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](#_Toc534715595)

[Wymagania 3](#_Toc534715596)

[Wymagane pliki 3](#_Toc534715597)

[Wymagane oprogramowanie 3](#_Toc534715598)

[Utworzenie certyfikatów 3](#_Toc534715599)

[Testowanie certyfikatu 6](#_Toc534715600)

[Wykonując nieprawidłowe polecenie 6](#_Toc534715601)

[Wykonując prawidłowe polecenie 6](#_Toc534715602)

[Modyfikacja programu klienta 6](#_Toc534715603)

# 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 będę wymagane przez serwer/klienta aby możliwa była komunikacja miedzy 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 –xzvf 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

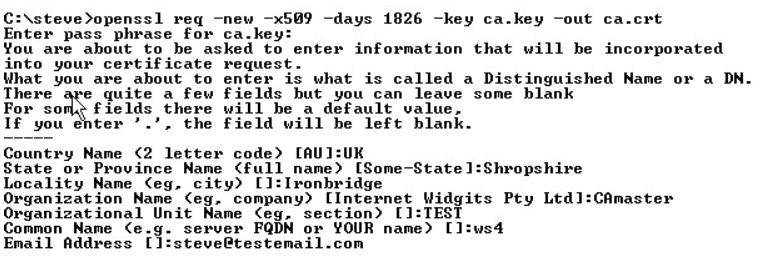
Wszystkie kroki muszą być wykonane w danej kolejności:

1. Utworzenie klucza CA klienta:

$ openssl genrsa –des3 –out ca.key 2048

1. Utworzenie certyfikatu klienta przy wykorzystaniu klucza:

$ **openssl req -new -x509 -days 1826 -key ca.key -out ca.crt**



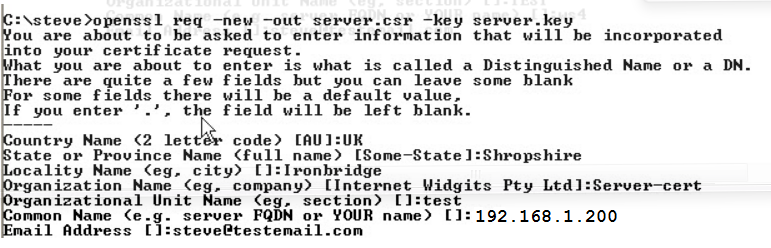
1. Utworzenie klucza CA brokera

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

1. Utworzenie żądanie certyfikatu brokera:

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

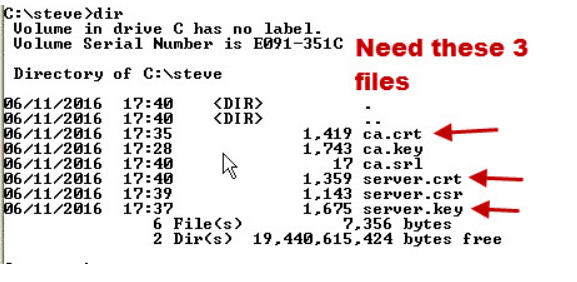
**WAŻNE** Common Name ma być ustawione na adres IP brokera (192.168.1.200) oraz dane muszą się choć trochę różnić od ca.crt (punkt 2)



1. 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**

1. Upewnienie się że mamy wszystkie potrzebne pliki



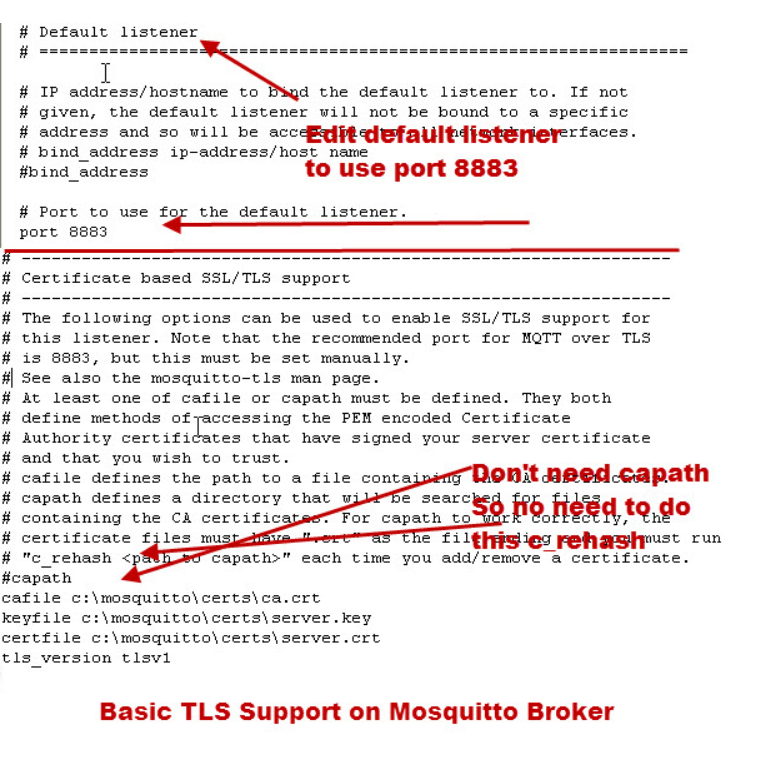
1. Przeniesienie potrzebnych plików:

$ mkdir /etc/mosquito/certs

$ cp ca.crt /etc/mosquito/certs/ca.crt

$ cp ca.crt /etc/mosquito/certs/server.crt

$ cp ca.crt /etc/mosquito/certs/server.key

1. Edytowanie pliku /etc/mosquito/mosquitto.conf
2. 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:

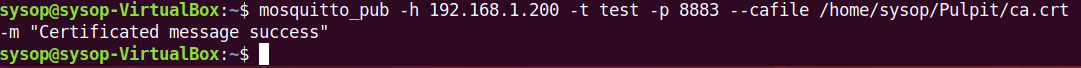
**Error: The connection was lost.**



### 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.



# 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/pyton**, ma on też swojego pipa i inne paczki w **/usr/local/lib/python2.7.9/bin**.