

MQTT – SSL Certification

Opis stosowania certyfikacji SSL przy użyciu protokołu MQTT

Autor	Bartłomiej Krasoń
Wersja	1
Data modyfikacji	08-01-2019

Spis treści

Wprowadzenie	3
Wymagania.....	3
Wymagane pliki	3
Wymagane oprogramowanie.....	3
Utworzenie certyfikatów	3
Testowanie certyfikatu	6
Wykonując nieprawidłowe polecenie	6
Wykonując prawidłowe polecenie	6
Modyfikacja programu klienta	6

Wprowadzenie

Niniejszy dokument prezentuje w jaki sposób zagwarantować zabezpieczony – kodowany przesył danych pomiędzy serwerem a klientem przy wykorzystaniu protokołu MQTT. W tym celu wykorzystamy certyfikację SSL. System ten opiera się na wyprodukowaniu plików, które będą wymagane przez serwer/klienta aby możliwa była komunikacja między nimi. Pliki takie nazywa się kluczami oraz certyfikatami.

Wymagania

Wymagane pliki

Zestaw potrzebnych plików:

- Brokera:
 - server.key – klucz certyfikatu brokera, potrzeby do utworzenia pliku server.csr
 - server.csr – plik żądania certyfikatu, potrzeby do utworzenia pliku server.crt
 - server.crt – właściwy plik certyfikujący brokera
- Klienta:
 - ca.key – klucz certyfikatu klienta, potrzebny do utworzenia pliku ca.crt
 - ca.crt – właściwy plik certyfikujący klientów

Wymagane oprogramowanie

Korzystamy z pakietu **openssl**, który można pozyskać w następujący sposób:

```
$ wget https://www.openssl.org/source/openssl-1.0.2q.tar.gz
$ tar -xzf openssl-1.0.2q.tar.gz
$ cd openssl-1.0.2q
$ ./config
$ make
$ sudo make install
$ /usr/local/ssl/bin/openssl version <- sprawdzenie czy instalacja zadziałała
```

Utworzenie certyfikatów

Wszystkie kroki muszą być wykonane w danej kolejności:

1. Utworzenie klucza CA klienta:
\$ openssl genrsa -des3 -out ca.key 2048
2. Utworzenie certyfikatu klienta przy wykorzystaniu klucza:
\$ **openssl req -new -x509 -days 1826 -key ca.key -out ca.crt**

```
C:\steve>openssl req -new -x509 -days 1826 -key ca.key -out ca.crt
Enter pass phrase for ca.key:
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]:UK
State or Province Name (full name) [Some-State]:Shropshire
Locality Name (eg, city) []:Ironbridge
Organization Name (eg, company) [Internet Widgits Pty Ltd]:CAmaster
Organizational Unit Name (eg, section) []:TEST
Common Name (e.g. server FQDN or YOUR name) []:ws4
Email Address []:steve@testemail.com
```

3. Utworzenie klucza CA brokera

```
$ openssl genrsa -out server.key 2048
```

4. Utworzenie żądanie certyfikatu brokera:

```
$ openssl req -new -out server.csr -key server.key
```

WAŻNE Common Name ma być ustawione na adres IP brokera (192.168.1.200) oraz dane muszą się choć trochę różnić od ca.crt (punkt 2)

```
C:\steve>openssl req -new -out server.csr -key server.key
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]:UK
State or Province Name (full name) [Some-State]:Shropshire
Locality Name (eg, city) []:Ironbridge
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Server-cert
Organizational Unit Name (eg, section) []:test
Common Name (e.g. server FQDN or YOUR name) []:192.168.1.200
Email Address []:steve@testemail.com
```

5. Utworzenie certyfikatu brokera przy użyciu klucza klientów:

```
$ openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt -
days 360
```

6. Upewnienie się że mamy wszystkie potrzebne pliki

```
C:\steve>dir
Volume in drive C has no label.
Volume Serial Number is E091-351C

Directory of C:\steve

06/11/2016  17:40    <DIR>          .
06/11/2016  17:40    <DIR>          ..
06/11/2016  17:35             1,419 ca.crt
06/11/2016  17:28             1,743 ca.key
06/11/2016  17:40                17 ca.srl
06/11/2016  17:40             1,359 server.crt
06/11/2016  17:39             1,143 server.csr
06/11/2016  17:37             1,675 server.key
               6 File(s)              7,356 bytes
               2 Dir(s) 19,440,615,424 bytes free
```

Need these 3 files

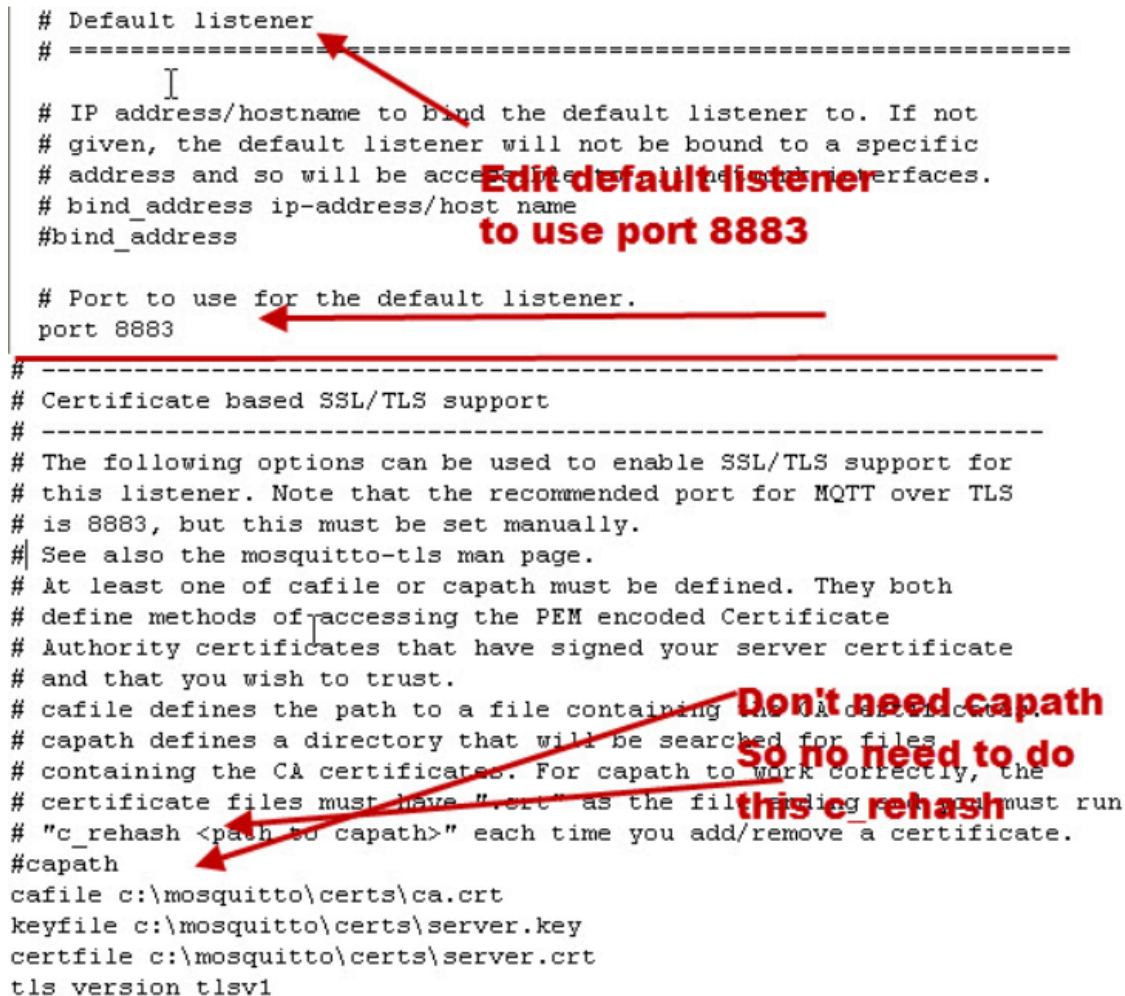


7. Przeniesienie potrzebnych plików:
\$ mkdir /etc/mosquito/certs
\$ cp ca.crt /etc/mosquito/certs/ca.crt
\$ cp server.crt /etc/mosquito/certs/server.crt
\$ cp server.key /etc/mosquito/certs/server.key
8. Edytowanie pliku /etc/mosquito/mosquitto.conf

```
# Default listener
# =====
# IP address/hostname to bind the default listener to. If not
# given, the default listener will not be bound to a specific
# address and so will be accessible on all network interfaces.
# bind_address ip-address/host name
#bind_address

# Port to use for the default listener.
port 8883

# -----
# Certificate based SSL/TLS support
# -----
# The following options can be used to enable SSL/TLS support for
# this listener. Note that the recommended port for MQTT over TLS
# is 8883, but this must be set manually.
# See also the mosquitto-tls man page.
# At least one of cafile or capath must be defined. They both
# define methods of accessing the PEM encoded Certificate
# Authority certificates that have signed your server certificate
# and that you wish to trust.
# cafile defines the path to a file containing the CA certificate.
# capath defines a directory that will be searched for files
# containing the CA certificates. For capath to work correctly, the
# certificate files must have ".crt" as the file ending and you must run
# "c_rehash <path to capath>" each time you add/remove a certificate.
#capath
cafile c:\mosquitto\certs\ca.crt
keyfile c:\mosquitto\certs\server.key
certfile c:\mosquitto\certs\server.crt
tls_version tlsv1
```



Basic TLS Support on Mosquitto Broker

9. Od teraz broker po uruchomieniu będzie działał na porcie **8883** który jest zalecany do certyfikacji SSL.

Testowanie certyfikatu

Należy skopiować plik **ca.crt** do dowolnego klienta. Następnie wykorzystując przykładowego klienta mosquitto można sprawdzić czy certyfikacja działa prawidłowo.

Wykonując nieprawidłowe polecenie

```
$ mosquitto_pub -h 192.168.1.200 -t test -p 8883 -m "Certificated message failed"
```

Powinniśmy otrzymać wiadomość:

Error: The connection was lost.

```
sysop@sysop-VirtualBox:~$ mosquitto_pub -h 192.168.1.200 -t test -p 8883 -m "Certificated message failed"
Error: The connection was lost.
```

Wykonując prawidłowe polecenie

```
$ mosquitto_pub -h 192.168.1.200 -t test -p 8883 --cafile /home/sysop/Pulpit/ca.crt -m "Certificated message success"
```

Powinno nie wyrzucić żadnego komunikatu, a wiadomość trafić na podany temat.

```
sysop@sysop-VirtualBox:~$ mosquitto_pub -h 192.168.1.200 -t test -p 8883 --cafile /home/sysop/Pulpit/ca.crt -m "Certificated message success"
sysop@sysop-VirtualBox:~$
```

Modyfikacja programu klienta

Zmienione elementy zaznaczone kolorem żółtym.

UWAGA! certyfikacja SSL/TLS wymaga wersji Pythona nowszej od wersji 2.7.9 lub 3.2.0.

Należy skopiować plik **ca.crt** do klienta i następnie wskazać do niego ścieżkę w programie.

```
#!/home/root/python279/Python-2.7.9/python
import paho.mqtt.client as mqtt

# The callback for when the client receives a CONNACK response from the
server.
def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    print("Hello, you successfully connected to your local server!")

    # Subscribing in on_connect() means that if we lose the connection
and
    # reconnect then subscriptions will be renewed.
    client.subscribe("test/#")

# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
    print("Writing data... => " + msg.topic + " " + str(msg.payload))
    file = open("data.txt", "a")
    file.write("Topic: " + msg.topic + ", Message: " + msg.payload +
"\n")
    file.close()
```

```
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
client.tls_set("ca.crt")
client.connect("192.168.1.200", 8883, 60)

# Blocking call that processes network traffic, dispatches callbacks
and
# handles reconnecting.
# Other loop*() functions are available that give a threaded interface
and a
# manual interface.
client.loop_forever()
```

Teraz w programie używamy Pythona w wersji 2.7.9 który jest zainstalowany w lokalizacji **#!/home/root/python279/Python-2.7.9/python**, ma on też swojego pip'a i inne paczki w **/usr/local/lib/python2.7.9/bin**.

Bibliografia

- [1] <http://www.steves-internet-guide.com/mosquitto-tls/>
- [2] <https://mosquitto.org/man/mosquitto-tls-7.html>