IoT Security and Privacy

Assignment 6 – MQTT

10 points

Team Members: Rajib Dey, Debashri Roy, Cody Baker, Kennedy Vrutaal

Assignment:

Each team is required to set up a mosquitto MQTT system. The system has three players: Client 1, Client 2, and MQTT broker (server). Client 1 must be on a Raspberry Pi. Client 2 and MQTT broker can be on the same Pi, or on different computers. Python is the recommended programming language although students are free to use C/C++/Java.

In the mosquitto MQTT system, Client 1 subscribes to the broker and Client 2 publishes to the broker. Client 1 should be able to receive messages published by Client 2. The Raspberry Pi can be installed with the Python client package paho-mqtt.

Students will also set up the TLS/SSL transport security for the MQTT system and use certificate based authentication for authenticating the clients by the broker.

NOTE 1: Instructions in the provided citations are only for reference. They may not work. It is the students' responsibility to correctly set up the system and meet the requirements below.

NOTE 2: Students can run the following command and get an example of bash script creating private keys, certificates and others.

wget https://github.com/owntracks/tools/raw/master/TLS/generate-CA.sh

Students **CANNOT** use private keys, certificates originally generated by generate-CA.sh. Students must use individual openssl commands to create those keys and certificates. Please provide the *openssl* commands in the report when asked. Students can read generate-CA.sh, dig out the openssl commands and use them. Students just cannot use generate-CA.sh directly although students can try this command and see what correct keys and certificate look like.

NOTE 3: openssl can view the content of a certificate. For example, the following command will display the content of the certificate file ca.crt.

```
openssl x509 -noout -text -in CA.crt
```

Requirements:

1. Set up the mosquitto MQTT system. Test the system works with either programs or *mosquitto_sub* and *mosquitto_pub* from *mosquitto*. Document the setup procedure and test results, including all the commands. (4 points)

Answer:

We were able to set up the mosquito MQTT system [1], by entering the following commands in the terminal:

```
sudo apt-get install software-properties-common
sudo apt-get install git
sudo apt-get install mosquitto
sudo apt-get install libmosquitto-dev
sudo apt-get install mosquitto-clients
```

We ensured that the Mosquitto broker is running by entering the following command in one terminal:

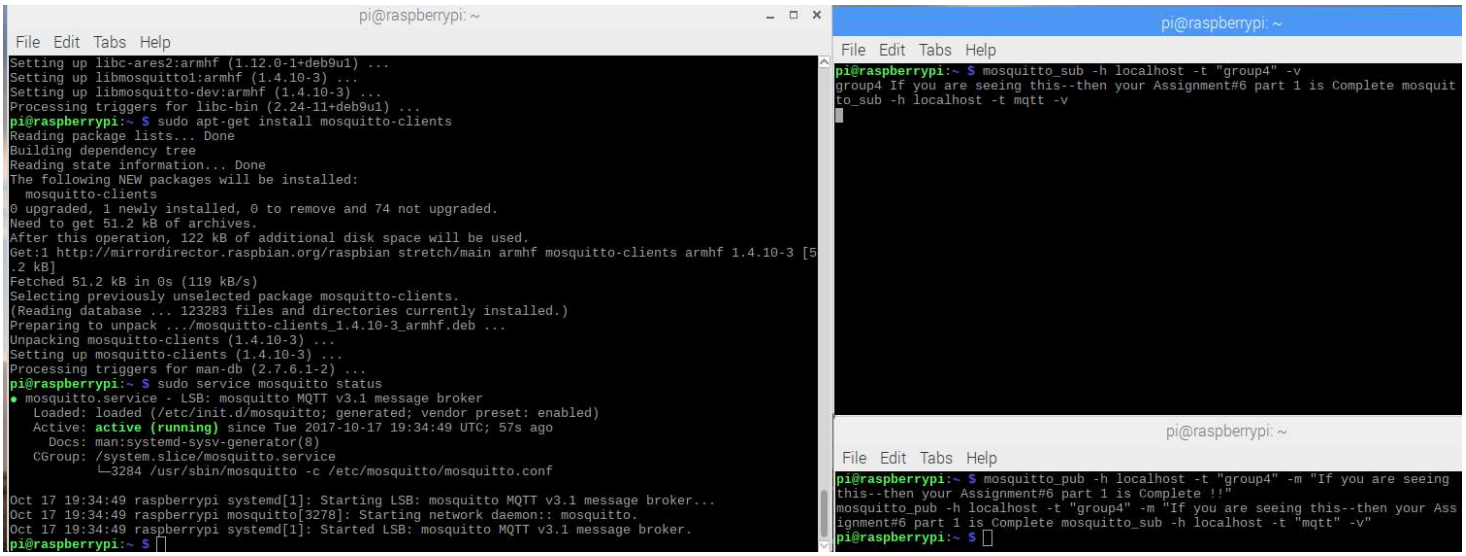
```
sudo service mosquitto status
```

We tested the to make sure that system works by entering the following command in a second terminal (to subscribe the topic "group4"):

```
mosquitto_sub -h localhost -t "group4" -v
```

We opened a third terminal and entered the following command to publish message to the topic "group4"

```
mosquitto_pub -h localhost -t "group4" -m "If you are seeing this--then your Assignment part 1 is Complete !!"
```



Screenshot 1 : Setup Mosquitto MQTT system and Test

2. Set up the mosquitto broker with SSL/TLS transport security. Please refer to Test the setup. Document the setup procedure and test results, including all the commands. (3 points)

Answer:

We were able to set up the mosquitto broker with SSL/TLS transport security [2], by running the following openssl commands in the terminal:

Generating Certificate for Certificate Authority (CA) [3]:

We created a private root key for CA by running:

```
openssl genrsa -out ca.key 2048
```

We self-signed the CA certificate by running:

```
openssl req -x509 -new -nodes -key ca.key -sha256 -days 1024 -out ca.crt
```

We entered some basic information about ourselves, which will be incorporated into your certificate request. That information can be found in the screenshot.

Generating Certificates for Server [3]:

We created a private root key for server by running:

```
openssl genrsa -out raspberrypi.key 2048
```

Once the key is created, we generated the certificate-signing request.

```
openssl req -new -key raspberrypi.key -out raspberrypi.csr
```

We were again asked various questions (Country, State/Province, etc.). We entered the same information as before. The important question to answer though is **common-name**. We gave it as "**localhost**". Point to be noted that, any other name than "localhost" will give errors to this process.

Next, we signed the CSR, which requires the CA root key, by running the following openssl command:

```
openssl x509 -req -in raspberrypi.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out raspberrypi.crt -days 500 -sha256
```

The screenshot of CA and server certificate generation is given bellow.

```
pi@raspberrypi: ~
File  Edit  Tabs  Help
pi@raspberrypi:~ $ ls
6         Desktop    Downloads  myCA      Public        Templates
Assign 6  Documents  Music      Pictures  python_games  Videos
pi@raspberrypi:~ $ openssl genrsa -out ca.key 2048
Generating RSA private key, 2048 bit long modulus
.....+++
...................+++
e is 65537 (0x010001)
pi@raspberrypi:~ $ openssl req -x509 -new -nodes -key ca.key -sha256 -days 1024 -out ca.crt
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:FL
Locality Name (eg, city) []:Orlando
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UCF
Organizational Unit Name (eg, section) []:EECS
Common Name (e.g. server FQDN or YOUR name) []:DR
Email Address []:dr@gmail.com
pi@raspberrypi:~ $ openssl genrsa -out raspberrypi.key 2048
Generating RSA private key, 2048 bit long modulus
......+++
.......................+++
e is 65537 (0x010001)
pi@raspberrypi:~ $ openssl req -new -key raspberrypi.key -out raspberrypi.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:FL
Locality Name (eg, city) []:Orlando
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UCF
Organizational Unit Name (eg, section) []:EECS
Common Name (e.g. server FQDN or YOUR name) []:localhost
Email Address []:dr2@gmail.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:6133
An optional company name []:
pi@raspberrypi:~ $ openssl x509 -req -in raspberrypi.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out raspberrypi.crt -days 500 -sha256
Signature ok
subject=C = US, ST = FL, L = Orlando, O = UCF, OU = EECS, CN = localhost, emailAddress = dr2@gmail.com
Getting CA Private Key
pi@raspberrypi:~ $ ls
6         ca.crt  ca.srl   Documents  Music  Pictures  python_games     raspberrypi.csr  Templates
Assign 6  ca.key  Desktop  Downloads  myCA   Public    raspberrypi.crt  raspberrypi.key  Videos
pi@raspberrypi:~ $
```

Next, we copied the necessary certificate files in **/etc/mosquitto/certs** folder, by running the following commands.

```
sudo -s
mkdir -p /etc/mosquitto/certs
cp ca.crt /etc/mosquitto/certs
cp raspberrypi.* /etc/mosquitto/certs
```

Then we edited the mosquitto configuration file (mosquitto.conf) located at /etc/mosquitto/ to look like the following:

```
# Plain MQTT protocol

listener 1883

# End of plain MQTT configuration

# MQTT over TLS/SSL

listener 8883
cafile /etc/mosquitto/certs/ca.crt
certfile /etc/mosquitto/certs/raspberrypi.crt
keyfile /etc/mosquitto/certs/raspberrypi.key

# End of MQTT over TLS/SLL configuration


# Plain WebSockets configuration

listener 9001
protocol websockets

# End of plain Websockets configuration

# WebSockets over TLS/SSL

listener 9883
protocol websockets
cafile /etc/mosquitto/certs/ca.crt
certfile /etc/mosquitto/certs/raspberrypi.crt
keyfile /etc/mosquitto/certs/raspberrypi.key
```

We then rebooted the pi using the reboot command to start testing the system.

Then we restarted the server:
sudo service mosquitto restart

The following command tests the server status:
sudo service mosquitto status

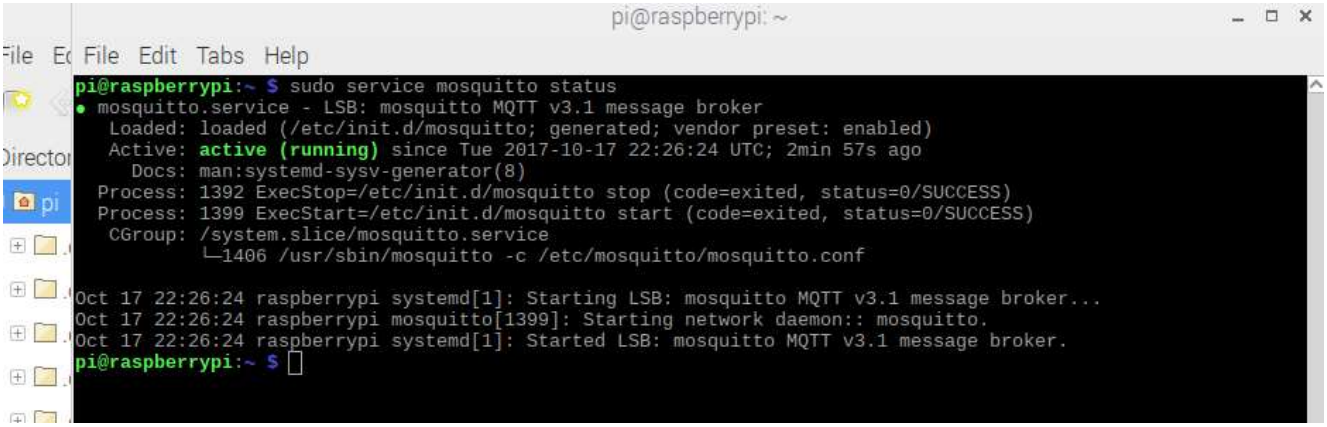
We subscribed to the port 8883 using the CA certificate using the following command:

mosquitto_sub -t \$SYS/broker/bytes/\# -v --cafile /etc/mosquitto/certs/ca.crt -p 8883

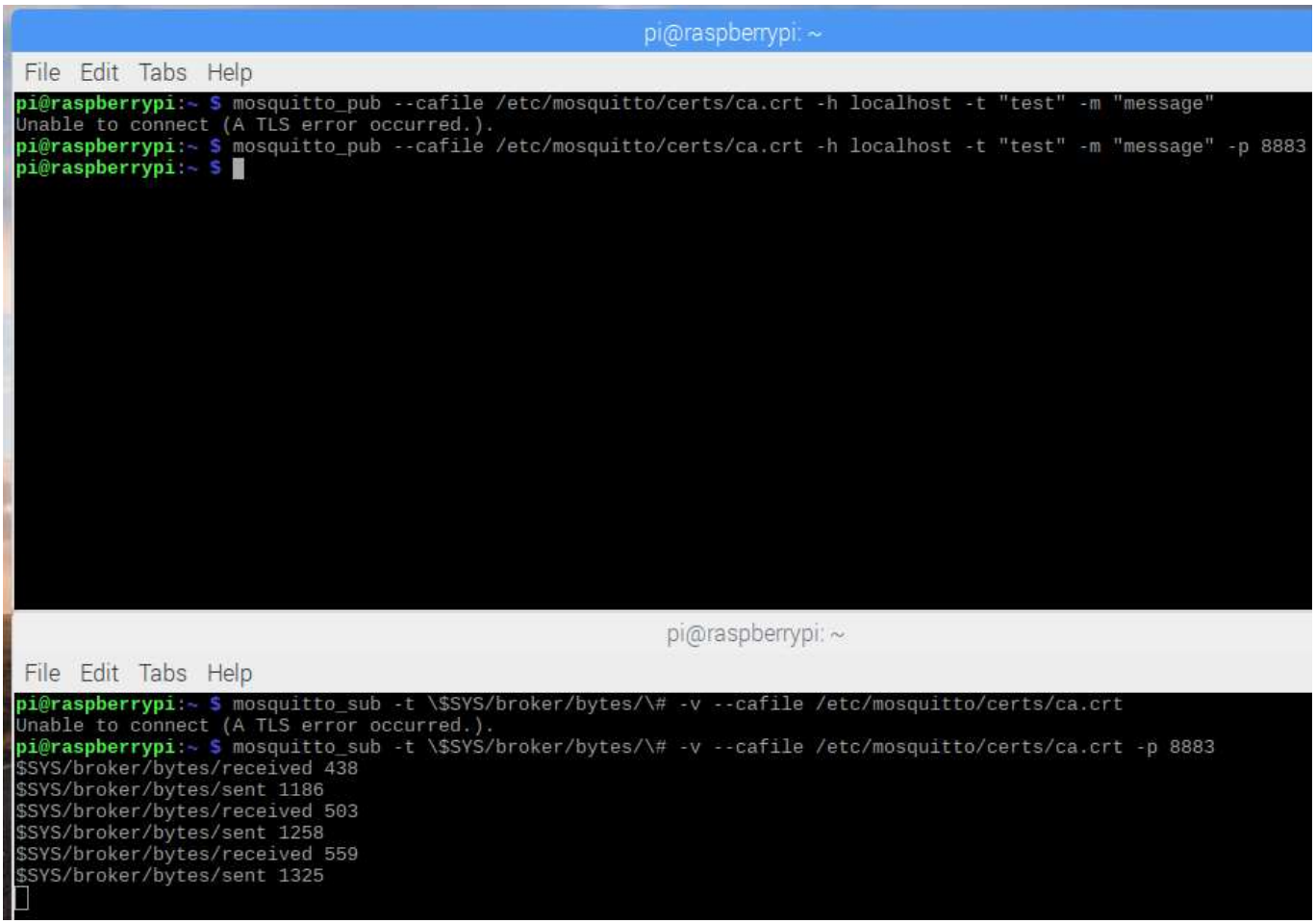
From another terminal, We published a message called “message” to the port 8883 using the following command:

```
mosquitto_pub --cafile /etc/mosquitto/certs/ca.crt -h localhost -t "test" -m "message" -p 8883
```

If we do not input the port number for the MQTT in the above command, then we will have a TLS error (Please refer to screenshot 2 and 3). Which confirms that the SSL/TLS transport security has been successfully implemented.



Screenshot 2 : mosquitto Broker Running Status



Screenshot 3 : Setup mosquitto broker with SSL/TLS transport security and Test

3. Set up the certificate based authentication between each client and the broker while using the mosquitto broker with SSL/TLS transport security. Test the setup. Document the setup procedure and test results, including all the commands. (3 points)

Answer:

We were able to set up the certificate-based authentication between each client and broker using mosquitto broker with SSL/TLS transport security, by following these steps [4]:

At first, we edited the mosquitto.conf file to make it look like this:

```
----------Generate server Certificate--------------------
# Plain MQTT protocol

listener 1883

# End of plain MQTT configuration
# MQTT over TLS/SSL

listener 8883
pid_file /var/run/mosquitto.pid
persistence true
persistence_location /var/lib/mosquitto/
log_dest file /var/log/mosquitto/mosquitto.log
cafile /etc/mosquitto/certs/ca.crt
certfile /etc/mosquitto/certs/raspberrypi.crt
keyfile /etc/mosquitto/certs/raspberrypi.key

# End of MQTT over TLS/SLL configuration
# Plain WebSockets configuration

listener 9001
protocol websockets

# End of plain Websockets configuration
# WebSockets over TLS/SSL

listener 9883
protocol websockets
cafile /etc/mosquitto/certs/ca.crt
certfile /etc/mosquitto/certs/raspberrypi.crt
keyfile /etc/mosquitto/certs/raspberrypi.key
```

Generating Certificates for Client 1:

We generated the client certificate by giving the following openssl commands on terminal:

```
openssl genrsa -out client.key 2048
openssl req -new -out client.csr -key client.key -subj "/CN=client/O=example.com"

openssl x509 -req -in client.csr -CA ca.crt -CAkey ca.key -CAserial ./ca.srl -out client.crt -days 3650 -addtrust clientAuth
```

We then changed the *#MQTT over TLS/SSL* section of the configuration file to make it look like this:

```
pid_file /var/run/mosquitto.pid
persistence true
persistence_location /var/lib/mosquitto/
log_dest file /var/log/mosquitto/mosquitto.log
cafile /etc/mosquitto/ca_certificates/ca.crt
certfile /etc/mosquitto/certs/raspberrypi.crt
keyfile /etc/mosquitto/certs/raspberrypi.key
require_certificate true
```

Please Notice, only the last line is added there.

Then we restarted the server:

```
sudo service mosquitto restart
```

The following command gave us an error, because client certificate was missing. Which was expected.

```
mosquitto_sub -t \$SYS/broker/bytes/\# -v --cafile /etc/mosquitto/certs/ca.crt -p 8883
```

Then the following command implements TLS and client certificate based authentication (Please refer to the Screenshot provided)

```
mosquitto_sub -t \$SYS/broker/bytes/\# -v --cafile /etc/mosquitto/certs/ca.crt --cert client.crt --key client.key -p 8883
```

Generating Certificates for Client 2:

To generate more client on terminal, we used the following commands:

For 2nd client, we generated the certificates:

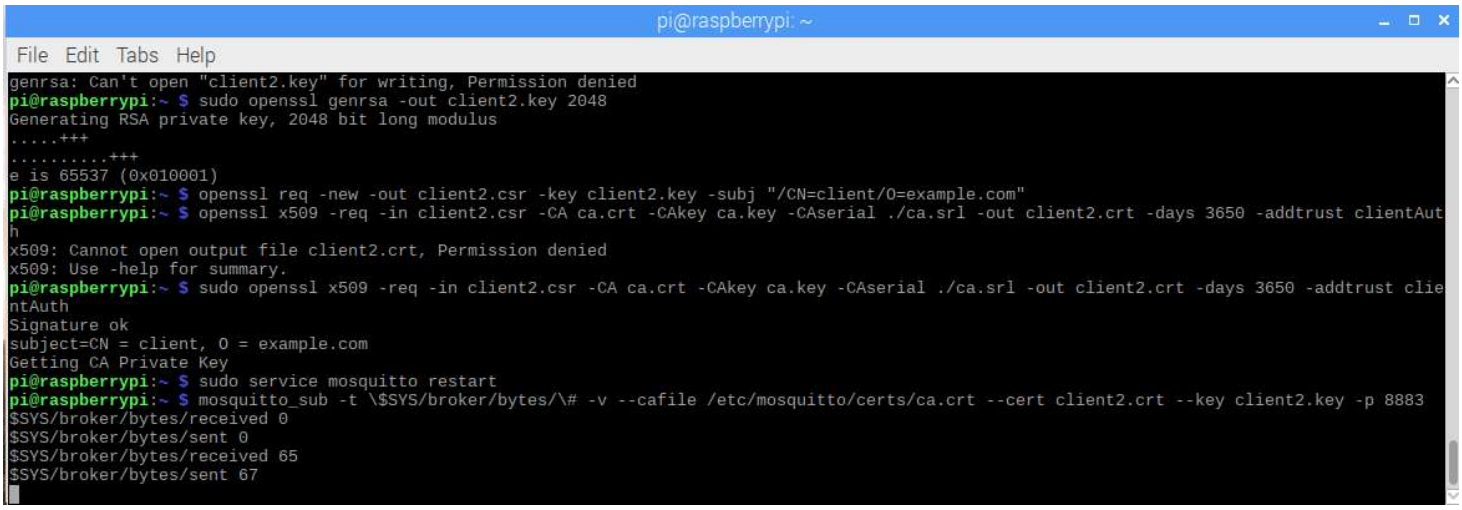
```
openssl genrsa -out client2.key 2048
openssl req -new -out client2.csr -key client2.key -subj "/CN=client/O=example.com"

openssl x509 -req -in client2.csr -CA ca.crt -CAkey ca.key -CAserial ./ca.srl -out client2.crt -days 3650 -addtrust clientAuth
```

Then again restart:

```
sudo service mosquitto restart
```

After generating the certificate for the 2nd client, mqtt worked with the following command:

```
mosquitto_sub -t \$SYS/broker/bytes/\# -v --cafile /etc/mosquitto/certs/ca.crt --cert client2.crt --key client2.key -p 8883
```



References

[1] http://wingsquare.com/blog/setting-up-mqtt-mosquitto-broker-in-ubuntu-linux/
[2] https://primalcortex.wordpress.com/2016/03/31/mqtt-mosquitto-broker-with-ssltls-transport-security/
[3] https://datacenteroverlords.com/2012/03/01/creating-your-own-ssl-certificate-authority/
[4] http://rockingdlabs.dunmire.org/exercises-experiments/ssl-client-certs-to-secure-mqtt