Comentario técnico: CTC-093
Componente: **MQTT sobre TLS con ESP32 y Mongoose-OS**
Autor: Sergio R. Caprile, Senior R&D Engineer

Revisiones	Fecha	Comentarios
0	04/05/20	

Mongoose-OS incorpora un cliente MQTT con soporte TLS. Esto nos permite por un lado autenticar al servidor, y por otro que la infraestructura del servidor pueda autenticar la identidad de esas conexiones, es decir, que realmente sean usuarios autorizados. Hemos visto el funcionamiento de TLS en el CTC-092, por lo que aquí nos centraremos en los detalles relativos a MQTT y más adelante en una característica diferente como es TLS-PSK, que permite realizar la misma tarea con un nivel más relajado en la autenticación y logística asociada.

Configuración

Configuramos el ESP32 con Mongoose-OS para tener un cliente MQTT y conectarse como cliente a una red WiFi. Dado que debemos conectarnos a un servidor, ésta nos resulta la forma tal vez más rápida y simple de realizar las pruebas y explicar la operación. La configuración puede realizarse manualmente mediante RPC, en un archivo de configuración JSON; o definirla en el archivo YAML que describe el proyecto. Para las pruebas elegimos esta última opción.

```
libs:
  - origin: https://github.com/mongoose-os-libs/mqtt       # incorpora el cliente MQTT
config_schema:
  - ["mqtt.enable", true]                   # Habilita el cliente MQTT
  - ["mqtt.server", "address:port"]         # Dirección IP del broker a utilizar
  - ["mqtt.ssl_ca_cert", "ca.crt"]          # Certificado del broker, requerido
  - ["mqtt.ssl_cert", "sandboxclient.crt"]  # nuestro certificado, para doble autenticación
  - ["mqtt.ssl_key", "sandboxclient.key"]   # nuestra clave, para doble autenticación
```

El puerto comúnmente utilizado es 8883. El servidor puede además solicitar que utilicemos nombre de usuario y password, los detalles de la configuración completa los podemos encontrar en la página de Mongoose-OS[1].
Finalmente, quien decide qué tipo de autenticación (simple o doble) utilizamos es el broker, aunque nosotros debemos proveer el certificado cuando es requerido.

Operación

Luego de compilado el código (`mos build`) y grabado el microcontrolador (`mos flash`) mediante *mos tool*, observaremos en el log si todo funciona como debe, o los errores que se hayan producido. Recordemos que debemos configurar las credenciales para conectarnos por WiFi a nuestra red (SSID y clave) y la dirección y port del broker MQTT que vayamos a utilizar. Al final de la nota comentamos cómo utilizar Mosquitto.
Por las mismas razones que allí, y también a los fines prácticos, reutilizamos los certificados provistos en el CTC-091 (y CTC-090), con algunos detalles. Algunos de los certificados que empleamos en dichos CTC están firmados con SHA-1 y no son aceptados por la configuración de *mbedtls* de Mongoose-OS. A tal fin, incorporamos una nueva versión del certificado de servidor para este CTC, la clave es la misma. Si por algún

1 https://mongoose-os.com/docs/mongoose-os/api/net/mqtt.md

motivo se nos mezclan y todo "mágicamente deja de funcionar", observaremos en el log la frase "`The certificate is signed with an unacceptable hash`".

Simple autenticación

De esta forma autenticamos al servidor, es decir, podemos confiar en que es el que esperamos que sea, pero el servidor no tiene idea de quién podemos llegar a ser nosotros.
Es la configuración más simple y no debemos tomar ningún recaudo más allá de informar el nombre del archivo que contiene el certificado de la CA almacenado en el directorio *fs* en el parámetro *ssl_ca_cert*, esto indica que se deberá validar al broker.

Doble autenticación

De esta forma autenticamos tanto al servidor como al cliente, es decir, ahora el servidor sabe que deberíamos ser quien nuestro certificado dice que somos. Este doble proceso requiere bastante procesamiento y lo notaremos en el tiempo de establecimiento de la conexión.
En esta configuración deberemos proveer nuestro certificado y clave en el directorio *fs*, además de configurar los parámetros *ssl_cert* y *ssl_key*; caso contrario el broker puede que nos dé una pista en el log, como lo hace Mosquitto:

```
Apr 29 19:00:52 Server mosquitto[2694]: OpenSSL Error: error:140890C7:SSL
routines:ssl3_get_client_certificate:peer did not return a certificate
```

TLS-PSK

Existe una solución un poco más simple que consiste en utilizar, en vez de certificados, una clave pre-compartida entre el broker y el dispositivo que se conecta.
El proceso de conexión es similar al empleado con certificados, sólo que en vez de validarse éstos y luego generarse una clave derivada, se utiliza como punto de partida la clave pre-compartida. Según el esquema utilizado, se evitan las costosas operaciones de clave pública/privada (Public Key Cryptography)[2], y la logística de manejo de claves suele ser más simple, además del hecho de no requerir una CA (Certification Authority).

Configuración

Configuramos el ESP32 con Mongoose-OS para operar mediante TLS-PSK.

```
libs:
  - origin: https://github.com/mongoose-os-libs/mqtt       # incorpora el cliente MQTT
config_schema:
  - ["mqtt.enable", true]                   # Habilita el cliente MQTT
  - ["mqtt.server", "address:port"]         # Dirección IP del broker a utilizar
  - ["mqtt.ssl_psk_identity", "bob"]        # identidad para la clave elegida
  - ["mqtt.ssl_psk_key", "000000000000000000000000deadbeef"]       # clave AES-128 (ó 256)
  - ["mqtt.ssl_cipher_suites", "TLS-PSK-WITH-AES-128-CCM-8:TLS-PSK-WITH-AES-128-GCM-SHA256:TLS-
ECDHE-PSK-WITH-AES-128-CBC-SHA256:TLS-PSK-WITH-AES-128-CBC-SHA"]   # claves utilizables
```

El puerto comúnmente utilizado es también 8883. El servidor puede además solicitar que utilicemos nombre de usuario y password, aunque lo más común es configurarlo para aprovechar la identidad que se envía, como hicimos para estas pruebas.

2 Existen al momento tres formas de operar, una de las cuales permite validar al broker mediante certificados y al dispositivo mediante clave pre-compartida (PSK). Sólo hemos probado la forma simple.

Respecto a las *cipher suites*, se trata de indicar un conjunto de esquemas criptográficos que el sistema soporta y entre ellos debemos acertar a elegir uno que esté soportado por el broker. Este parámetro es obligatorio, sin él el dispositivo no anuncia ningún esquema criptográfico compatible con TLS-PSK y la conexión TLS no se establece.[3] Lo habitual es acordarlo con el administrador del broker.[4]
Por último, la clave pre-compartida (*PSK*, parámetro *ssl_psk_key*) debe tener una longitud de 128-bits (16-bits) para sets basados en AES-128 y de 256-bits (32-bytes) para sets basados en AES-256.

Operación

Al iniciar el procesador, observaremos en el log si todo funciona como debe, o los errores que se hayan producido. En este último caso, resulta bastante difícil determinar las causas sin un sniffer dado que las pistas de ambos lados no suelen ser muy precisas. Tanto el handshake como la operación MQTT dentro de TLS pueden observarse en un sniffer, ingresando dicha clave en el mismo (por ejemplo Wireshark[5]).

Brokers

Para las pruebas hemos usado Mosquitto, el cual configuramos manualmente para cada esquema. Entre algunas de las opciones disponibles en la nube que mencionamos en el CTC-092, comprobamos que tanto Eclipse como CloudMQTT soportan autenticación simple (del broker) por TLS[6].

Mosquitto

Detallamos, a modo instructivo y como ayuda rápida, las configuraciones mínimas necesarias para operar en Mosquitto. Los paths están en formato GNU/Linux y se espera que pongamos nuestros certificados allí.

Simple autenticación (Autenticación del broker)

```
listener 8883 192.168.5.1
log_dest syslog
cafile /etc/mosquitto/certs/ca.crt
certfile /etc/mosquitto/certs/server.crt
keyfile /etc/mosquitto/certs/server.key
```

Doble autenticación (Autenticación de broker y clientes)

```
listener 8883 192.168.5.1
log_dest syslog
cafile /etc/mosquitto/certs/ca.crt
certfile /etc/mosquitto/certs/server.crt
keyfile /etc/mosquitto/certs/server.key
require_certificate true
```

TLS-PSK

```
listener 8883 192.168.5.1
log_dest syslog
```

3 Al iniciar la conexión TLS, el dispositivo incluye los esquemas soportados en el mensaje *ClientHello*; el broker elige uno compatible y lo indica en el mensaje *ServerHello*.
4 Dado que esta lista la obtuvimos analizando el código fuente, decidimos publicarla aquí para mayor utilidad. En nuestro caso en particular sólo uno de los esquemas coincidió con los disponibles en el broker: TLS-PSK-WITH-AES-128-CCM-8
5 En Wireshark es tan simple como ingresar la clave en *Edit->Preferences->Protocols->TLS->Pre-Shared Key*
6 Algunos proveedores utilizan el nombre SSL, recordemos que hablamos de lo mismo y que TLS deriva de SSL.

```
use_identity_as_username true          # evitamos que requiera usuario en el header MQTT
psk_file /etc/mosquitto/pskfile
psk_hint cualquiera                    # cualquier nombre[7]
```

El archivo *pskfile*, a su vez, debe contener las identidades y claves pre-compartidas:

```
bob:00000000000000000000000deadbeef
```

7 En el caso que el cliente se conecte a varios lugares, esto le da una pista (hint) de qué identidad utilizar para identificarse. No tiene utilidad en nuestro caso.