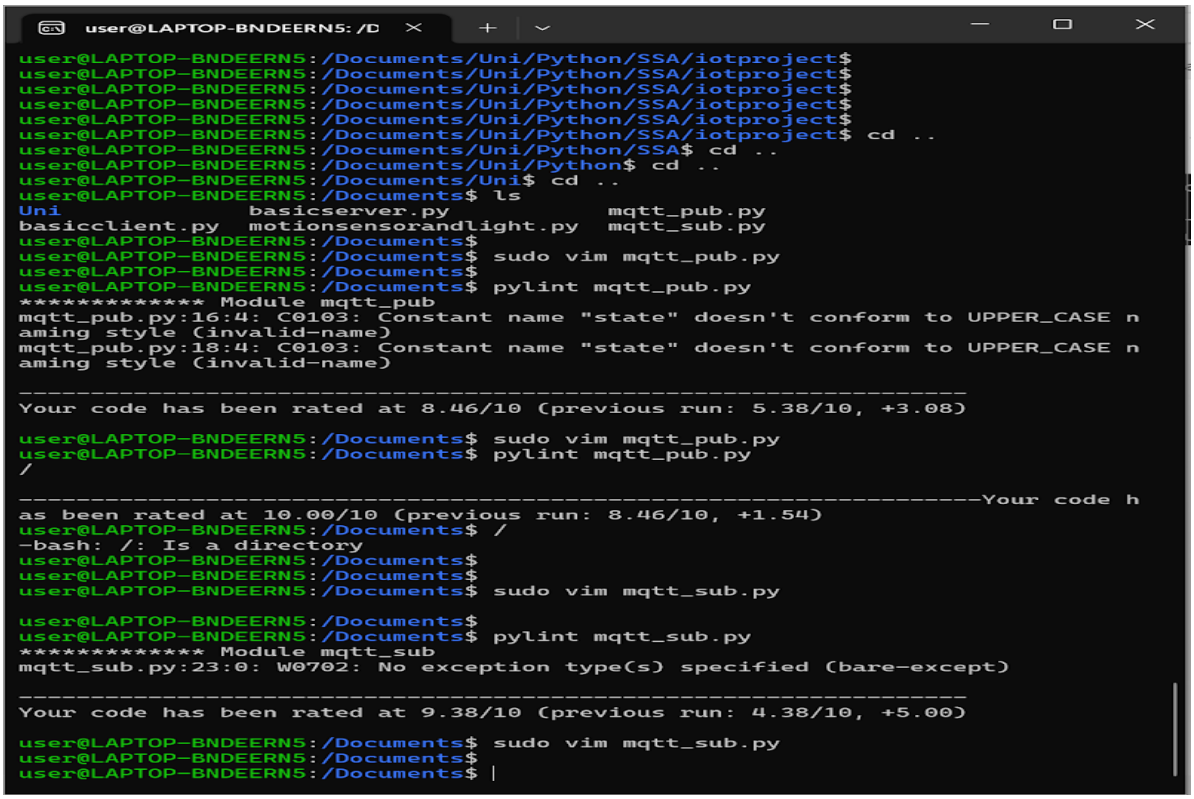

Figure 2.8: Light client simulator pylint after improvements.

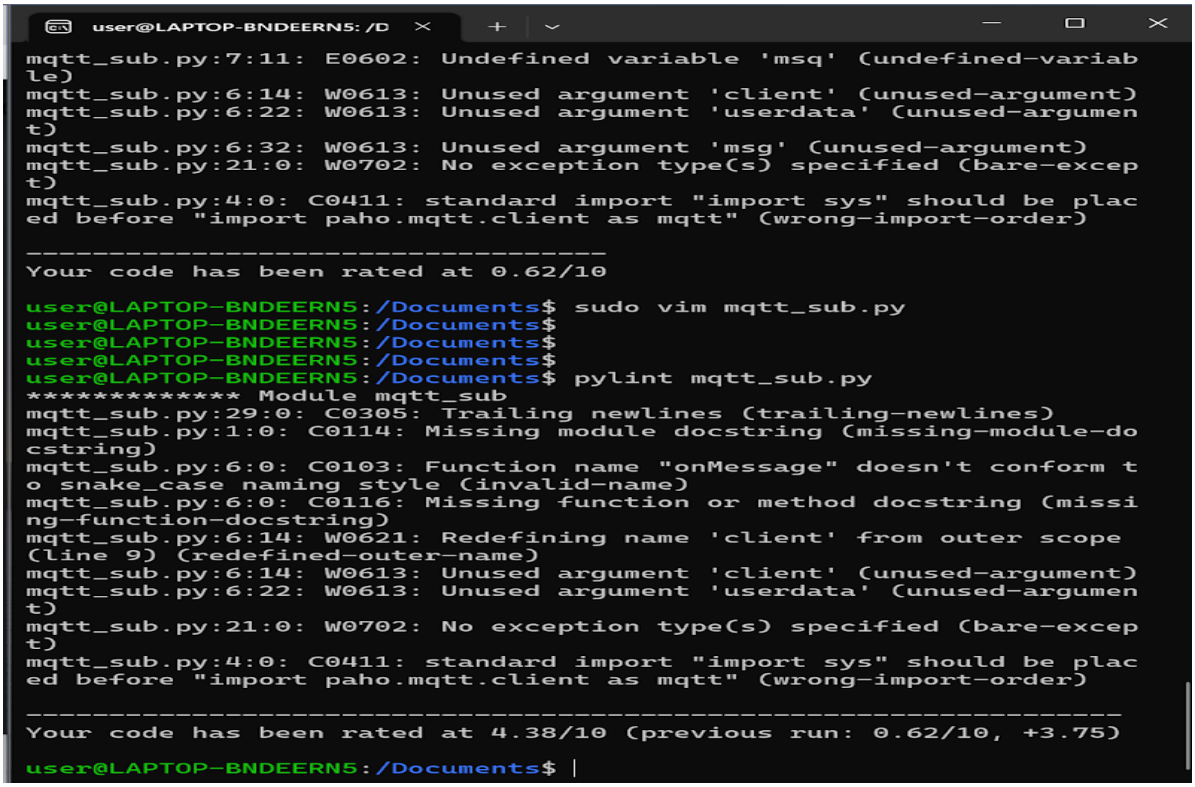

Figure 2.9: Hub simulator pylint score before improvements.

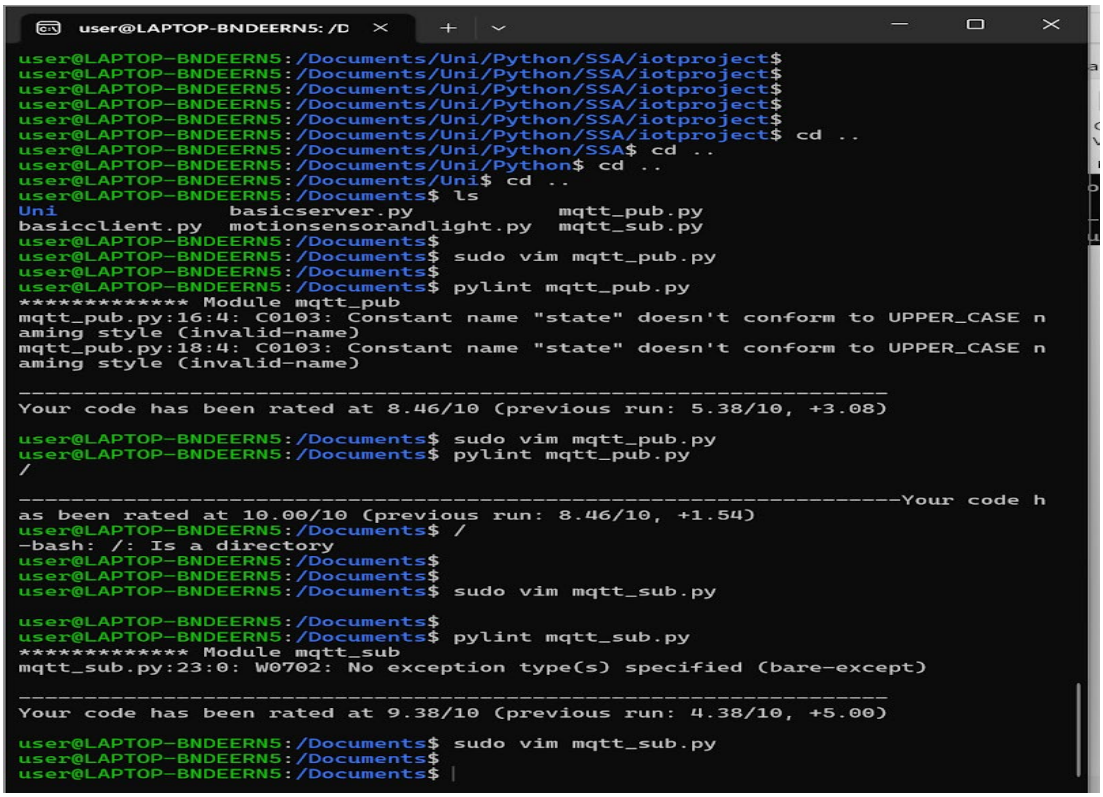

Figure 2.10: Hub simulator pylint after before improvements.

These scores show a large improvement in each of the scripts, allowing for more efficient code using best practices.

2.5 Docker Vulnerability test

The test can only be carried out after building the image, then go to the container tab and selecting view image, several vulnerabilities were identified by Docker as it built the images with the most stable release of the Debian OS.

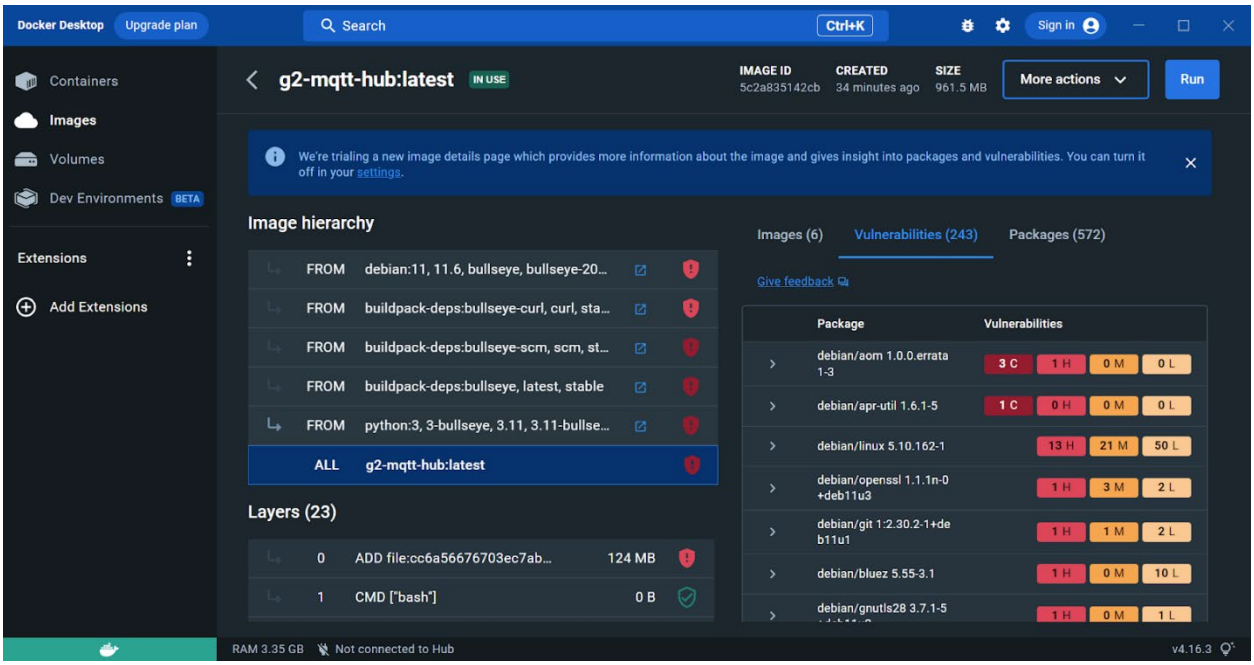

Figure 2.11: List of 243 identified vulnerabilities within the Docker Image

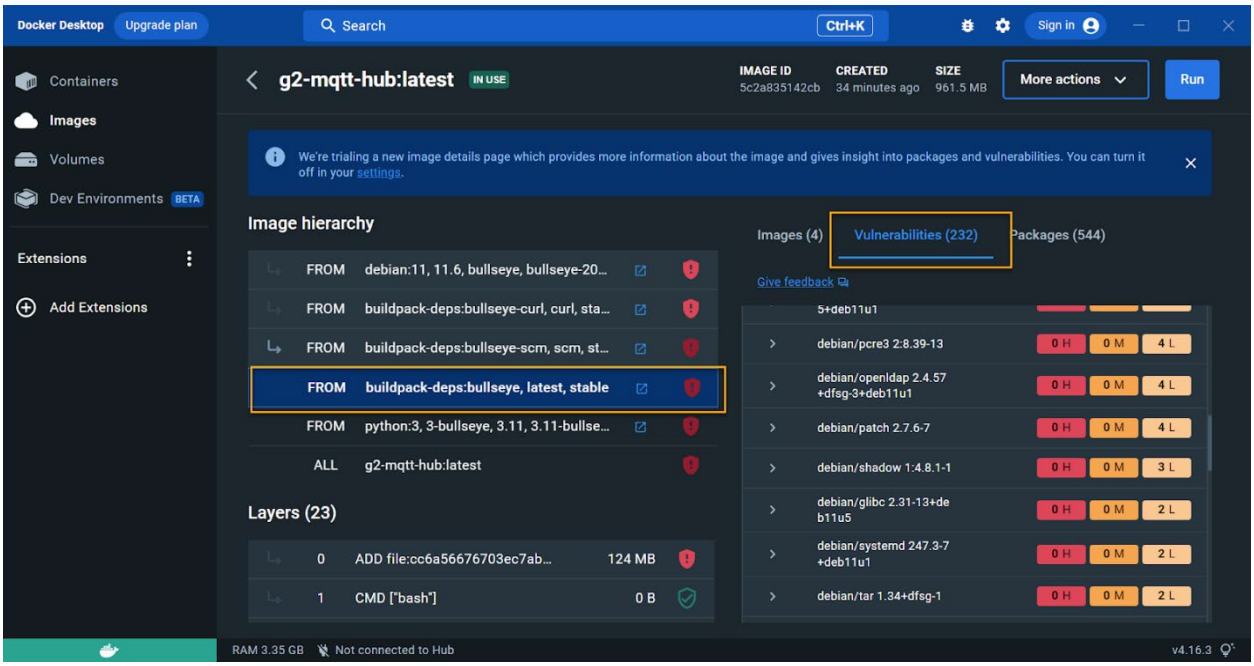

Figure 2.12: The above image is for the Debian destroy utilised within docker

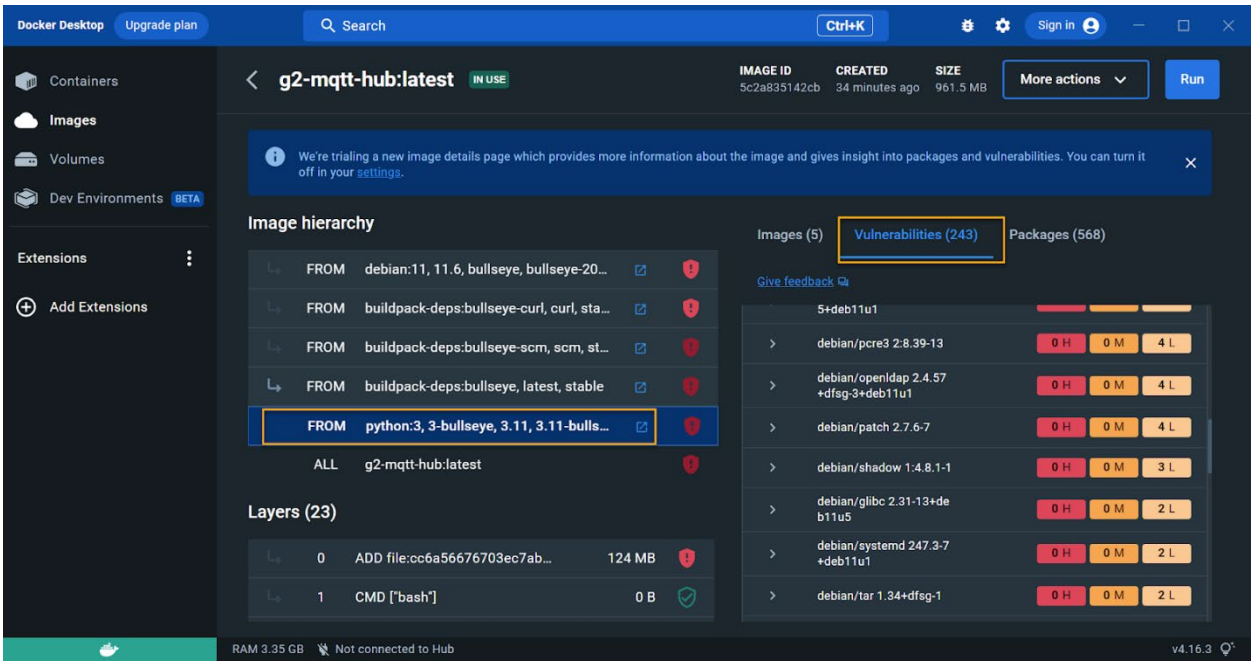

Figure 2.13: The above image is for vulnerabilities with python 3.11 and used libraries.

2.6 Bandit Testing

Step 1: install bandit with the following command:

Pip install bandit

Step 2: run the following command to test the python code

Bandit -r motionDevice_Copy.py

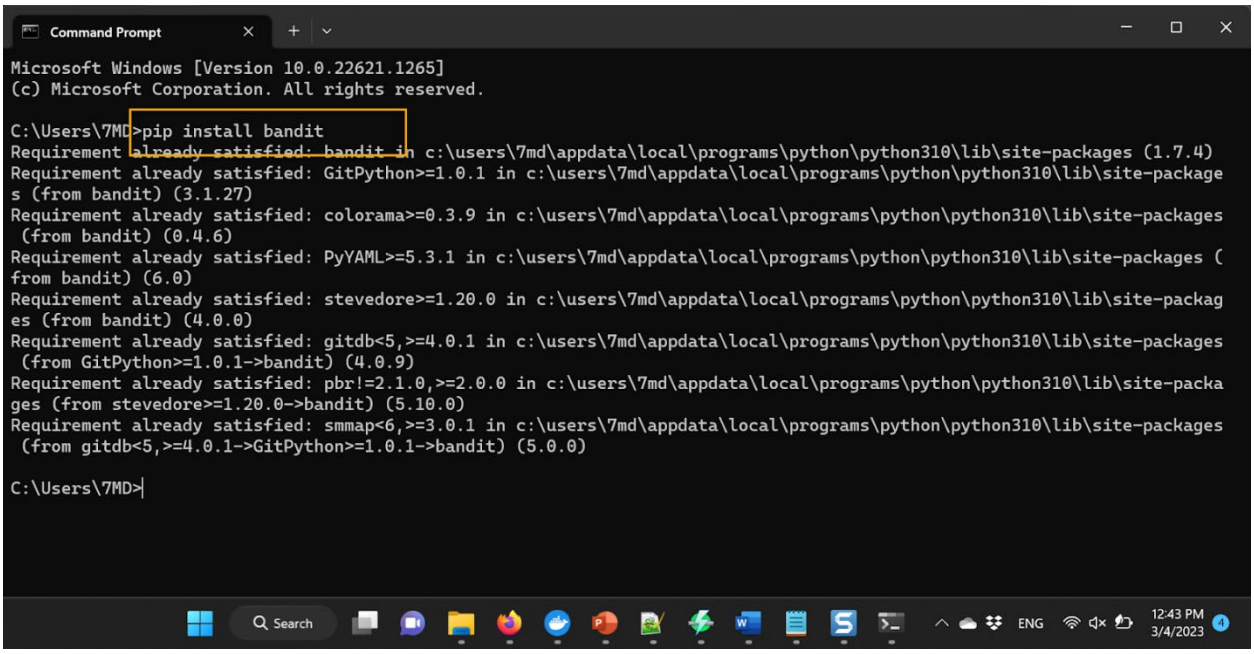

Figure 2.14: installing bandit

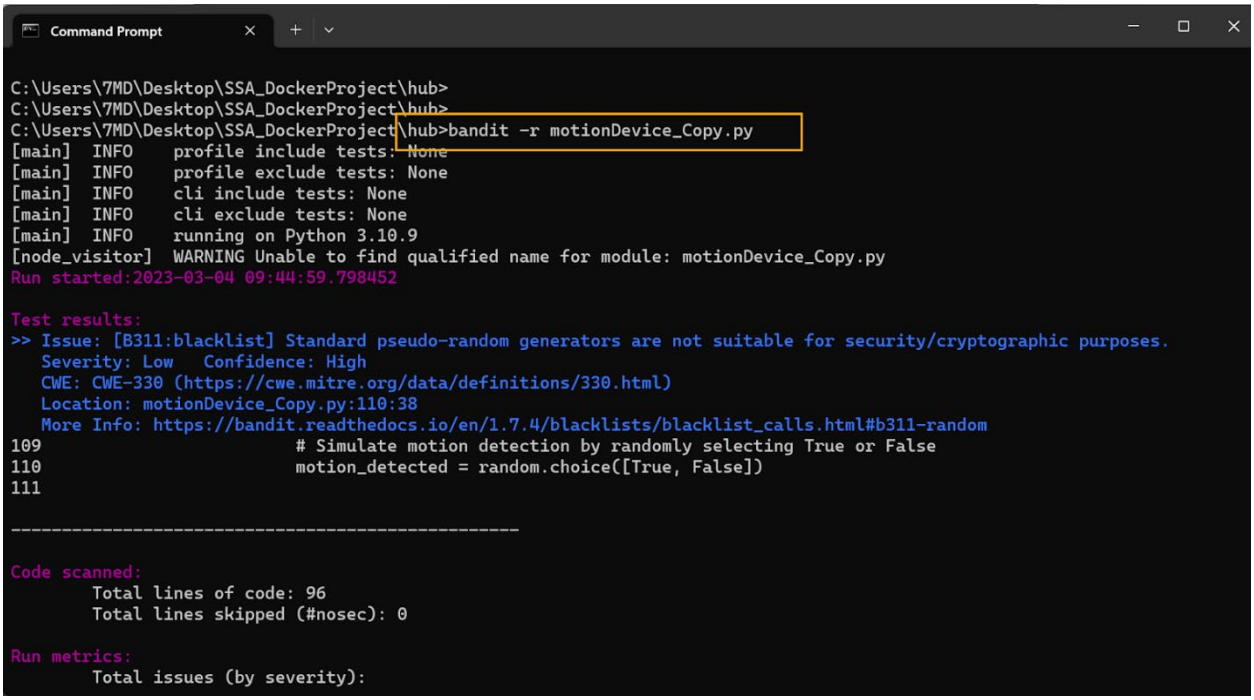

Figure 2.15: Running Bandit test on our python code

Final Bandit output:

```
Test results:
>> Issue: [B311:blacklist] Standard pseudo-random generators are not suitable for
security/cryptographic purposes.
   Severity: Low   Confidence: High
   CWE: CWE-330 (https://cwe.mitre.org/data/definitions/330.html)
   Location: motionDevice_Copy.py:110:38
```

2.7 Conclusion

We have concluded that the hypothesis requires more time to test and confirm if authentication and authorization can be carried out within python efficiently without the assistance of additional microservices adding further complexity to the simulation. although the desired result was achieved, further evidence such as log's and appropriate checks on the security of the authentication and authorisation, was not achieved as it broke the MQTT code and stopped the main objective was to simulate the IoT environment required for a smart home light system.

Recommendations and improvements:

- Physical network/VPN for ultimate security; to connect the IoT devices with a master control hub in a secure manner.
- (TLS) Transport layer security with certificate credentials from CA for all connections;

- All inbound ports should be disabled at MQTT edge clients; and additionally restricted with a firewall access list implementation.
- Only two TCP/IP ports (8883 and 443) should be open at the MQTT server;
- Use MQTT client username/password at MQTT servers; and ensure audit trails are stored separately for forensic use if required.
- ACLs should be used to limit MQTT client access to the topic levels they can either publish or subscribe to.
- Enable the use of biometric authentication prior to access of the MQTT platform / Smarthome system to ease the login process of the user.

---

3. HOW TO / CODE INSTRUCTIONS/SOURCE CODE

Requirements and setup/ running the application guideline

Install docker desktop on windows available from:

https://www.docker.com/products/docker-desktop/

Install python for windows, available from: https://www.python.org/downloads/ current version in use is 3.11

Steps to run the microservices applications:

1. Navigate to the folder SSA_DockerProject with CMD
    a. One terminal window should be dedicated to running the Hub Docker file
    b. One terminal window should be dedicated to running the Light Docker file

2. Within the first command line window run the command to build the docker image
    1. docker build -t g2-mqtt-hub . # command is used to build the image
    2. docker run -i -t g2-mqtt-hub # command is used to run the container

When deleting the images on docker first start with deleting the container, then delete the image when it's no longer required in that order.

download paho.mqtt by running

```
pip install paho-mqtt
```

to run a pyscript use cmd terminal and run the name of the script eg. lightControl.py then run the python file

```
#MQTT Subscriber

import paho.mqtt.client as mqtt
import sys

def onMessage(client, userdata, msg):
    print( msg.payload.decode())
```

```
client =mqtt.Client()
client.on_message = onMessage

if client.connect("localhost", 1883, 60) != 0:
    print("Could not connect to MQTT Broker!")
    sys.exit(-1)

client.subscribe("lightstate")

try:
    print("Press CTRL+C to exit....")
    client.loop_forever()
except:
    print("Disconnecting from broker")

client.disconnect()

# This Python script uses the MQTT (Message Queuing Telemetry Transport) protocol
to communicate between a motion sensor device and a light control system.
# The script consists of two parts: one for the motion sensor device (motionDevice.py)
and one for the light control system (lightControl.py).
```

```
# This is the first part of the script (motionDevice.py), which runs on the motion sensor device.

#Import of the libraries
import paho.mqtt.client as mqtt #for MQTT protocol
import time #to simulate IoT delays
import random #to create random id
import json #to convert the python dictionary into a JSON string that can be written into a file

# Set up the device's ID and type
device_id = "Device_001"
device_type = "motion_sensor"

# Define the on_connect and on_publish functions for the MQTT client
def on_connect(client, userdata, flags, rc): #rc (return code) is used for checking that the connection was established.
    print("Connected to broker with result code "+str(rc))

def on_publish(client, userdata, mid): #mid value is an integer that corresponds to the published message number as assigned by the client.
    print("Message published with mid "+str(mid))
```

```
# Create MQTT client instance
client = mqtt.Client()

#Assign the on_connect and on_publish functions to it
client.on_connect = on_connect
client.on_publish = on_publish

# Connect to the MQTT broker
client.connect("mqtt.eclipseprojects.io") # a public test MQTT broker address/service
"https://mqtt.eclipseprojects.io/ "

# Loop indefinitely
while True:
    # Simulate motion detection by randomly selecting True or False
    motion_detected = random.choice([True, False])

    # Create a dictionary containing the device's ID, type, and whether motion was
detected
    data = {
        "device_id": device_id,
        "device_type": device_type,
        "motion_detected": motion_detected
    }
```

```
    # Convert the dictionary to a JSON string and publish it to the "motion_sensor" topic
    payload = json.dumps(data)
    print("Publishing: " + payload)
    client.publish("motion_sensor", payload)

    # If motion was detected, also publish a message to turn on the light
    if motion_detected:
        print("Motion detected! Lights on")
        client.publish("light_control", "on")
    else:
        print("No motion detected.")

    # Wait for 5 seconds before repeating the loop
    time.sleep(5)
```

#CODE SUMMARY:

It starts by importing the required libraries then it defines the MQTT broker parameters/connection and sets up the device's ID and type.
It also defines the on_connect and on_publish functions for the MQTT client, which are called when the client connects to the broker and publishes a message, respectively.

The script creates an MQTT client instance, assigns the on_connect and on_publish functions to it, and connects to the MQTT broker.
The script then enters an infinite loop where it simulates motion detection by randomly selecting True or False,
creates a dictionary containing the device's ID, type, and whether motion was detected,
converts the dictionary to a JSON string, and publishes it to the "motion_sensor" topic.

If motion was detected, it also publishes a message to turn on the light by publishing an "on" message to the "light_control" topic.
The loop then waits for 5 seconds before repeating.

This Python script uses the MQTT (Message Queuing Telemetry Transport) protocol to communicate between a motion sensor device and a light control system.
The script consists of two parts: one for the motion sensor device (motionDevice.py) and one for the light control system (lightControl.py).

#This is the second part of the script (lightControl.py), which runs on the light control system.

#Import libraries
import paho.mqtt.client as mqtt #for MQTT protocol

```
import json #to convert the python dictionary into a JSON string that can be written into a file

# Set up the initial state of the light
light_state = "off"

# Define the on_connect and on_message functions for the MQTT client
def on_connect(client, userdata, flags, rc):
    print("Connected to broker with result code "+str(rc))

    # Subscribe to the "motion_sensor" and "light_control" topics
    client.subscribe("motion_sensor")
    client.subscribe("light_control")

def on_message(client, userdata, message):
    # Extract the topic and payload from the received message
    global light_state
    topic = message.topic
    payload = message.payload.decode()
    print("Received message: "+payload+" on topic: "+topic)

    # If the message is from the motion sensor, check if motion was detected and turn on the light if it's currently off
```

```
    if topic == "motion_sensor":
        data = json.loads(payload)
        motion_detected = data["motion_detected"]
        if motion_detected:
            print("Motion detected!")
            if light_state == "off":
                client.publish("light_control", "on")
                print("Turning on the light.")
                light_state = "on"
        else:
            print("No motion detected.")

    # If the message is a light control message, update the state of the light accordingly
    elif topic == "light_control":
        if payload == "on":
            print("Light turned on.")
            light_state = "on"
        elif payload == "off":
            print("Light turned off.")
            light_state = "off"

# Create MQTT client instance
client = mqtt.Client()
```

```
# Assign the on_connect and on_message functions to MQTT client instance
client.on_connect = on_connect
client.on_message = on_message # Set up callback function for message received event

# Connect to the MQTT broker
client.connect("mqtt.eclipseprojects.io") # a public test MQTT broker address/service "https://mqtt.eclipseprojects.io/ "

# Subscribe to the motion detection topic and define callback
client.subscribe("home/light")

# Start the MQTT loop indefinitely to listen and handle messages
client.loop_forever()


# CODE SUMMARY:

# It starts by importing the required libraries, including the Paho MQTT client library and JSON and defines the MQTT broker parameters,
# sets up the initial state of the light, and defines the on_connect and on_message functions for the MQTT client.
```

The on_connect function is called when the client connects to the broker and subscribes to the "motion_sensor" and "light_control" topics.

The on_message function is called when the client receives a message, extracts the topic and payload from the received message,

and checks if the message is from the motion sensor or the light control system.

If the message is from the motion sensor, it checks if motion was detected and turns on the light if it's currently off.

If the message is a light control message, it updates the state of the light accordingly.

The script creates an MQTT client instance, assigns the on_connect and on_message functions to it, and connects to the MQTT broker.

The script then starts the MQTT loop indefinitely to listen and handle messages.

---

4. REFERENCES

Abdullah, T., Ali, W., Malebary, S. & Ahmed, A. A. (2019) A Review of Cyber Security Challenges, Attacks and Solutions for Internet of Things Based Smart Home. *International Journal of Computer Science and Network Security (IJCSNS),* 19(9), pp. 139-146.

Abomhara, M. and Køien, G.M. (2015) Cyber security and the internet of things: vulnerabilities, threats, intruders and attacks. *Journal of Cyber Security and Mobility*, pp.65-88.

Ahamed, J. and Rajan, A.V. (2016) Internet of Things (IoT): Application systems and security vulnerabilities. In *2016 5th International conference on electronic devices, systems and applications (ICEDSA)* (pp. 1-5). IEEE.

Ahmed J. Hintaw, Selvakumar Manickam, Mohammed Faiz Aboalmaaly & Shankar Karuppayah (2021) MQTT Vulnerabilities, Attack Vectors and Solutions in the Internet of Things (IoT), IETE Journal of Research

Anand, P. et al. (2020) IoT Vulnerability Assessment for Sustainable Computing: Threats, Current Solutions, and Open Challenges. *IEEE Access,* Volume 8, pp. 168825-168853.

Apriorit. (2022) *Internet of Things (IoT) Security: Challenges and Best Practices.* [Online] Available at: https://www.apriorit.com/white-papers/513-iot-security [Accessed 02 February 2023].

Al-Naji, F.H. and Zagrouba, R., 2020. A survey on continuous authentication methods in Internet of Things environment. Computer Communications, 163, pp.109-133.

Arthi, R. and Krishnaveni, S. (2021) Design and Development of IOT Testbed with DDoS Attack for Cyber Security Research. In *2021 3rd International Conference on Signal Processing and Communication (ICPSC)* (pp. 586-590). IEEE.

Asim , Z. (2022) Applications of MQTT protocol with its benefits and comparison [Online] https://highvoltages.co/iot-internet-of-things/mqtt/applications-benefits-and-comparison-of-mqtt-protocol/

Bhattacharjee, S., Salimitari, M., Chatterjee, M., Kwiat, K. and Kamhoua, C.(2017) Preserving data integrity in IoT networks under opportunistic data manipulation. *IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)* (pp. 446-453). IEEE.

Borgini, J. (2021) *Tackle IoT application security threats and vulnerabilities.* [Online] Available at: https://www.techtarget.com/iotagenda/tip/Tackle-IoT-application-security-threats-and-vulnerabilities [Accessed 2 February 2023].

Cam-Winget, N., Sadeghi, A.R. and Jin, Y. (2016) Can IoT be secured: Emerging challenges in connecting the unconnected. In *Proceedings of the 53rd Annual Design Automation Conference* (pp. 1-6).

Cooper, J. and James, A. (2009) Challenges for database management in the internet of things. *IETE Technical Review*, *26*(5), pp.320-329.

Cruz-Piris, L., Rivera, D., Marsa-Maestre, I., De La Hoz, E. and Velasco, J.R., (2018) Access control mechanism for IoT environments based on modelling communication procedures as resources. *Sensors*, *18*(3), p.917.

Dasgupta, S. and Hooshangi, S., 2017. 'Code quality: Examining the efficacy of automated tools.', *Emergent Research Forum Paper.*

de Almeida, M.G. and Canedo, E.D., 2022. Authentication and authorization in microservices architecture: A systematic literature review. *Applied Sciences*, *12*(6), p.3023.

Ghasemi, M., Saadaat, M. and Ghollasi, O. (2019) Threats of social engineering attacks against security of Internet of Things (IoT). In *Fundamental Research in Electrical Engineering: The Selected Papers of The First International Conference on Fundamental Research in Electrical Engineering* (pp. 957-968). Springer Singapore.

Gupta, B.B., Chaudhary, P., Chang, X. and Nedjah, N. (2022) Smart defense against distributed Denial of service attack in IoT networks using supervised learning classifiers. *Computers & Electrical Engineering*, *98*, p.107726.

Hübschmann, I. (2021) The Pros and Cons of Using MQTT Protocol in IoT [Online] https://www.nabto.com/mqtt-protocol-iot/

Janes, B., Crawford, H. and OConnor, T.J. (2020) Never ending story: authentication and access control design flaws in shared IoT devices. In *2020 IEEE Security and Privacy Workshops (SPW)* (pp. 104-109). IEEE.

Jiang, Y., Xie, W. and Tang, Y. (2018) November. Detecting authentication-bypass flaws in a large scale of IoT embedded web servers. In *Proceedings of the 8th International Conference on Communication and Network Security* (pp. 56-63).

Matthews, K. (2020) MQTT: A Conceptual Deep-Dive [online] https://ably.com/topic/mqtt

Meador,D. (2020) Distributed Systems. [online] https://www.tutorialspoint.com/Distributed-Systems

Mlambo, N., Farnan, A., Ahmad, H. & Mutegi, B., 2023. Development Team Project_Final. London: Unpublished.

Rizvi, S., Pipetti, R., McIntyre, N., Todd, J. and Williams, I. (2020) Threat model for securing internet of things (IoT) network at device-level. *Internet of Things*, *11*, p.100240.

Samarth. (2021) What is MQTT for IoT Devices? [Online] https://psiborg.in/advantages-of-using-mqtt-for-iot-devices/

Shin, S. and Seto, Y. (2020) Development of IoT security exercise contents for cyber security exercise system. In *2020 13th International Conference on Human System Interaction (HSI)* (pp. 1-6). IEEE.

Touqeer, H. et al. (2021) Smart home security: challenges, issues and solutions at different IoT layers. *The Journal of Supercomputing,* Volume 77, pp. 14053-14089.

Tweneboah-Koduah, S., Skouby, K.E. and Tadayoni, R. (2017) Cyber security threats to IoT applications and service domains. *Wireless Personal Communications*, *95*, pp.169-185.

Vaccari, I., Narteni, S., Aiello, M., Mongelli, M. and Cambiaso, E. (2021) Exploiting Internet of Things protocols for malicious data exfiltration activities. *IEEE Access*, *9*, pp.104261-104280.

Yarygina, T. and Bagge, A.H., 2018, March. Overcoming security challenges in microservice architectures. In *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)* (pp. 11-20). IEEE.

Yudidharma, A. (2022) A systematic literature review: Messaging protocols and electronic platforms used in the internet of things for the purpose of building smart homes. 7th International Conference on Computer Science and Computational Intelligence 2022

5.0 CODE REFERENCES

Anon, 2021. Developing inside a Docker Container in Visual Studio Code. [Online] Available at: https://francescopochetti.com/developing-inside-a-docker-container-in-visual-studio-code/ [Accessed 8 February 2023].

Bender, M., et al, 2021,. 'Open-source mqtt evaluation.', *18th Annual Consumer Communications & Networking Conference,* pp. 1-4. IEEE.

Cope, S., N.D.. How to Use The Paho MQTT Python Client for Beginners. [Online] Available at: http://www.steves-internet-guide.com/into-mqtt-python-client/ [Accessed 11 February 2023].

How to hash, salt and verify passwords in NodeJs, python Golan and Java. (supertokens team, 2022) available at: https://supertokens.com/blog/password-hashing-salting [Accessed on 27 February, 2023]

Joy, A., N.D.. A Deep Dive Into Fernet Module in Python. [Online] Available at: https://pythonistaplanet.com/fernet/ [Accessed 20 February 2023].

PyPi, N.D.. paho-mqtt 1.6.1. [Online] Available at: https://pypi.org/project/paho-mqtt/#:~:text=This%20code%20provides%20a%20client,.9%2B%20or%203.6%2B. [Accessed 26 February 2023].

Python (2023). 'sys – System-specific parameters and functions', *Python*. [Online] Available at: https://docs.python.org/3/library/sys.html [Accessed: 28/02/2023]

Python login and signup, Available at: https://github.com/br34th3r/PythonLoginAndRegister [Accessed on 26 February 2023]

Python program to check the validity of a password (2023), available at: https://www.geeksforgeeks.org/python-program-check-validity-password/ [Accessed on 26 Febraury 2023]

TutorialsTeacher (2023). 'Python – Random Module', *TutorialsTeacher*. [Online] Available at: https://www.tutorialsteacher.com/python/random-module#:~:text=The%20random%20module%20is%20a,%2C%20shuffle%20elements%20randomly%2C%20etc. [Accessed: 28/02/2023]