June 13, 2022
Final Document (653 words)

Secure Systems Architecture Final Document

Group 1
Shota Kameyama
Ying Chan
Austin Mundy
Mathew van Beek

Table of Contents

List of Figures

List of Tables

1. Prerequisite

The following should be installed already before setup.

- Python3
- Mosquitto Installed

1.1. How to install mosquitto

- MacOS: `brew install mosquitto`
- Debian: `apt-get update && apt-get install mosquitto`

2. Getting Started

2.1. Install Necessary Source Code and Libraries

```
git clone https://github.com/ShotaKameyama/ssa_iot.git
cd ssa_iot
make install
```

2.2. Setup env variables and mosquitto config files.

As we use access control using username and password, hence we use env variables to avoid info leakage by the source code.

```
chmod +x setup.sh
./setup.sh
```

This shell script will take care of the following:

1. Create Mosquitto Access Control List File: `./config/mosquitto.acl`
2. Create Mosquitto User Credential File: `./config/mosquitto.pass`
3. Create Mosquitto Config File: `./config/mosquitto.conf`
4. Add Environment Variables to `~/.bash_profile`

3. Basic Information

3.1. Instances

This IoT system consists of 4 instances:

1. MQTT broker
2. IoT Controller
3. IoT Camera
4. IoT DoorLock

You should start 4 instances parallel.

3.1.1 Instance 1

```
mosquitto -c config/mosquitto.conf
```

3.1.2. Instance 2

```
python3 iot_controller.py
```

3.1.3. Instance 3

```
python3 iot_client_doorlock.py
```

3.1.4. Instance 4

```
python3 iot_client_camera.py
```

3.2. How to read QR codes?

Once you configured 4 instances, then you can start reading QR codes using your USB camera.

```
python3 qr_read.py
```

Then read a QR file under `static/qr`

3.3. IoT Doorlock MQTT Publish

Alternatively, you can do the following to do the same.

```
python3 iot_publish_doorlock.py <Request>
```

Publish Open Request Sample

```
python3 iot_publish_doorlock.py Open
```

Publish Close Request Sample

```
python3 iot_publish_doorlock.py Close
```

3.4. Virtualization

if you need virtualization, you can use `venv`.

```
python3 -m venv pymyenv
. pymyenv/bin/activate
```

3.5. How to force authentication on Mosquitto

1. run `mosquitto -c mosquitto.conf`
if you don't have the `mosquitto.conf` file, make sure that you have run `./setup.sh`.

3.6 How to enable TLS on Mosquitto

3.6.1. Requirement

Certificate Authority (CA) server – OpenSSL for the self-sign certificate in this case. It could be signed by an online CA server for the public trust certificate.

3.6.2. In the CA server:

Generate a CA server key pair with password protection.

```
openssl genrsa -des3 -out ca.key 4096
```

Request the certificate with the required information, including Country Name, State, Locality, Organization, Unit Name, CA server hostname (Common Name) and Email address.

```
openssl req -x509 -new -key ca.key -sha256 -days 365 -out ca.crt
```

3.6.3. In the Broker server:

Generate a broker server key pair with password protection.

```
openssl genrsa -out server.key 4096
```

Request the certificate with the required information, including Country Name, State, Locality, Organization, Unit Name, broker server hostname (Common Name) and Email address.

```
openssl req -new -key server.key -sha256 -days 365 -out server.csr
```

3.6.4. In the CA server (self-sign):

Copy the request file server.csr to the CA server to verify and sign the certificate.

```
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt -days 365
-sha256
```

3.6.5. In the Broker server

Copy the signed certificate file server.crt and CA server certificate ca.crt to the Broker server to the Keystore. Update the mosquito configuration file and the related IoT device to use TLS for the MQTT transaction.

4. How to run tests

4.1. Test Report Repository

The exported documents are published below:

- Locust Result at static/reports/locust_report
- Flake8 result at static/reports/flake8_report
- Bandit result at static/reports/bandit_report

4.2. Guide Enforcement & SAST

PyLint/Flake8/Bandit are used for the test.

```
make lint
make flake8
make bandit
```

4.3. Perf Test

1. run `locust`
2. Open `http://0.0.0.0:8089/` on your browser
3. Set the values and start the test

4.3.1. Prerequisite

MacOS: You need to install the following libraries to pass `make install`

```
brew install postgresql, librdkafka, zbar, opencv
```

if M1 then run either of the following:

```
C_INCLUDE_PATH=/opt/homebrew/Cellar/librdkafka/1.8.2/include
LIBRARY_PATH=/opt/homebrew/Cellar/librdkafka/1.8.2/lib pip install confluent_kafka`
   `CPATH=/opt/homebrew/Cellar/librdkafka/1.8.2/include pip install confluent-kafka`
```

Ref: onfluent-kafka-python github issue

```
mkdir ~/lib && ln -s $(brew --prefix zbar)/lib/libzbar.dylib ~/lib/libzbar.dylib
```

5. How to contribute

To contribute to this project, follow these steps:

1. Fork this repository.
2. Create a branch: `git checkout -b <branch_name>`.
3. Make your changes and check with: `make check`
4. Commit them: `git commit -m '<commit_message>'`
5. Push to the original branch: `git push origin <branch>`
6. Create the pull request.

Alternatively, see the GitHub documentation on [creating a pull request](https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/creating-a-pull-request).

6. Discussion of vulnerabilities identified, mitigations deployed and discussion of any omissions/lack of mitigations. (653 words)

Vulnerabilities Identified and Mitigations Deployed on an MQTT Broker

As Saidi et al. (2020) examined to identify the ways to detect IoT devices connected to the Internet, protecting the Internet-connected devices is the main approach. Table 1 shows vulnerabilities identified and mitigations deployed. All of them are vulnerabilities on the MQTT broker. As each IoT client connects to the MQTT broker, if the MQTT broker is affected, the visitors can intrude into the house (by unlocking the door).

Table 1 Vulnerabilities detected and Mitigations

Vulnerabilities Detected	Description	Test Tools	Mitigations Deployed	Alternative Solutions
Packet Capturing	All the information in the MQTT protocol was shown, including user id and password.	Wireshark	TLS Encryption	
Bruteforce Attack	Once host is identified, user id and passwords were cracked.	Python-Pwn	Login by Certificate (partially deployed)	Scan auth logs and bans the host when multiple authentication failures are detected. e.g. Fail2Ban Python Library
Leaking data	All the topics & messages on the MQTT protocol were identified.	Python-Pwn	MQTT Access Control	
DoS Attack	By the DoS attack, all the messages were undelivered	Locust	Does not allow public access over the Internet	Block by Firewall, Router configuration

Once the MQTT broker host is detected using Saidi et al. (2020) methodology if no security measures are deployed, there are many ways to capture information such as packet capturing and subscribing to all topics. Even protecting access by applying access control, password cracking is easy by running the python-pwn program. Hence, a comprehensive security approach or a combination of different security measures is essential.

Investigation on Denial of Service Attacks, a vulnerability that lacks mitigations

By applying mitigation measures on packet capturing, password cracking and leaking data as described in Table 1, most critical attacks can be protected. However, we need further consideration on (Distributed) Denial of Service ((D)DoS) attacks.

For example, Figure 1 illustrates the DoS attack on an MQTT broker on Raspberry pi with invalid credentials. We identified that at around 20,000 requests per second (RPS), the MQTT broker was not able to respond because tcpd denied access, as illustrated in Figure 2.

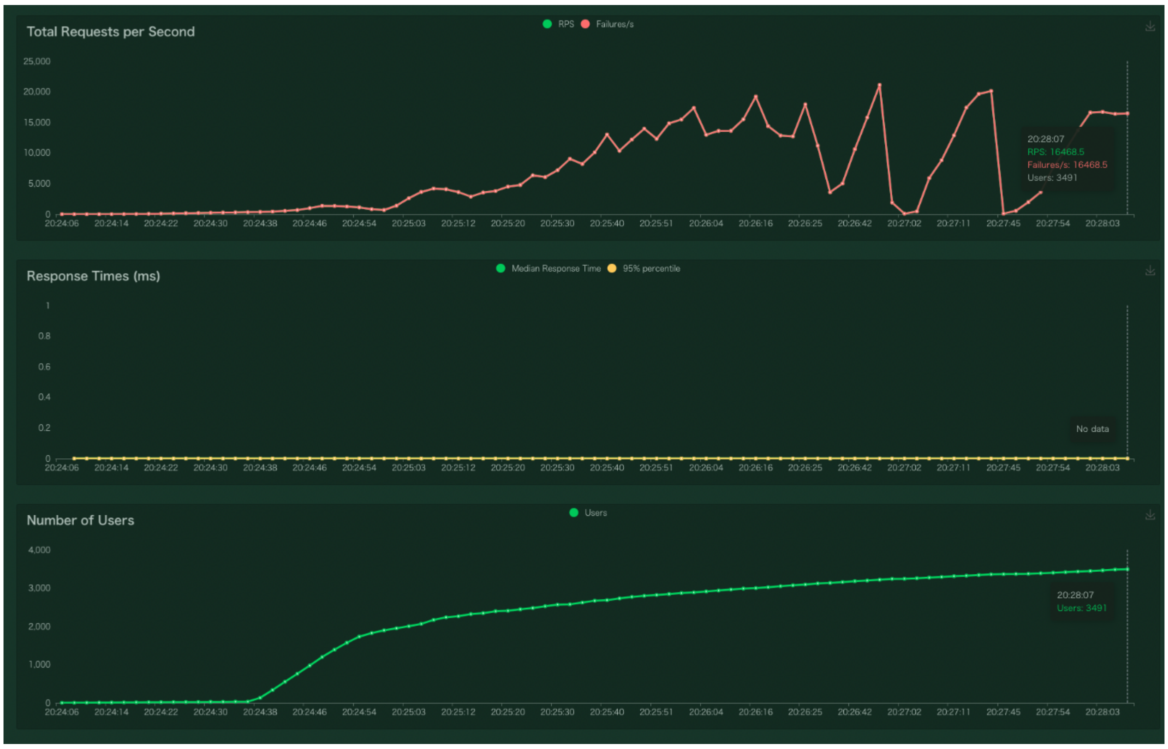
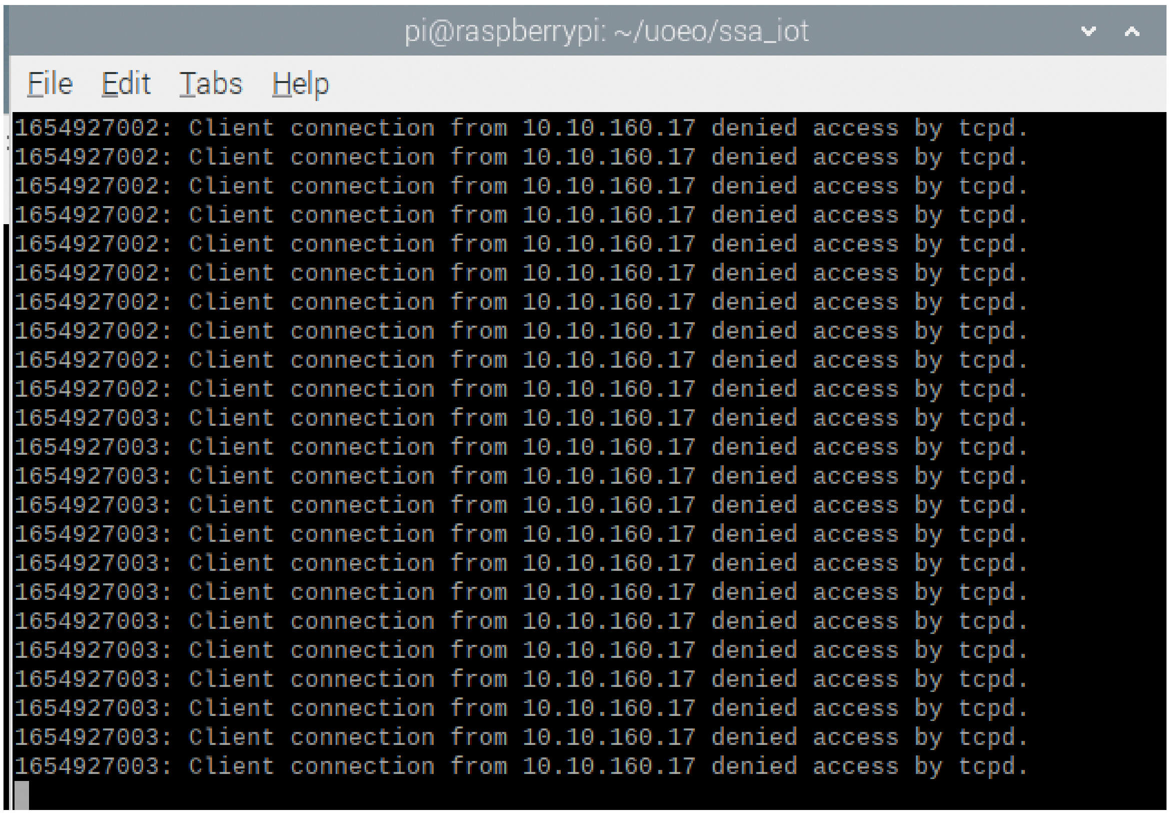


Figure 1 Locust Test



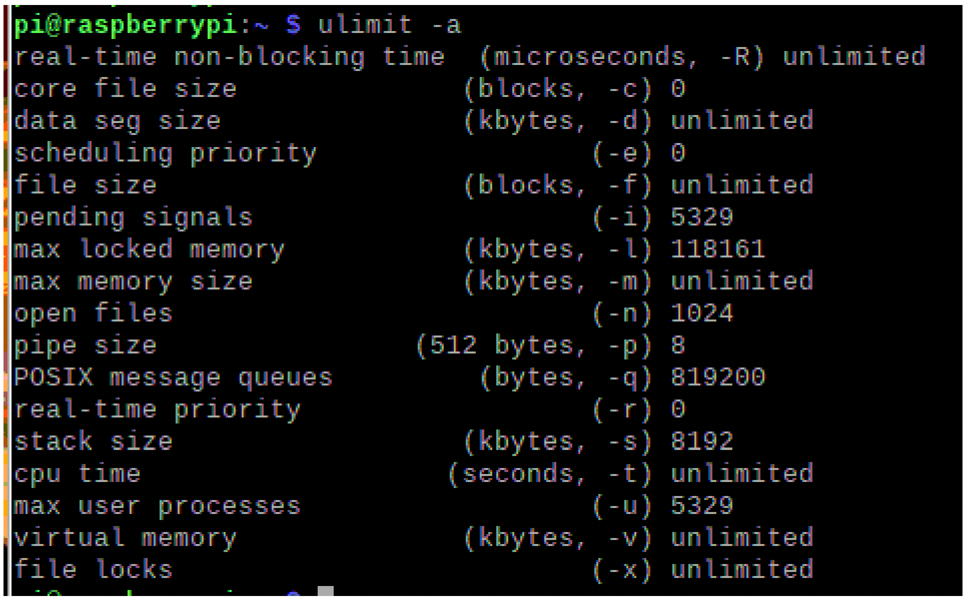
pi@raspberrypi: ~/uoeo/ssa_iot

File Edit Tabs Help

```
1654927002: Client connection from 10.10.160.17 denied access by tcpd.
1654927002: Client connection from 10.10.160.17 denied access by tcpd.
1654927002: Client connection from 10.10.160.17 denied access by tcpd.
1654927002: Client connection from 10.10.160.17 denied access by tcpd.
1654927002: Client connection from 10.10.160.17 denied access by tcpd.
1654927002: Client connection from 10.10.160.17 denied access by tcpd.
1654927002: Client connection from 10.10.160.17 denied access by tcpd.
1654927002: Client connection from 10.10.160.17 denied access by tcpd.
1654927002: Client connection from 10.10.160.17 denied access by tcpd.
1654927002: Client connection from 10.10.160.17 denied access by tcpd.
1654927003: Client connection from 10.10.160.17 denied access by tcpd.
1654927003: Client connection from 10.10.160.17 denied access by tcpd.
1654927003: Client connection from 10.10.160.17 denied access by tcpd.
1654927003: Client connection from 10.10.160.17 denied access by tcpd.
1654927003: Client connection from 10.10.160.17 denied access by tcpd.
1654927003: Client connection from 10.10.160.17 denied access by tcpd.
1654927003: Client connection from 10.10.160.17 denied access by tcpd.
1654927003: Client connection from 10.10.160.17 denied access by tcpd.
1654927003: Client connection from 10.10.160.17 denied access by tcpd.
1654927003: Client connection from 10.10.160.17 denied access by tcpd.
1654927003: Client connection from 10.10.160.17 denied access by tcpd.
1654927003: Client connection from 10.10.160.17 denied access by tcpd.
1654927003: Client connection from 10.10.160.17 denied access by tcpd.
```

Figure 2 Denial of access by tcpd on Raspberry pi

When investigating further, this might stem from the ulimit (SS64, 2022) of Raspberry Pi. Figure 3 shows the default value of ulimit in the Raspberry Pi.

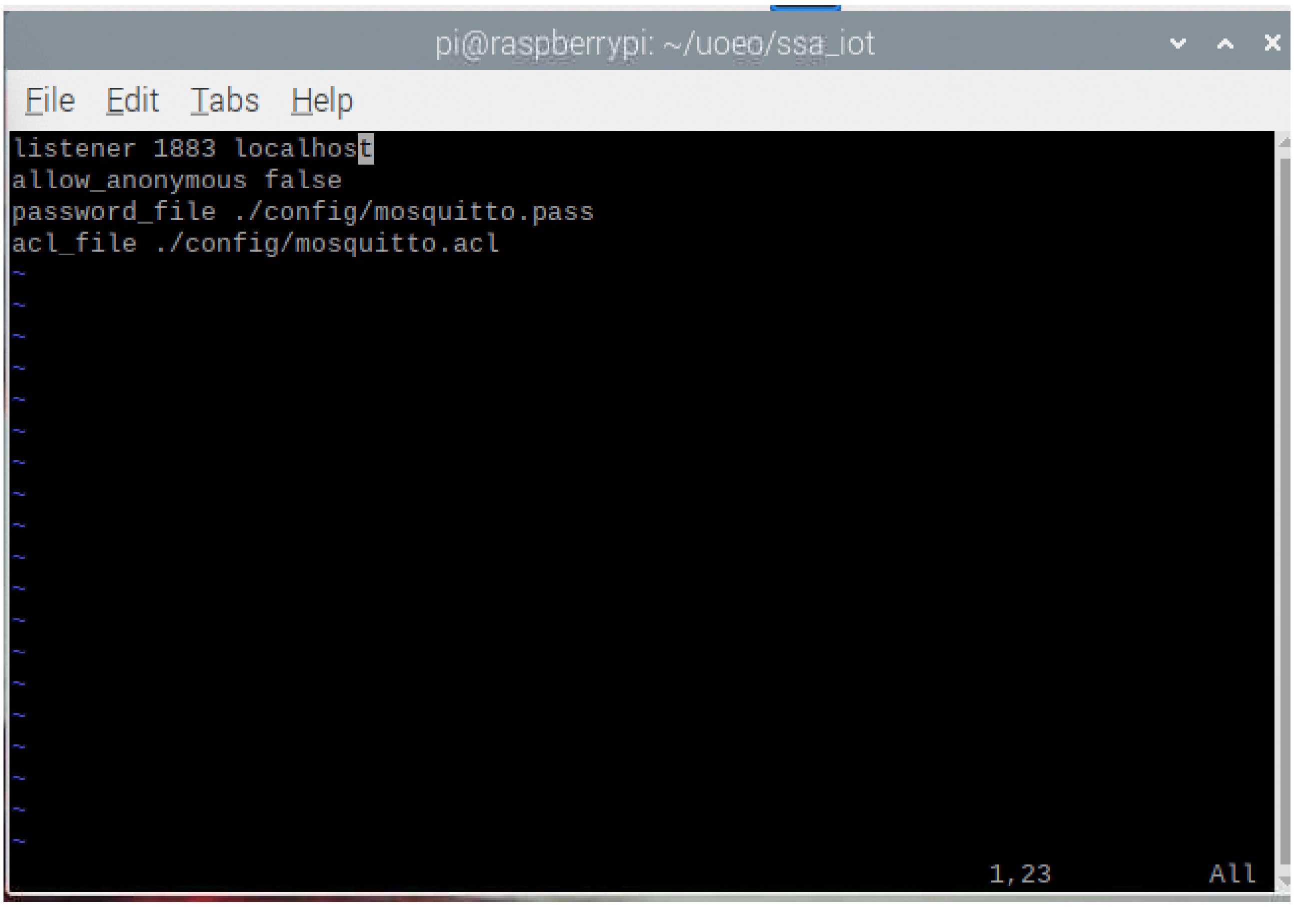


```
pi@raspberrypi:~ $ ulimit -a
real-time non-blocking time  (microseconds, -R) unlimited
core file size              (blocks, -c) 0
data seg size               (kbytes, -d) unlimited
scheduling priority                 (-e) 0
file size                   (blocks, -f) unlimited
pending signals                     (-i) 5329
max locked memory           (kbytes, -l) 118161
max memory size             (kbytes, -m) unlimited
open files                          (-n) 1024
pipe size                (512 bytes, -p) 8
POSIX message queues         (bytes, -q) 819200
real-time priority                  (-r) 0
stack size                  (kbytes, -s) 8192
cpu time                   (seconds, -t) unlimited
max user processes                  (-u) 5329
virtual memory              (kbytes, -v) unlimited
file locks                          (-x) unlimited
```

Figure 3 The Default ulimit setting

Modifying the values did not improve. However, for future study, this hardware and configuration aspect is worth further investigation.

Another approach that we may consider is to restrict the host or IP by Mosquitto configuration file. Figure 4 illustrates an example. This worked fine to prevent the simple DoS attack. However, it is easy to spoof a host or IP so this is not a perfect solution. DoS attack from the internal attack cannot be prevented in this case. Moreover, this configuration will make the configuration complicated, hence this solution will increase the cognitive load for the user, which increases another security risk from the human factor perspective. This solution should be considered as a supplement.



```
listener 1883 localhost
allow_anonymous false
password_file ./config/mosquitto.pass
acl_file ./config/mosquitto.acl
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
                                                        1,23          All
```

Figure 4 Configuring IP or Host in the Mosquitto Conf file

The best approach against DoS can be network-related countermeasures such as firewalls, DoS-prevention mechanisms on routers, switches, etc.

Vulnerabilities on IoT devices

Each IoT device has different vulnerabilities. As each device has simple logic to conduct some actions, it is easy to exploit different attacks once successfully hacking the MQTT broker. For instance, the Lock IoT device unlocks the door with the simple message “Open”. Hence, adversaries can unlock the door to enter the house once they understand that they can open the door with the command.

Table 2 lists of IoT device vulnerabilities and potential mitigations

Vulnerabilities Detected	Description	Device	Mitigations Deployed	Possible Mitigatinos
Open Door over publish without proper access control	Unlock door by sending a simple command "Open"	Lock	None	- Require username & password - Access control - Additional condition
Bruteforce Attack	Order Lock IoT client to open the door by cracking the access code	Controller	None	- Require username & password - Additional condition (such as applying check in time)
Economic/Storage DoS Attack	Take a picture by sending a simple command "Capture". If DoS attack conducted, many image files being taken, which leads to economic DoS if using Cloud or reaching storage upper limit in a short time.	Camera	None	- Require username & password - Overwrite an image - Max number of images - Lower resolutions - Restrict frequency of taking pictures per second

The second vulnerability is the brute-force attack vulnerability on the controller. The current spec requires a single key, “589SD463215SW657asd3321ad123ADGA”, to let the IoT Controller order the Lock IoT device to unlock the door. Hence, a brute-force attack against the subscribed topic may result in getting the ID in the end.

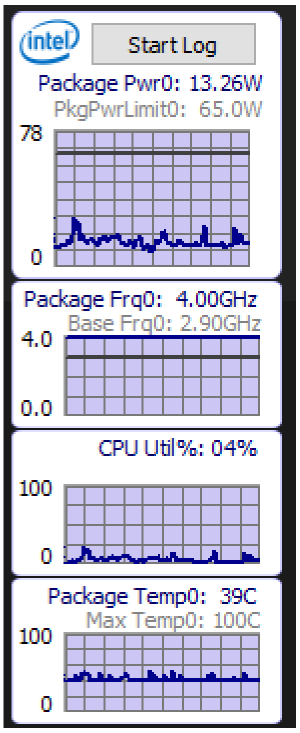
Another vulnerability identified was the Economic DoS attack (EDoS) on the IoT camera. The IoT camera would take a picture and save it as an image file with a simple command “Capture”. As adversaries can let the device take many pictures, this DoS attack becomes an EDoS if the device sends an image to clouds, or reaches the storage upper limit in a short time.

The countermeasures against these vulnerabilities can be requiring credentials, additional conditions such as time match or the frequency within a specific time range, or dynamic message changes.

7. Appendix

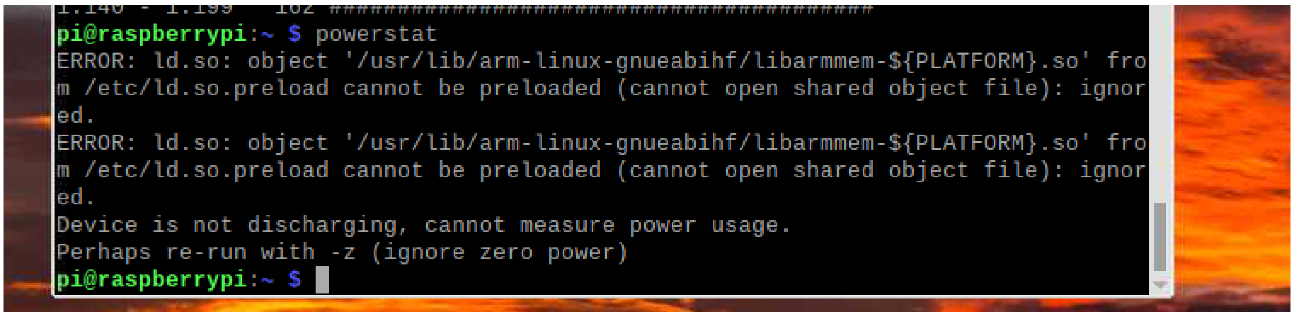
7.1 Power Stat testing

We have tried to test power consumption using Intel Core and Raspberry Pi chips. However, it was difficult to show the outputs because 1) showing the overall consumption and no advanced setting, 2) Raspberry Pi does not detect any power consumption using `powerstat` command.



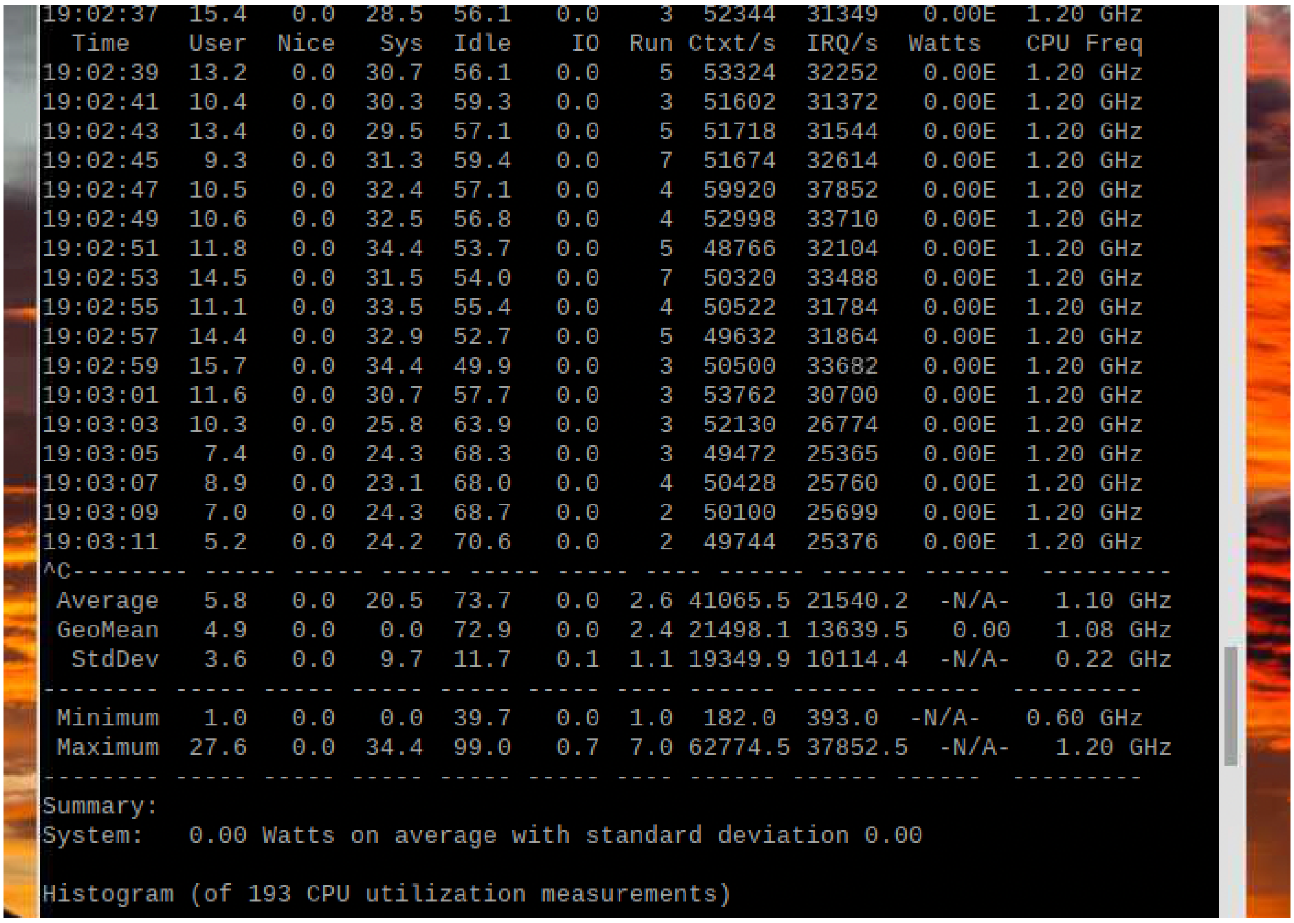
Package Pwr0: 13.26W
PkgPwrLimit0: 65.0W
Package Frq0: 4.00GHz
Base Frq0: 2.90GHz
CPU Util%: 04%
Package Temp0: 39C
Max Temp0: 100C

Figure 5 Intel Power Monitoring



```
1.140 - 1.199   102 ##############################################
pi@raspberrypi:~ $ powerstat
ERROR: ld.so: object '/usr/lib/arm-linux-gnueabihf/libarmmem-${PLATFORM}.so' fro
m /etc/ld.so.preload cannot be preloaded (cannot open shared object file): ignor
ed.
ERROR: ld.so: object '/usr/lib/arm-linux-gnueabihf/libarmmem-${PLATFORM}.so' fro
m /etc/ld.so.preload cannot be preloaded (cannot open shared object file): ignor
ed.
Device is not discharging, cannot measure power usage.
Perhaps re-run with -z (ignore zero power)
pi@raspberrypi:~ $
```

Figure 6 Failure of detecting power usage on Raspberry Pi



```
19:02:37  15.4   0.0  28.5  56.1   0.0    3  52344  31349   0.00E  1.20 GHz
  Time    User  Nice   Sys  Idle    IO  Run Ctxt/s  IRQ/s  Watts   CPU Freq
19:02:39  13.2   0.0  30.7  56.1   0.0    5  53324  32252   0.00E  1.20 GHz
19:02:41  10.4   0.0  30.3  59.3   0.0    3  51602  31372   0.00E  1.20 GHz
19:02:43  13.4   0.0  29.5  57.1   0.0    5  51718  31544   0.00E  1.20 GHz
19:02:45   9.3   0.0  31.3  59.4   0.0    7  51674  32614   0.00E  1.20 GHz
19:02:47  10.5   0.0  32.4  57.1   0.0    4  59920  37852   0.00E  1.20 GHz
19:02:49  10.6   0.0  32.5  56.8   0.0    4  52998  33710   0.00E  1.20 GHz
19:02:51  11.8   0.0  34.4  53.7   0.0    5  48766  32104   0.00E  1.20 GHz
19:02:53  14.5   0.0  31.5  54.0   0.0    7  50320  33488   0.00E  1.20 GHz
19:02:55  11.1   0.0  33.5  55.4   0.0    4  50522  31784   0.00E  1.20 GHz
19:02:57  14.4   0.0  32.9  52.7   0.0    5  49632  31864   0.00E  1.20 GHz
19:02:59  15.7   0.0  34.4  49.9   0.0    3  50500  33682   0.00E  1.20 GHz
19:03:01  11.6   0.0  30.7  57.7   0.0    3  53762  30700   0.00E  1.20 GHz
19:03:03  10.3   0.0  25.8  63.9   0.0    3  52130  26774   0.00E  1.20 GHz
19:03:05   7.4   0.0  24.3  68.3   0.0    3  49472  25365   0.00E  1.20 GHz
19:03:07   8.9   0.0  23.1  68.0   0.0    4  50428  25760   0.00E  1.20 GHz
19:03:09   7.0   0.0  24.3  68.7   0.0    2  50100  25699   0.00E  1.20 GHz
19:03:11   5.2   0.0  24.2  70.6   0.0    2  49744  25376   0.00E  1.20 GHz
^C-------- ----- ----- ----- ----- ----- ---- ------ ------ ------  ---------
 Average   5.8   0.0  20.5  73.7   0.0  2.6 41065.5 21540.2  -N/A-    1.10 GHz
 GeoMean   4.9   0.0   0.0  72.9   0.0  2.4 21498.1 13639.5   0.00    1.08 GHz
  StdDev   3.6   0.0   9.7  11.7   0.1  1.1 19349.9 10114.4  -N/A-    0.22 GHz
-------- ----- ----- ----- ----- ----- ---- ------ ------ ------  ---------
 Minimum   1.0   0.0   0.0  39.7   0.0  1.0  182.0  393.0  -N/A-   0.60 GHz
 Maximum  27.6   0.0  34.4  99.0   0.7  7.0 62774.5 37852.5  -N/A-    1.20 GHz
-------- ----- ----- ----- ----- ----- ---- ------ ------ ------  ---------
Summary:
System:   0.00 Watts on average with standard deviation 0.00

Histogram (of 193 CPU utilization measurements)
```

Figure 7 Failure of detecting power usage on Raspberry Pi 2

8. References

1. Saidi, S. J., Mandalari, A. M., Kolcun, R., Haddadi, H., Dubois, D. J., Choffnes, D., Smaragdakis, G. & Feldmann, A. (2020) 'A haystack full of needles: Scalable detection of IoT devices in the wild', *Proceedings of the ACM Internet Measurement Conference*. Virtual Event, the United States, 27-29 October. New York: Association for Computing Machinery. 87-100.
2. SS64 (2022) ulimit. Available from: https://ss64.com/bash/ulimit.html [Accessed June 12, 2022].