Development Team Project

Readme for Secure Systems Architecture (SSA)

MSc Cyber Security

Unit 6 Submission

Words 659

Solution Description

The solution comprises an MQTT (Message Queuing Telemetry Transport) broker connected to a controller, providing two node options; a smart bulb and a thermostat comprising both a sensor and actuator. This has been set up to run in Docker and is scalable, currently capable of having up to 9999 containers of each node type. See Appendix A for implementation instructions.

SysML is used to formally model the Smart Home topology. Below is the requirements diagram with the further diagrams in Appendix C.

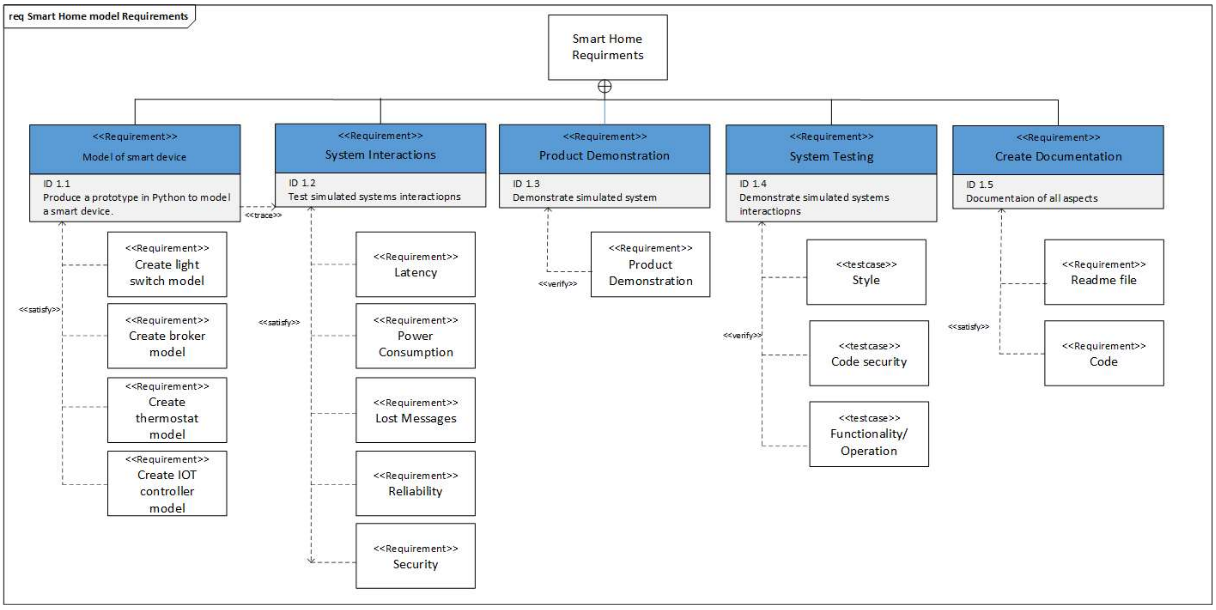


Figure 1: SysML Requirements diagram

Security Benefits

The solution has the following security benefits in Phase 1 (Table 3):

- Authentication (TLS), node authentication and authorisation have been prioritised within the development timeframe.
- Third party components for strong encryption and authentication used but checked for legitimacy prior to implementation.
- User has been specified in the Dockerfiles to avoid privilege escalation to the host OS (The Digital Life, 2021).
- Disabled internet connection and set up on an isolated network.
- The maximum temperature of the thermostat has been set to 40C, to limit the impact of integrity attacks.
- Secure coding practices have been used, including but not limited to input validation.

When identifying security controls, the OWASP Top 10 for Internet of Things (IoT) was considered (OWASP, 2018). TLS in particular mitigates most of the vulnerabilities identified in the original Attack-Defence Tree. The severe vulnerabilities were targeted first given the timeframe.

Security Challenges

The security challenges faced throughout development included:

- Security impacting intended functionality of the Smart Home system as found in final testing. Usability and security should be balanced appropriately to bring the most benefit to the end user.
- ID and password not encrypted for MQTT (Sardeshmukh & Ambawade, 2017).
- With no widely accepted IoT security standards or regulations to follow, ensuring full coverage of security and testing requirements is challenging.

Distributed System Challenges

Distributed systems can present challenges, including latency and traffic loss. IoT devices also have limitations, including limited bandwidth and processing power (Gerber & Romeo, 2017). Table 1 demonstrates mitigations for these challenges.

Challenge	**Solution(s)**
Latency	MQTT implemented (Sherozhenko. M, 2017).
Power Consumption	MQTT clients are small and require minimal resources (MQTT, 2021). Whilst node battery life with WiFi would be shorter as standard (Mocnej et al., 2019), the use of MQTT reduces overhead and makes WiFi a viable option.
Lost Messages	The MQTT Quality of Service (QoS) levels implemented. Only QoS 0 and 1 have been used, QoS 2 has not been implemented due to cost of transportation.
High Transport Cost	
Reliability	MQTT session persistence ensures an efficient and reliable connection to the broker (MQTT, 2021). TCP was the chosen method of transport as this is compatible with MQTT and ensures reliable communication.
Limited Bandwidth	A lightweight publish-subscribe middleware approach using MQTT was chosen (MQTT, 2021).
Fault tolerance	Container architecture means that node failure will not impact the correct functioning of other microservices.

Table 1

Areas for Security Improvement

The following improvements are suggested for future development in Phase 2 (Table 3):

- Strong ciphers are enforced with use of TLS v1.2 and above only. Using TLS v1.3 may provide additional efficiencies (Lundqvist et al., 2019).
- Ensure client individual certificates are used.
- Rejection of connections where the broker name is different to the broker certificate.
- Third layer of authentication and encryption in the application layer.
- Two Factor Authentication (2FA) on the controller GUI.

- Use namespaces and control groups in Docker for management and security improvements (Kozhirbayev & Sinnott, 2017).
- Recommend WPA3 to prevent DEAUTH attacks.
- Intrusion detection automation based on the broker logs.
- Vulnerabilities raised by Bandit and Snyk (Appendix B) reviewed and managed.

Conclusion

When developing this solution, security challenges and requirements for implementation were considered, some of which were found from testing (Appendix B). Although unable to complete all mitigations, the design was implemented with identified vulnerabilities in mind (Appendix C). The solution provided shows the foundational requirements of an IoT device; however, further security improvements are required in the next phases.

Appendix A - Instructions

Code Dependencies

Several prerequisites are required to be able to run this system. The following software needs to be installed ahead of completing the initial set up of the system.

Name	Link	Rationale
Windows Subsystem for Linux (WSL 2)	https://docs.microsoft.com/en-us/windows/wsl/install	Required to run the simulation (only necessary when you do not operate within Linux)
Docker Desktop	https://www.docker.com/products/docker-desktop	Required to run the simulation
Visual Studio (VS) Code	https://code.visualstudio.com/download	Required to manage code
VS Code extension “Docker”	https://code.visualstudio.com/docs/containers/overview	Required to manage code
VS Code extension “Remote - WSL”	https://code.visualstudio.com/docs/remote/wsl	Required to manage code (only necessary when you do not operate within Linux)

Table 2

Initial Set Up Instructions

1. Clone the code from the Github Repository
2. To compose & start the broker:
    a. Open a Linux shell in VS Code
    b. Navigate to .../MQTT_broker_Mosquitto
    c. Start the broker with the following command:
    $ sudo docker-compose up
3. Build docker image from controller and node Dockerfiles
    a. Right click on controller Dockerfile & click build image named ‘controller’
    b. Right click on controller Dockerfile & click build image named ‘thermostat’
    c. Right click on controller Dockerfile & click build image named ‘lamp’
4. Run thermostat
    a. Open shell
    b. Start the thermostat with the following command
    $ docker run -i --name thermo_001 thermostat
    c. You will get the message: “Integrate the node in Docker network and hit Enter”
    d. Before pressing Enter, open a Windows shell and type: $ docker network connect mqtt_broker_mosquitto_default thermo_001
    e. Hit enter in the shell where you had the “Integrate the node in Docker network and hit Enter” message.
5. Run lamp
    a. Open shell

    b. Start the lamp with the following command
       $ docker run -i --name lamp_001 lamp
    c. You will get the message: “Integrate the node in Docker network and hit Enter”
    d. Before pressing Enter, open a Windows shell and type: $ docker network connect mqtt_broker_mosquitto_default lamp_001
    e. Hit enter in the shell where you had the “Integrate the node in Docker network and hit Enter” message.
6. Run controller
    a. Open shell
    b. Start the controller with the following command:
       $ docker run -i --name controller_001 controller
    c. You will get the message: “Integrate the node in Docker network and hit Enter”
    d. Before pressing Enter, open a Windows shell and type: $ docker network connect mqtt_broker_mosquitto_default controller_001
    e. Hit enter in the shell where you had the “Integrate the node in Docker network and hit Enter” message.

Operational Instructions

Light

To operate the lamp, follow the on-screen instructions in the controller terminal.

Thermostat

To operate the thermostat, follow the on-screen instructions in the controller terminal.

Appendix B - Testing

When designing this system, the chosen QoS priorities are functional suitability, security and performance efficiency based on the ISO 25010 standard (Calidad Software, N.D.). These were chosen in line with the assignment requirements to produce a secure smart home system that makes allowance for the challenges of distributed systems, which are often performance related. The following testing assesses the extent to which this solution meets these objectives.

Functional Testing

With regards to functional suitability, this testing focuses on the solution's coverage of intended use cases as well as the integrity of the functions through successfully operating the IoT device settings from the controller. The efficiency of the functions will be covered in the 'Performance Efficiency' section below.

Use Cases

- Turn light on/off

At the time of testing, we were unable to test use cases for the lamp. This was due to the authorisation set up, we were unable to access the lamp itself to see if it could be changed.

- Turn temperature up/down

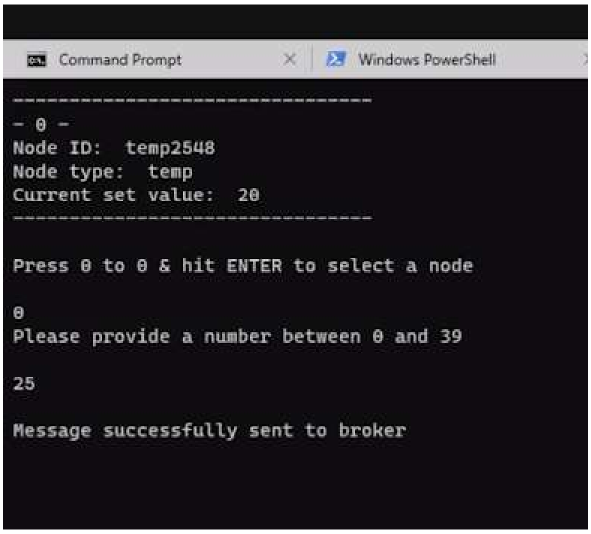


Image 1 Change Temperature

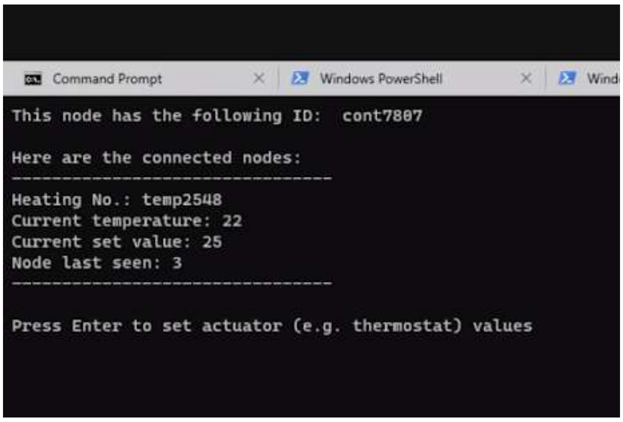


Image 2 Proof of Change in Temperature

Integrity of Functions

- Input validation (string and incorrect numbers)

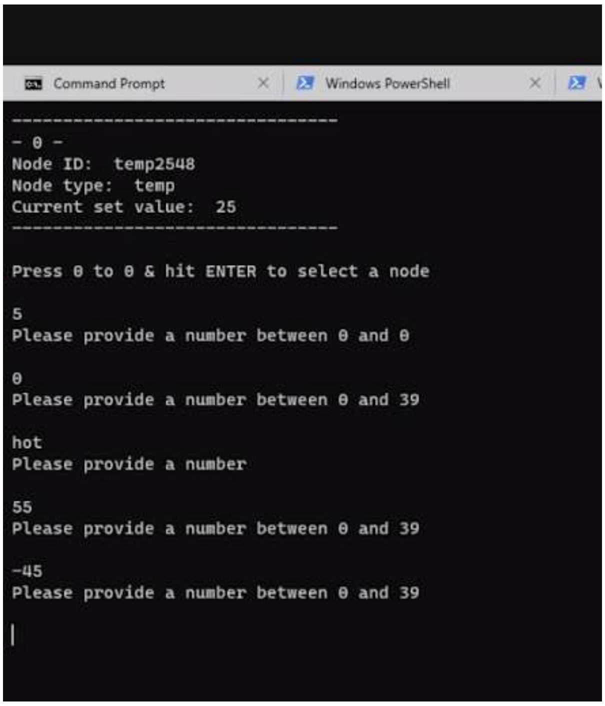


Image 3 Input Validation

It was not possible to test input validation for the second node (lamp). This was due to the authorisation set up, meaning we were not able to make any changes to the lamp.

Efficiency of Functions

- Competing messages/collisions - throughout testing we did not see any errors to suggest competing messages or collisions.

Non-Functional Testing

Security

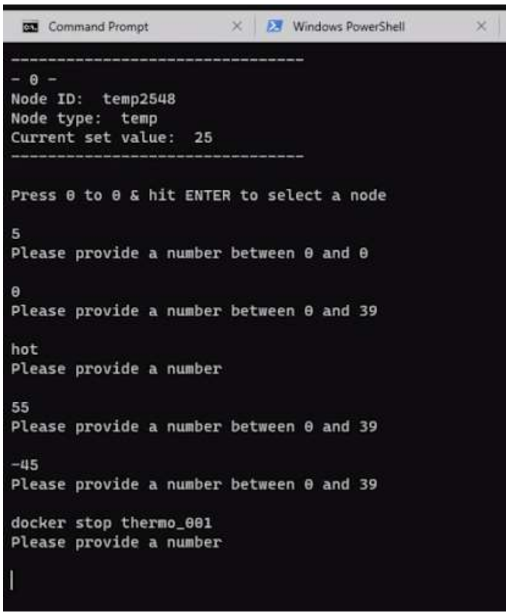
Security testing has been performed at all layers, including application (SAST, etc.) and network (Wireshark, etc.), however physical security and black box testing is not within scope as this is an unhosted simulation only. Multiple security testing tools are used to achieve wide coverage of possible security vulnerabilities and testing has been tailored to the underlying technologies, e.g. Snyk for Docker image scanning.

Abuse and misuse cases from the perspective of an attacker have been identified, these include:

Input validation

- Temp >40 on thermostat - Please refer to Image 3 above.

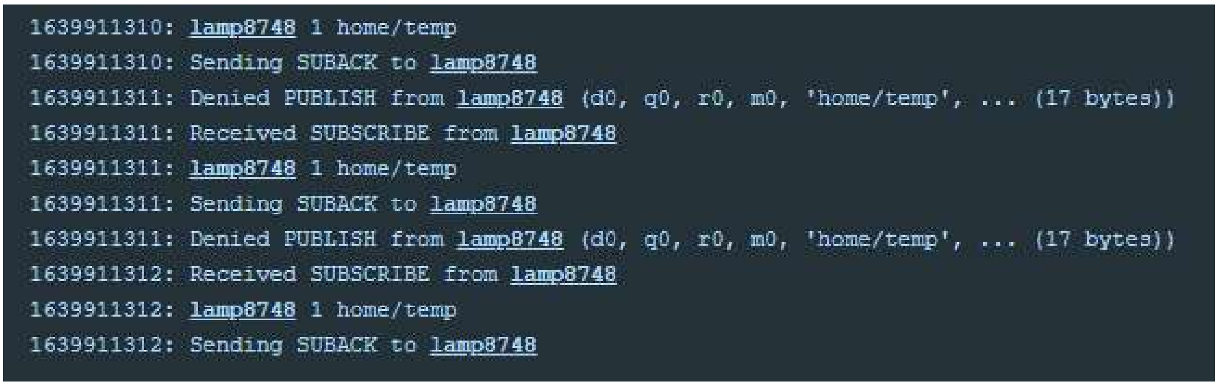
- Docker stop string



```
Command Prompt    Windows PowerShell
--------------------------------
- 0 -
Node ID:  temp2548
Node type:  temp
Current set value:  25
--------------------------------

Press 0 to 0 & hit ENTER to select a node

5
Please provide a number between 0 and 0

0
Please provide a number between 0 and 39

hot
Please provide a number

55
Please provide a number between 0 and 39

-45
Please provide a number between 0 and 39

docker stop thermo_001
Please provide a number
```

Image 4 Docker Stop

Authorisation Checks



```
1639911310: lamp8748 1 home/temp
1639911310: Sending SUBACK to lamp8748
1639911311: Denied PUBLISH from lamp8748 (d0, q0, r0, m0, 'home/temp', ... (17 bytes))
1639911311: Received SUBSCRIBE from lamp8748
1639911311: lamp8748 1 home/temp
1639911311: Sending SUBACK to lamp8748
1639911311: Denied PUBLISH from lamp8748 (d0, q0, r0, m0, 'home/temp', ... (17 bytes))
1639911312: Received SUBSCRIBE from lamp8748
1639911312: lamp8748 1 home/temp
1639911312: Sending SUBACK to lamp8748
```

Image 5 Authorisation Check

Whilst it was out of the scope of this testing, it is recommended to penetration test the system in order to determine vulnerability to more sophisticated misuse and abuse cases, for example credential abuse and malicious nodes attempting to connect to the network.

Application Security

Static Application Security Testing (SAST)

LGTM was used throughout development to identify and address code vulnerabilities (LGTM, 2018). Bandit has then been used to provide further coverage of possible vulnerabilities with some novel vulnerabilities identified (Image 7) (PyCQA, N.D.).

LGTM

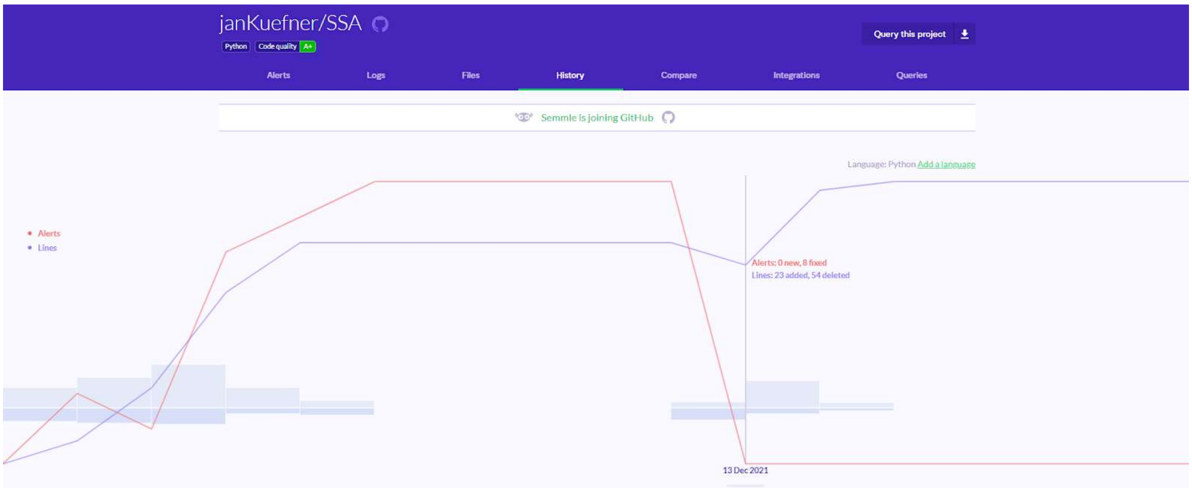
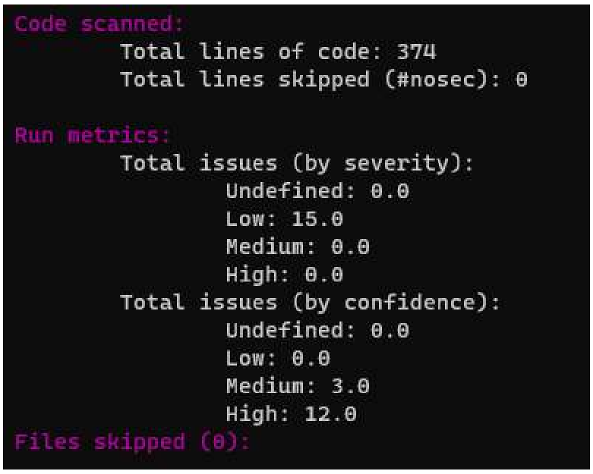


Image 6 LGTM Alert History

For further insight into the LGTM results, please see the dashboard for this code repository.

Bandit

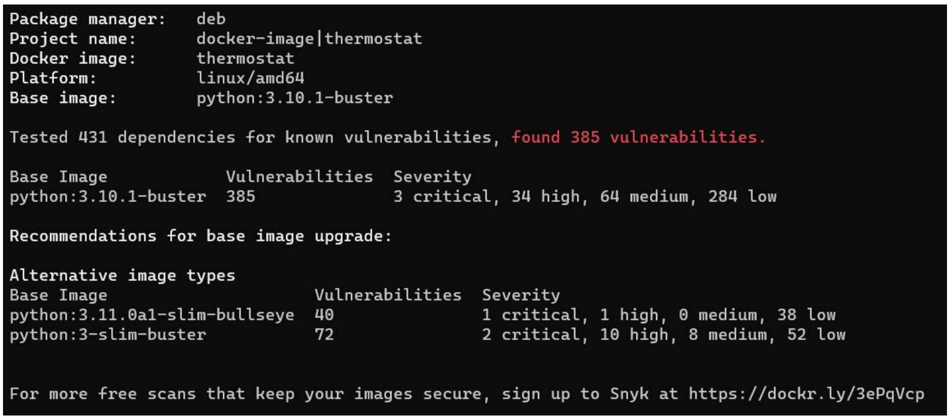


```
Code scanned:
        Total lines of code: 374
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0.0
                Low: 15.0
                Medium: 0.0
                High: 0.0
        Total issues (by confidence):
                Undefined: 0.0
                Low: 0.0
                Medium: 3.0
                High: 12.0
Files skipped (0):
```

Image 7 Bandit Summary

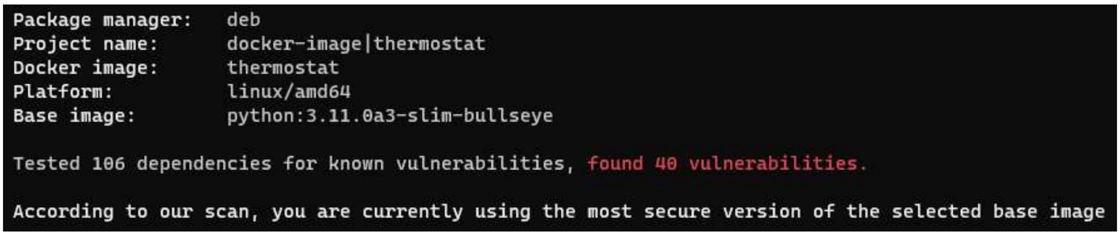
Docker image Scan (Snyk)

Snyk was used to scan the Docker images for the Thermostat, Lamp, Controller and the Broker (Docker, N.D.). The screenshots below show that the original thermostat base image was linked to 385 vulnerabilities and the recommendation to use a more secure base image was followed. Although vulnerabilities are still present, the scan has also informed us that currently this is the most secure version of the selected image.



```
Package manager:   deb
Project name:      docker-image|thermostat
Docker image:      thermostat
Platform:          linux/amd64
Base image:        python:3.10.1-buster

Tested 431 dependencies for known vulnerabilities, found 385 vulnerabilities.

Base Image            Vulnerabilities  Severity
python:3.10.1-buster  385              3 critical, 34 high, 64 medium, 284 low

Recommendations for base image upgrade:

Alternative image types
Base Image                   Vulnerabilities  Severity
python:3.11.0a1-slim-bullseye  40             1 critical, 1 high, 0 medium, 38 low
python:3-slim-buster           72             2 critical, 10 high, 8 medium, 52 low


For more free scans that keep your images secure, sign up to Snyk at https://dockr.ly/3ePqVcp
```

Image 8 Thermostat Before



```
Package manager:   deb
Project name:      docker-image|thermostat
Docker image:      thermostat
Platform:          linux/amd64
Base image:        python:3.11.0a3-slim-bullseye

Tested 106 dependencies for known vulnerabilities, found 40 vulnerabilities.

According to our scan, you are currently using the most secure version of the selected base image
```

Image 9 Thermostat After

Port Scanning

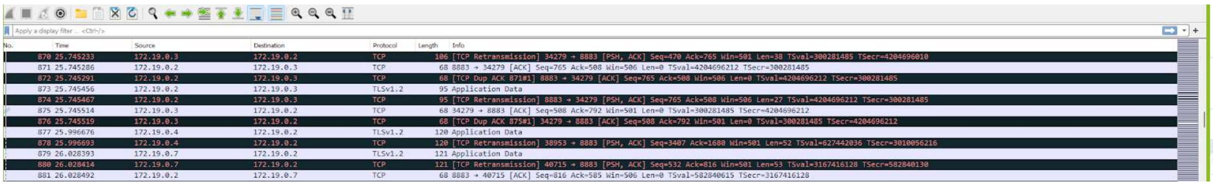
Each container has an IPv4 address to scan and nmap has been used to demonstrate port restrictions in place on the containers.

```
┌──(root㉿ a30245af31e0)-[/]
└─# nmap -p1-9999 172.24.03
Starting Nmap 7.92 ( https://nmap.org ) at 2021-12-05 10:02 UTC
Nmap scan report for 172.24.03 (172.24.0.3)
Host is up (0.000011s latency).
rDNS record for 172.24.0.3: thermo_001.mqtt_broker_mosquitto_default
All 9999 scanned ports on 172.24.03 (172.24.0.3) are in ignored states.
Not shown: 9999 closed tcp ports (reset)
MAC Address: 02:42:AC:18:00:03 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 58.58 seconds

┌──(root㉿ a30245af31e0)-[/]
└─#
```

Image 10 NMap

Network Security

A tcpdump packet capture has been used with Wireshark to demonstrate the TLS encrypted network communications (Docker Hub, 2016). TLS v1.2 was in use (Image 11).



No.	Time	Source	Destination	Protocol	Length	Info
870	25.745233	172.19.0.3	172.19.0.2	TCP	106	[TCP Retransmission] 34279 → 8883 [PSH, ACK] Seq=470 Ack=765 Win=501 Len=38 TSval=300281485 TSecr=4204696010
871	25.745286	172.19.0.2	172.19.0.3	TCP	68	8883 → 34279 [ACK] Seq=765 Ack=508 Win=506 Len=0 TSval=4204696212 TSecr=300281485
872	25.745291	172.19.0.2	172.19.0.3	TCP	68	[TCP Dup ACK 871#1] 8883 → 34279 [ACK] Seq=765 Ack=508 Win=506 Len=0 TSval=4204696212 TSecr=300281485
873	25.745456	172.19.0.2	172.19.0.3	TLSv1.2	95	Application Data
874	25.745467	172.19.0.2	172.19.0.3	TCP	95	[TCP Retransmission] 8883 → 34279 [PSH, ACK] Seq=765 Ack=508 Win=506 Len=27 TSval=4204696212 TSecr=300281485
875	25.745514	172.19.0.3	172.19.0.2	TCP	68	34279 → 8883 [ACK] Seq=508 Ack=792 Win=501 Len=0 TSval=300281485 TSecr=4204696212
876	25.745519	172.19.0.3	172.19.0.2	TCP	68	[TCP Dup ACK 875#1] 34279 → 8883 [ACK] Seq=508 Ack=792 Win=501 Len=0 TSval=300281485 TSecr=4204696212
877	25.996676	172.19.0.4	172.19.0.2	TLSv1.2	120	Application Data
878	25.996693	172.19.0.4	172.19.0.2	TCP	120	[TCP Retransmission] 38953 → 8883 [PSH, ACK] Seq=3407 Ack=1680 Win=501 Len=52 TSval=627442036 TSecr=3010056216
879	26.028393	172.19.0.7	172.19.0.2	TLSv1.2	121	Application Data
880	26.028414	172.19.0.7	172.19.0.2	TCP	121	[TCP Retransmission] 40715 → 8883 [PSH, ACK] Seq=532 Ack=816 Win=501 Len=53 TSval=3167416128 TSecr=582840130
881	26.028492	172.19.0.2	172.19.0.7	TCP	68	8883 → 40715 [ACK] Seq=816 Ack=585 Win=506 Len=0 TSval=582840615 TSecr=3167416128

Image 11 TCP Dump

Performance Efficiency

The focus of this testing will be the efficiency of the solution functions and components. Time behaviour is covered to ensure that individual functions would meet the expectations of an end user. Resource utilisation is also tested, and capacity estimated, with a view to establishing if the system can be suitably scaled, for a large home setting for example.

Time Behaviour

Based on the system use cases, e.g. changing temperature, the functions worked within expected timeframes.

Resource Utilization

Docker stats has been used to demonstrate various performance metrics of the containers running, for example CPU and memory usage (Docker, 2018). The usage suggests that the system could be easily scaled to a large domestic property given the resources available. If control groups were implemented, the system could be tested as a whole and this could be valuable when making scaling decisions.

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
f550e767a964	thermo_001	0.05%	14.5MiB / 7.632GiB	0.19%	212kB / 277kB	0B / 0B	2

Image 12 CPU Thermostat

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
2bc4a30ccbce	lamp_001	0.37%	14.64MiB / 7.632GiB	0.19%	1.27MB / 1.52MB	0B / 0B	2

Image 13 CPU Lamp

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
afba8ed1986d	controller_001	0.29%	14.01MiB / 7.632GiB	0.18%	453kB / 635kB	0B / 0B	2

Image 14 CPU Controller

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
255abc42fbce	mosquitto_container	0.21%	1.516MiB / 7.632GiB	0.02%	2.74MB / 2.17MB	0B / 0B	1

Image 15 CPU Container

Scalability has also been tested with more than one thermostat added to the controller.

```
Windows Powe   Windows Powe   Windows Powe   Windows Powe
This node has the following ID:  cont7807

Here are the connected nodes:
--------------------------------
Heating No.: temp2548
Current temperature: 26
Current set value: 25
Node last seen: 2
--------------------------------
Heating No.: temp2478
Current temperature: -2
Current set value: 0
Node last seen: 1
--------------------------------

Press Enter to set actuator (e.g. thermostat) values
```

Image 16 Multiple Devices

Style

This code has been written in line with PEP8 Python best practice to ensure readability and maintainability (Python, 2013). The Flake8 linter has been chosen due to its rich features (Flake8, 2021). Before initial testing, there were hundreds of alerts. The team then started using the Visual Studio Code extension to ensure continuous adherence to style and this was successful (Image 17).

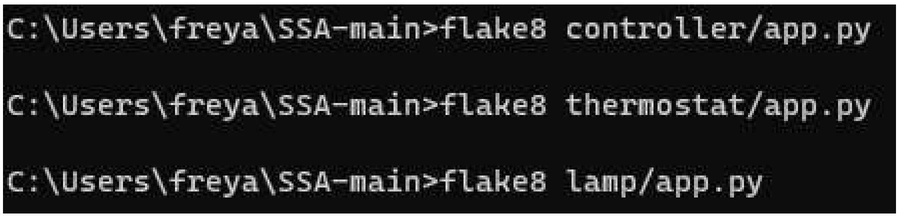


Image 17 Flake8

Appendix C - Design Decisions

System Design

Below are the further SysML diagrams required to formally model the Smart Home platform.

Figure 2 is the package diagram showing elements that are part of the SysML models for this requirement. The Activity diagram (Figure 3) highlights one aspect of the Smart Home platform in activating a light bulb. The Block Design Diagram (Figure 4) shows the architectural elements of the Smart Home platform. The Internal Design Diagram is a representation of the communication pathways and protocols utilised.

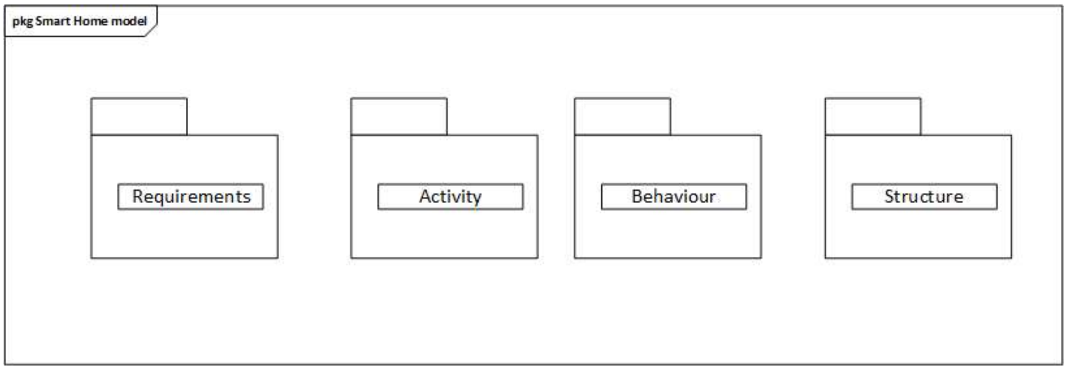


Figure 2: Package Diagram

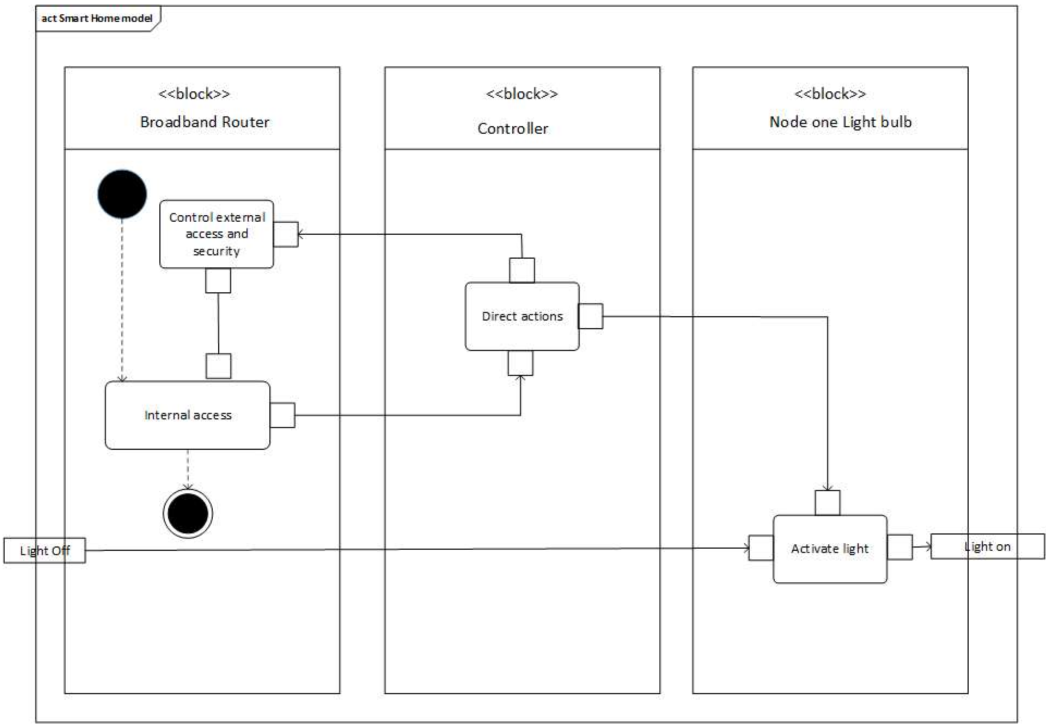


Figure 3: Activity Diagram

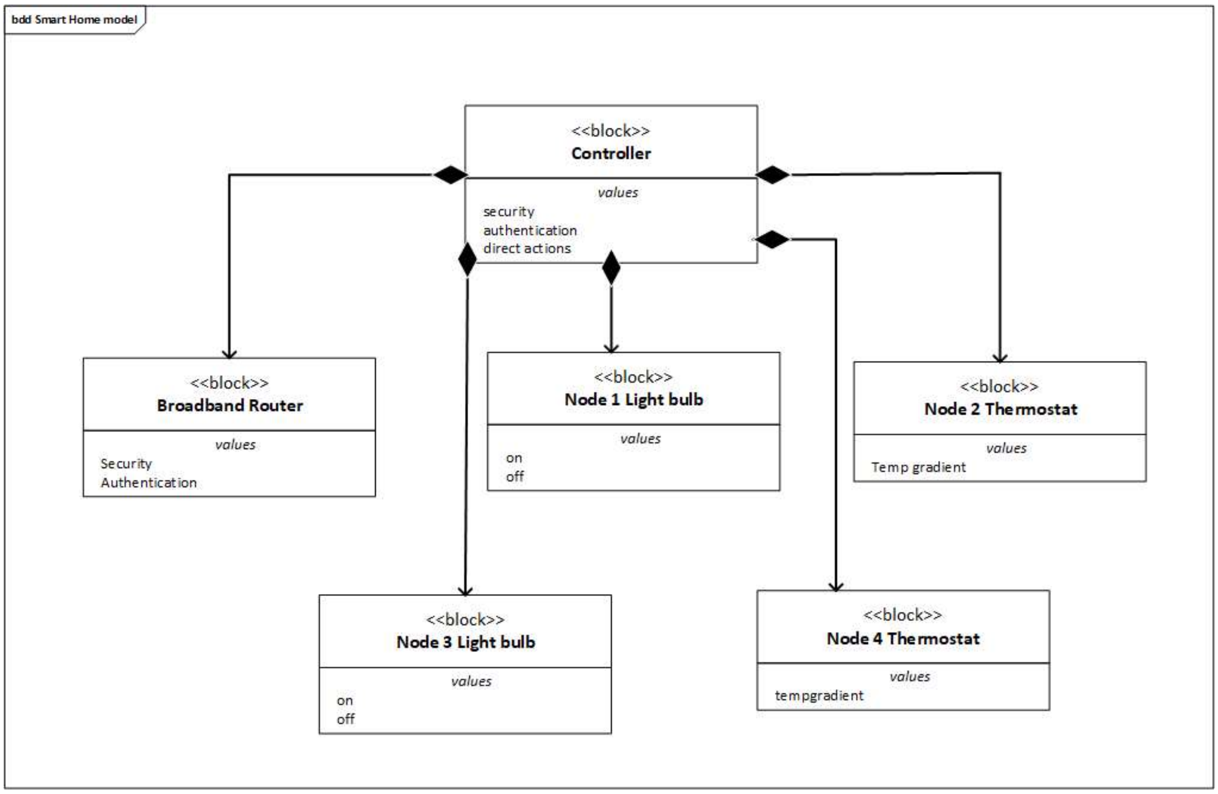


Figure 4: Block Definition diagram

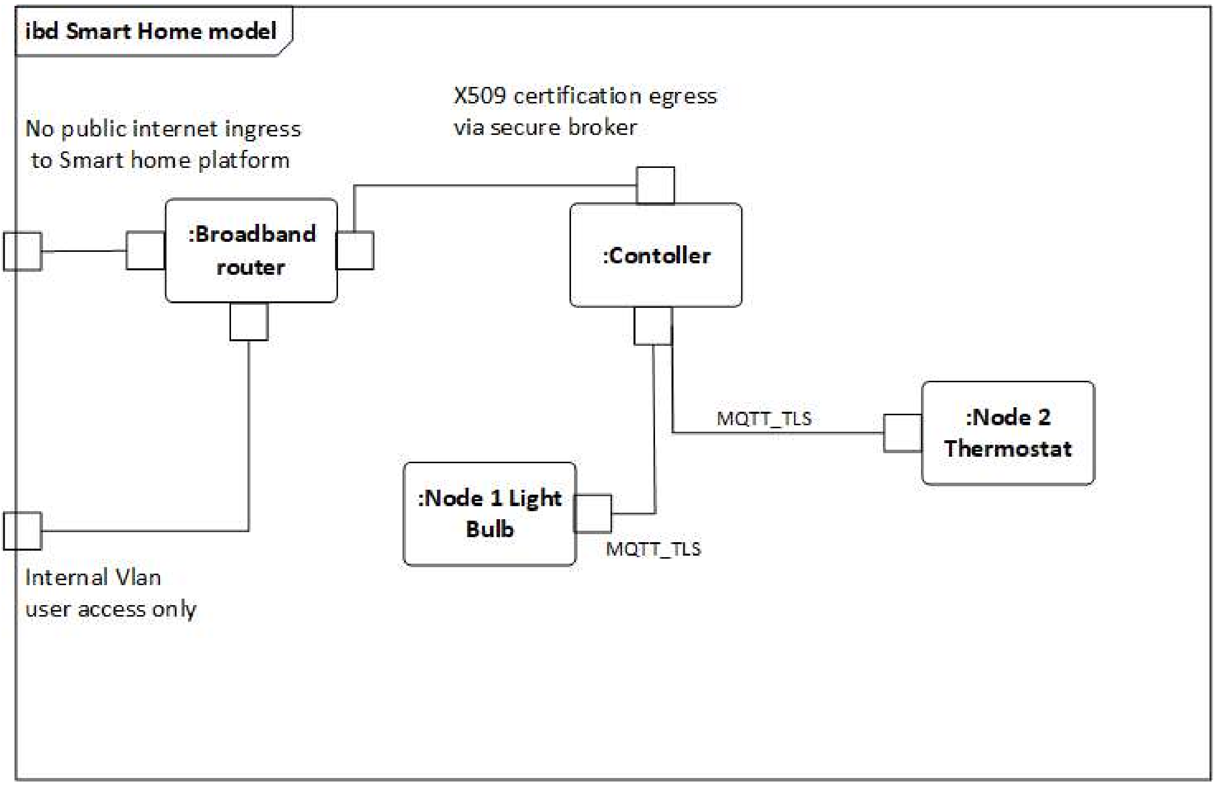


Figure 5: Internal Block Diagram

A microservices architecture is suitable for a Smart Home system because the inherent flexibility means it can be easily adapted to individual user requirements and extended if required to accommodate new, heterogenous devices (Luntovskyy & Spillner, 2017). This solution is based on a containers architecture because containers are versatile, lightweight and overall more efficient to use for microservices development than virtual machines (Kozhirbayev & Sinnott, 2017). Docker containers have been chosen as Docker is well-documented and simple to use, making it suitable for developers with less experience. Docker is also compatible with all major cloud platforms which provides a great deal of options for extending the functionality of this proof of concept.

Application Layer	MQTT
Transport Layer	TCP
Network Layer	IPv4
Physical Layer	WiFi (IEEE 802.11 a/b/g/n)

Figure 6

Figure 6 shows the technology decisions made for each layer corresponding to the TCP/IP stack (Gerber & Romeo, 2017). The decision was taken to use well-established, open-source protocols. This ensures reliable communication between the architecture layers (White et al., 2017), interoperability with additional devices and ultimately provides future-proofing for the solution (Irons-McLean et al., 2019).

Vulnerability Management

Table 3 shows the vulnerabilities enumerated as part of the Unit 3 assignment (Wilson et al., 2021). Phase 1 reflects the scope of the simulation for this assignment. Phase 1 controls have been prioritised according to what can be implemented within the limitations of the simulation and within the given timelines. Phase 2 reflects what would be built to further enhance security in a real world situation.

No.	Recommended Mitigations (Unit 3)	Phase 1 (Implemented)	Phase 2
1	Implement auto-update function by default, including security patching.	N/A - Need internet connectivity.	✓
2	End User Education - social engineering and home physical security measures.	N/A - Proof of concept only.	✓
3	In-built logging and basic Intrusion Detection System (IDS) on the controller.	✓ Broker logging enabled.	✓ IDS automation overlay.
4	Firewall on controller, with use of access control lists and port restrictions.	✓ ACL & port restrictions implemented - only ports 1883 & 8883 are open.	✓ Full firewall implementation.
5	Antivirus on the controller with regular updates required.	N/A - Proof of concept only.	✓
6	Bluetooth out of band pairing (OOB) via near field communication (NFC) for device authentication.	N/A - Physical components out of scope.	✓
7	Two-factor authentication (2FA) for the end user to authenticate with the controller to change settings and perform other actions.	✓ Authentication implemented via username & password.	✓ 2FA overlay required.
8	Employ randomly generated, strong passwords on the controller by default and encourage users to change to a chosen strong password on setup.	N/A - Proof of concept only.	✓
9	All communications should be Transport Layer Security (TLS) encrypted with strong ciphers enforced.	✓ TLS encrypted communications.	✓ Strong ciphers need to be enforced.
10	Hardware secure key storage via dedicated chip.	N/A - Physical components out of scope.	✓
11	Physical security controls to protect devices from tampering during manufacturing through to end user operation.	N/A - Physical components out of scope.	✓
12	Restrict access to ports, such as Secure Shell (SSH) on port 22.	✓ Port restrictions in place - only ports 1883 & 8883 are open.	N/A
13	Session management controls to support authentication on the controller, such as automatically logging the user off after one hour.	N/A - Lower priority control not implemented.	✓
14	Predetermined authorised actions based on device type, e.g. temperature alert via email but smart bulb has no authorisation to email.	✓ Authorisation of actions concept implemented.	N/A
15	Following secure development practices, such as input validation.	✓ Input validation included.	✓ Use additional secure techniques, such as safe failures.
16	Verify software prior to download. For example, verify checksums when downloading software packages during development and verify signatures before installing firmware updates.	✓ Software verified.	✓ Firmware updated verification implementation.
17	Ensure data minimisation on nodes and controller and sensitive data encrypted at rest.	N/A - Physical components out of scope.	✓

18	Robust third party supplier security assurance, for chip manufacturers for example.	N/A - Proof of concept only.	✓
19	Use reputable open source software libraries.	✓ Reputable software used & lower risk base images implemented.	N/A
20	Function for the operator to wipe personal data to support secure disposal.	N/A - Proof of concept only.	✓
21	Rate limit pairing attempts with the nodes to prevent malicious battery drainage.	N/A - Physical components out of scope.	✓

Table 3

Documentation References

Calidad Software (N.D.) ISO/IEC 25010. Available from: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010 [Accessed 18 December 2021].

Docker (2018) Runtime Metrics. Available from: https://docs.docker.com/config/containers/runmetrics/ [Accessed 5 December 2021]

Docker (N.D.) Vulnerability scanning for Docker local images. Available from: https://docs.docker.com/engine/scan/ [Accessed 19 December 2021].

Docker Hub (2016) TCPDump. Available from: https://hub.docker.com/r/kaazing/tcpdump [Accessed 11 December 2021]

Flake8 (2021) Flake8: Your Tool For Style Guide Enforcement. Available from: https://flake8.pycqa.org/en/latest/ [Accessed 18 December 2021].

Gerber, A. & Romeo, J. (2017) *Connecting all the Things in the Internet of Things. IBM Developer.* https://developer.ibm.com/articles/iot-lp101-connectivity-network-protocols/

Irons-McLean, R., Sabella, A. & Yannuzzi, M. (2019) *IoT and Security Standards and Best Practices. Cisco Press.* https://www.ciscopress.com/articles/article.asp?p=2923211&seqNum=6

Kozhirbayev, Z. & Sinnott, R. (2017). A performance comparison of container-based technologies for the Cloud. *Future Generation Computer Systems* 68: 175–182.

LGTM (2018) About LGTM. Available from: https://lgtm.com/help/lgtm/about-lgtm [Accessed 12 December 2021]

Lundqvist, C., Keranen, A., Smeets, B., Fornehed, J., Azevedo, C. & von Wrycza, P. (2019) Key Technology Choices for Optimal Massive IoT Devices. *Ericsson Technology Review.*

Luntovskyy, A. & Spillner, J. (2017) *Architectural Transformations in Network Services and Distributed Systems*. Springer.

Mocnej, J., Pekar, A., Seah, W. K. G., Kajati, E. & Zolotova, I. (2019) Internet of Things Unified Protocol Stack. *Acta Electrotechnica et Informaticap* 19(2): 24-32.

MQTT (2021) MQTT: The Standard for IoT Messaging. Available from: https://mqtt.org/ [Accessed 4 December 2021].

OWASP (2018) OWASP Top 10 Internet of Things. Available from: https://owasp.org/www-pdf-archive/OWASP-IoT-Top-10-2018-final.pdf [Accessed 4 December 2021]

PyCQA (N.D.) Bandit. Available from: https://github.com/PyCQA/bandit [Accessed 19 December 2021].

Python (2013) PEP 8 -- Style Guide for Python Code. Available from: https://www.python.org/dev/peps/pep-0008/ [Accessed 18 December 2021].

Sardeshmukh, H. & Ambawade, D. (2017) Internet of Things: Existing Protocols and Technological Challenges in Security. *International Conference on Intelligent Computing and Control*: 1-7.

Sherozhenko, M. (2017) MQTT vs. HTTP: which one is the best for IoT?. Available from: https://medium.com/mqtt-buddy/mqtt-vs-http-which-one-is-the-best-for-iot-c868169b3105 [Accessed 3 December 2021]

The Digital Life (2021) Docker VSCode Python Tutorial: Run your App in a Container. Available from: https://www.youtube.com/watch?v=jtBVppyfDbE [Accessed 14 December 2021].

White, G., Nallur, V. & Clarke, S. (2017) Quality of Service Approaches in IoT: A Systematic Mapping. *Journal of Systems and Software* 132: 186-203.

Wilson, C., Basey, F., Watts, C., Isic, D. & Küfner, J. (2021) *Design Document for Secure Systems Architecture (SSA): Smart Home Vulnerability Report.* Essex: Kaplan.

Code References

Bruh Automation (2016) How-To Get Started with Mosquitto MQTT Broker on a Raspberry Pi. Available from: https://www.youtube.com/watch?v=AsDHEDbyLfg [Accessed 10 December 2021].

Ev3Dev (2016) Sending and Receiving messages with MQTT. Available from: https://www.ev3dev.org/docs/tutorials/sending-and-receiving-messages-with-mqtt/ [Accessed 10 December 2021].

Fireship (2020) Learn Docker in 7 Easy Steps - Full Beginner's Tutorial. Available from: https://www.youtube.com/watch?v=gAkwW2tuIqE& [Accessed 10 December 2021].

Gram, N. (2019) How To Containerize Your Python Application using Docker. Available from: https://www.youtube.com/watch?v=gHaFqYRJ_aw [Accessed 10 December 2021].

OpenNest (2021) MQTTS : How to use MQTT with TLS?. Available from: https://openest.io/en/2020/01/03/mqtts-how-to-use-mqtt-with-tls/ [Accessed 13 December 2021].

Pelaez, A. (2021)Security: Connect to Ubidots MQTT broker with TLS Security. Available from:https://help.ubidots.com/en/articles/1083734-security-connect-to-ubidots-mqtt-broker-with-tls-security [Accessed 13 December 2021].

Stack Overflow (2009) Finding local IP addresses using Python's stdlib. Available from: https://stackoverflow.com/questions/166506/finding-local-ip-addresses-using-pythons-stdlib [Accessed 15 December 2021].

Stack Overflow (2013) How to get a Docker container's IP address from the host. Available from: https://stackoverflow.com/questions/17157721/how-to-get-a-docker-containers-ip-address-from-the-host [Accessed 15 December 2021].

Stack Overflow (2014) How can I limit the user input to only integers in Python. Available from: https://stackoverflow.com/questions/23326099/how-can-i-limit-the-user-input-to-only-integers-in-python [Accessed 15 December 2021].

Stack Overflow (2018) Can mosquitto forward the ClientID of a messages sender?. Available from: https://stackoverflow.com/questions/51963413/can-mosquitto-forward-the-clientid-of-a-messages-sender [Accessed 15 December 2021].

Stack Overflow (2019) Cannot ping from one Docker container to another. Available from: https://stackoverflow.com/questions/59277602/cannot-ping-from-one-docker-container-to-another [Accessed 15 December 2021].

Stack Overflow (2019) "unknown ca" with self-generated CA, certificates and client/server. Available from: https://stackoverflow.com/questions/53104296/unknown-ca-with-self-generated-ca-certificates-and-client-server [Accessed 15 December 2021].

Steve's Internet Guide (2021) Python MQTT Client Connections– Working with Connections. Available from: http://www.steves-internet-guide.com/client-connections-python-mqtt/ [Accessed 11 December 2021].

The Digital Life (2021) Docker VSCode Python Tutorial Run your App in a Container. Available from: https://www.youtube.com/watch?v=jtBVppyfDbE [Accessed 10 December 2021].