

Grupo 5
Carlos Pérez Cano
Miquel Àngel Román Colom

PRÁCTICA

11769 – Conectivitat i Integració de Sistemes a IoT

Contenido

Preparación del entorno.....	2
Ejercicio 1.....	4
Ejercicio 2.....	8
Ejercicio 3.....	11
Ejercicio 4.....	15

Preparación del entorno

Todo el material generado y necesario utilizado en este documento se puede encontrar [aquí](#).

Para poder reproducir completamente este ejercicio necesitaremos instalar los paquetes siguientes de Ubuntu:

- **Mosquitto:** Un servidor que implementa el protocolo MQTT. Además, viene con un conjunto de herramientas útiles para su gestión.

```
sudo apt-get update  
sudo apt-get install mosquitto
```

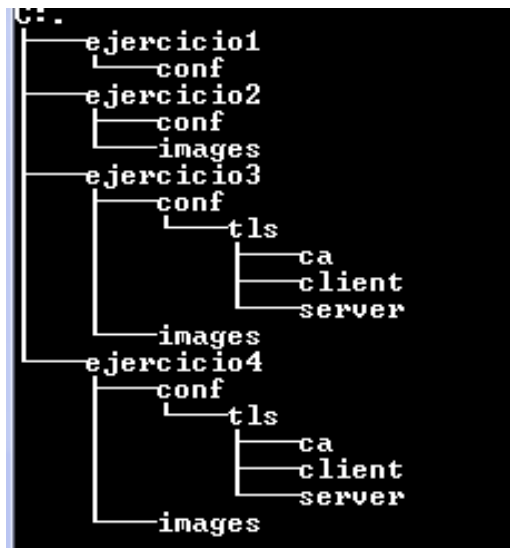
- **Mosquitto clients:** Conjunto de clientes para poder hacer operaciones como publicar y suscribirse en tópicos.

```
sudo apt-get install mosquitto-clients
```

- **Tcpflow:** Una utilidad de la línea de comandos que nos permite leer el tráfico entrante y saliente de nuestra máquina. Útil para verificar si mosquitto y los clientes cifran los datos antes de enviarlos.

```
sudo apt-get install tcpflow
```

En el enlace proporcionado al principio de este apartado se puede observar una estructura de directorios del siguiente tipo:



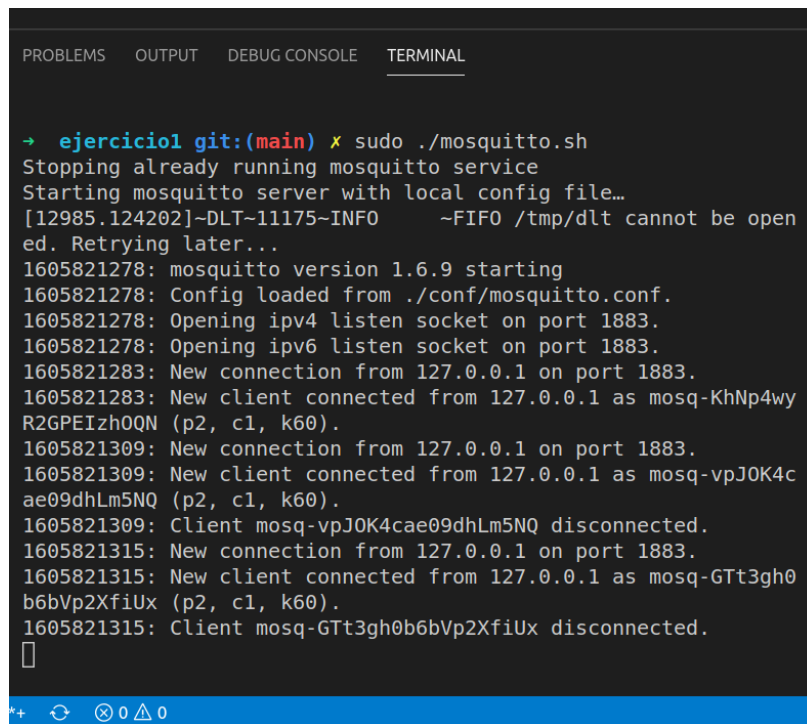
En cada uno de los directorios existe un script **mosquitto.sh** cuya funcionalidad es simplificar el acceso a los ficheros locales de cada subdirectorio de configuración. Es decir, en el directorio ejercicio1 contiene su propio fichero **mosquitto.conf** dentro del directorio **conf** con los cambios necesarios para el ejercicio1 y el script **mosquitto.sh** se encarga de lanzar el servidor de mosquitto usando la configuración correspondiente al ejercicio1 y análogamente para el resto de ejercicios.

A parte de estos “wrappers” se pueden encontrar otros scripts que simplifican la simulación de las pruebas para verificar que solucionamos los ejercicios de manera satisfactoria.

Ejercicio 1

Ejercicio (1 punto): Configurar el broker para que pueda controlar a los usuarios no anónimos. Crear varios usuarios con su contraseña y realizar diferentes pruebas verificando que el broker permite o restringe su acceso.

Así sin más, y con un fichero de configuración de mosquitto vacío podemos ejecutar el servidor como ilustran las Figuras 1, 2 y 3 para enviar y recibir mensajes sin ningún tipo de restricción.

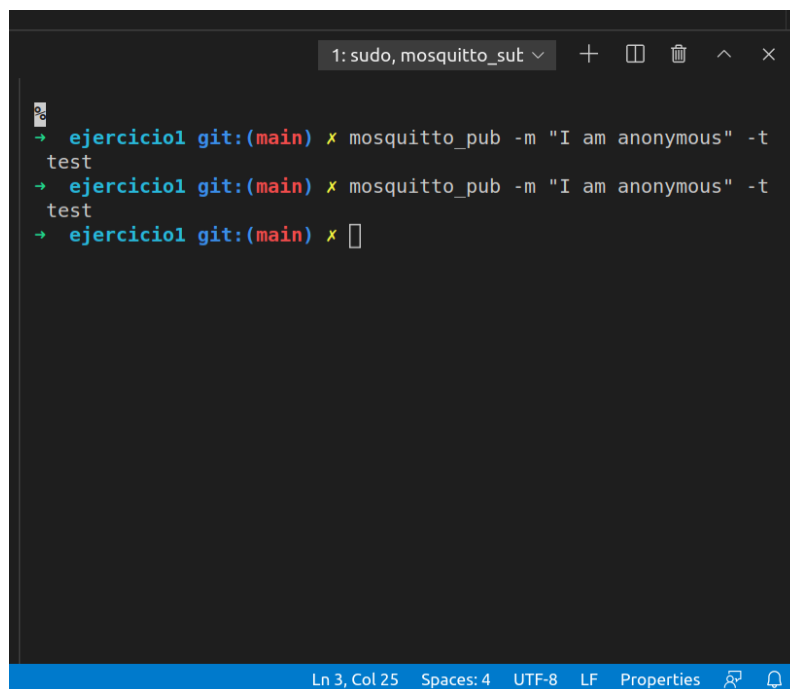


```
→ ejercicio1 git:(main) x sudo ./mosquitto.sh
Stopping already running mosquitto service
Starting mosquitto server with local config file...
[12985.124202]~DLT~11175~INFO      ~FIFO /tmp/dlt cannot be open
ed. Retrying later...
1605821278: mosquitto version 1.6.9 starting
1605821278: Config loaded from ./conf/mosquitto.conf.
1605821278: Opening ipv4 listen socket on port 1883.
1605821278: Opening ipv6 listen socket on port 1883.
1605821283: New connection from 127.0.0.1 on port 1883.
1605821283: New client connected from 127.0.0.1 as mosq-KhNp4wy
R2GPEIzh0QN (p2, c1, k60).
1605821309: New connection from 127.0.0.1 on port 1883.
1605821309: New client connected from 127.0.0.1 as mosq-vpJ0K4c
ae09dhLm5NQ (p2, c1, k60).
1605821309: Client mosq-vpJ0K4cae09dhLm5NQ disconnected.
1605821315: New connection from 127.0.0.1 on port 1883.
1605821315: New client connected from 127.0.0.1 as mosq-GTt3gh0
b6bVp2XfiUx (p2, c1, k60).
1605821315: Client mosq-GTt3gh0b6bVp2XfiUx disconnected.
█
```

Figura 1: Output del mosquitto.sh del ejercicio 1

```
→ ejercicio1 git:(main) x mosquitto_sub -t test
Connection error: Connection Refused: not authorised.
→ ejercicio1 git:(main) x mosquitto_sub -t test
I am anonymous
I am anonymous
█
```

Figura 2 Suscriptor suscribiéndose al tópico "test" y recibiendo mensajes

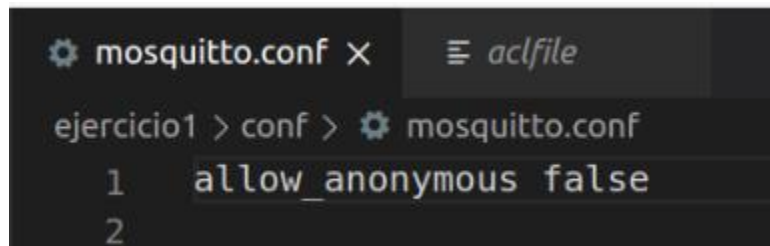


The screenshot shows a terminal window with a title bar that includes a tab labeled "1: sudo, mosquitto_sut". The terminal content shows a user named "ejercicio1" in a "git:(main)" environment. They execute the command "mosquitto_pub -m 'I am anonymous' -t test" twice. The first execution is followed by a new line, and the second is followed by a cursor. The status bar at the bottom indicates "Ln 3, Col 25", "Spaces: 4", "UTF-8", "LF", and "Properties".

```
→ ejercicio1 git:(main) x mosquitto_pub -m "I am anonymous" -t
test
→ ejercicio1 git:(main) x mosquitto_pub -m "I am anonymous" -t
test
→ ejercicio1 git:(main) x █
```

Figura 3 Publicador enviando mensajes en el tópico "test"

A continuación, vamos a restringir el acceso de los usuarios no autenticados. Para ello vamos a poner la opción **allow_anonymous** a false de tal manera que rechazamos todas aquellas conexiones sin credenciales de usuario.



```
mosquitto.conf x aclfile
ejercicio1 > conf > mosquitto.conf
1 allow_anonymous false
2
```

Figura 4 Línea del fichero mosquitto.conf que restringe el acceso a usuarios anónimos

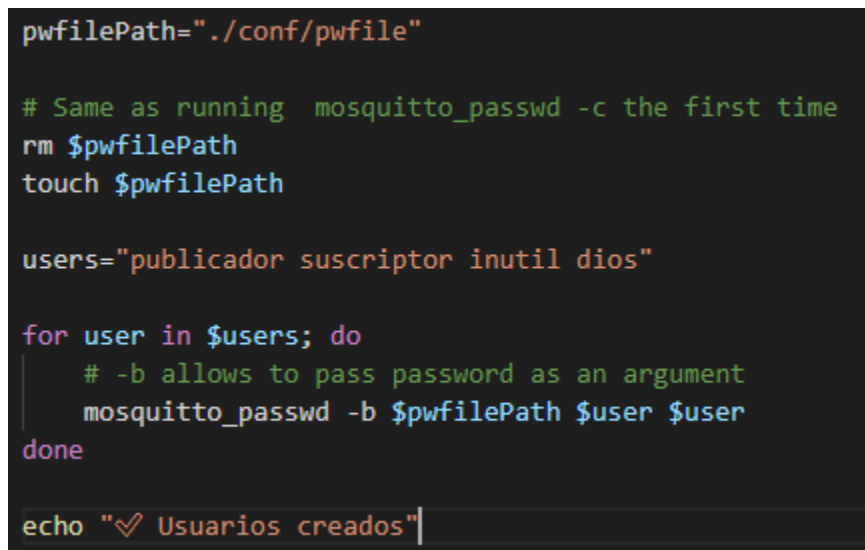


```
→ ejercicio1 git:(main) x mosquitto_sub -t test
Connection error: Connection Refused: not authorised.

→ ejercicio1 git:(main) x mosquitto_pub -t test -m "I am anonymous"
Connection error: Connection Refused: not authorised.
```

Figura 5 Clientes anónimos que no pueden conectarse al servidor

Para poder conectar de nuevo al servidor vamos a dar de alta una serie de usuarios. Nos vamos a ayudar del script **create_users.sh** para darnos más prisa. El script define un par de usuarios con el mismo username y contraseña almacenándolos en el fichero **pwfile** dentro del directorio **conf**.



```
pwfilePath="./conf/pwfile"

# Same as running mosquitto_passwd -c the first time
rm $pwfilePath
touch $pwfilePath

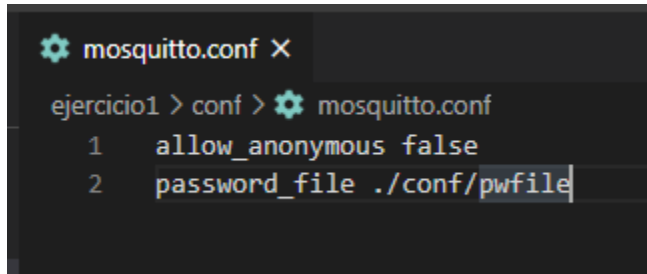
users="publicador suscriptor inutil dios"

for user in $users; do
    # -b allows to pass password as an argument
    mosquitto_passwd -b $pwfilePath $user $user
done

echo "✔ Usuarios creados"
```

Figura 6 Código fuente del create_users.sh

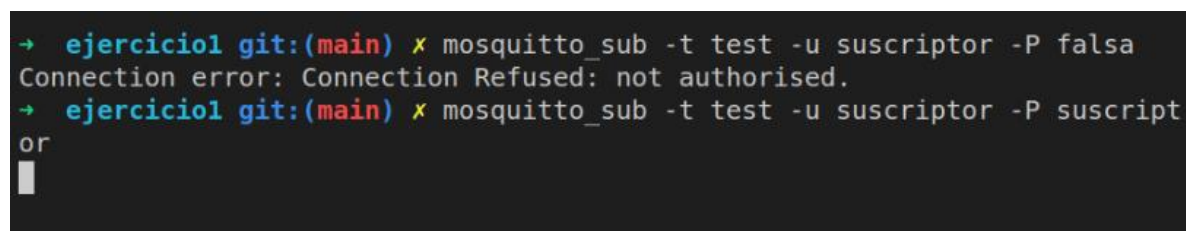
Ahora solamente nos hace falta referenciar el fichero **pwfile** desde el mosquitto.conf.



```
mosquitto.conf X
ejercicio1 > conf > mosquitto.conf
1 allow_anonymous false
2 password_file ./conf/pwfile
```

Figura 7 Versión final del mosquitto.conf del ejercicio1

Con esto hecho podemos reiniciar el servidor y lanzar los mismos comandos que antes, pero indicando las credenciales de los usuarios y deberíamos poder enviar y recibir mensajes.



```
→ ejercicio1 git:(main) x mosquitto_sub -t test -u suscriptor -P falsa
Connection error: Connection Refused: not authorised.
→ ejercicio1 git:(main) x mosquitto_sub -t test -u suscriptor -P suscript
or
█
```

Figura 8 En el primer comando intentamos suscribirnos con unas credenciales invalidas y el servidor nos desconecta. En el siguiente comando usamos las credenciales correctas y ya no recibimos el "Connection Refused: not authorised"

Ejercicio 2

Ejercicio: (2 puntos): Configurar el broker para que pueda controlar el acceso (lectura-escritura-ambos) a los *topics* por parte de los clientes. Configura algunos casos de diferentes *topics* con diferentes permisos para que se pueda comprobar la funcionalidad de ACL del broker mosquitto.

Como se ha comentado en el ejercicio anterior, los suscriptores sólo reciben los mensajes de los temas o *topics* a los que están suscritos. **Mosquitto** nos permite implementar restricción de acceso a recursos (en este caso temas), es decir: nos permite gestionar qué usuarios pueden enviar o recibir mensajes de cada tema.

Estas reglas se configuran en el fichero ACL (*Access Control Lists*) o *acl file*. En nuestro caso definiremos las siguientes reglas:

```
ejercicio2 > conf > 📄 aclfile
user publicador
topic write test

user suscriptor
topic read test

# user 'inutil' no puede hacer nada

user dios
topic readwrite #
```

Figura 9 El usuario publicador puede publicar en test, dios puede hacer de todo en cualquier topic, suscriptor solo puede leer en test e inútil no puede hacer nada

Una vez configuradas las reglas debemos referenciar tal fichero desde **mosquitto.conf**:

```
ejercicio2 > conf > ⚙️ mosquitto.conf
1 allow_anonymous false
2
3 password_file ./conf/pwfile
4 acl_file ./conf/aclfile
```

Figura 10 Versión final del mosquitto.conf del ejercicio2

Con todo esto hecho podemos verificar el correcto funcionamiento de nuestras configuraciones. Para ello debemos arrancar primero el servidor mosquitto y luego ejecutar el fichero **simulation.sh**.

```
→ ejercicio2 git:(main) x ./simulation.sh
Esta simulación ilustra la autorizacion del ACL FILE
'publicador' no es capaz de recibir mensajes cuando se suscribe a 'test'
'suscriptor' no es capaz de enviar mensajes en 'test'
'dios' puede hacer de todo
---> 'dios' como suscriptor, 'publicador' como publicador
'publicador' publica
---> 'suscriptor' como suscriptor, 'dios' como publicador
'dios' publica
✓✓ Fin ✓✓
```

Figura 11 Output de la ejecución del fichero simulation.sh

```
ejercicio2 > simulation.sh
1  echo "Esta simulación ilustra la autorizacion del ACL FILE"
2
3  echo "'publicador' no es capaz de recibir mensajes cuando se suscribe a 'test'"
4
5  mosquitto_sub -t test -u publicador -P publicador &
6  sleep 1
7  mosquitto_pub -t test -m "'dios' puede publicar donde quiera" -u dios -P dios
8  pkill -f mosquitto_sub
9
10 echo "'suscriptor' no es capaz de enviar mensajes en 'test'"
11
12 mosquitto_sub -t test -u dios -P dios &
13 sleep 1
14 mosquitto_pub -t test -m "Dios puede publicar donde quiera" -u suscriptor -P suscriptor
15 pkill -f mosquitto_sub
16
17 echo "'dios' puede hacer de todo"
18
19 echo "---> 'dios' como suscriptor, 'publicador' como publicador"
20 mosquitto_sub -t test -u dios -P dios &
21 sleep 1
22 mosquitto_pub -t test -m "'publicador' publica" -u publicador -P publicador
23 pkill -f mosquitto_sub
24
25 echo "---> 'suscriptor' como suscriptor, 'dios' como publicador"
26 mosquitto_sub -t test -u suscriptor -P suscriptor &
27 sleep 1
28 mosquitto_pub -t test -m "'dios' publica" -u dios -P dios
29 pkill -f mosquitto_sub
30
31
32 echo "✓✓ Fin ✓✓"
```

Figura 12 Código fuente el simulation.sh

Como esto puede ser poco ilustrativo vamos a revisar paso a paso las figuras 11 y 12 para demostrar que nuestra simulación se ha ejecutado acordemente.

Primero de todo lanzamos un proceso en segundo plano que se suscribe en el tópico “test” con el usuario “publicador”, que no tiene permisos. Esperamos un segundo antes de hacer que el usuario “dios” publique en el tópico “test” y vemos que no hay ningún output adicional. Las credenciales de “publicador” son válidas (y por eso el proceso no muere inmediatamente) pero no tiene derecho a recibir mensajes desde el tópico “test” y por ende no se imprime por **stdout** el mensaje que “dios” ha publicado.

A continuación, tenemos el caso inverso: lanzamos a “dios” en background para escuchar mensajes en el tópico “test” pero no tenemos output ya que el usuario que intenta publicar es “suscriptor” y no tiene permisos. Obtuvimos el mismo resultado que antes ya que “suscriptor” usa credenciales válidas, pero no tiene permisos y por ende “dios” no llega a escuchar nada.

Finalmente probamos los casos que “están bien”:

1. Lanzamos a “dios” como suscriptor y “publicador” publica: podemos ver como se imprime *‘publicador’ publica* por la consola de comandos justo después del comando echo.
2. Lanzamos a “suscriptor” como suscriptor y “dios” publica: podemos ver como se imprime *‘dios’ publica* por la consola de comandos justo después del comando echo.

Estos mensajes que aparecen en pantalla son los argumentos que reciben por el argumento *-m* (message) dentro del script de simulación.

Ejercicio 3

Ejercicio: (3 puntos) Seguir las indicaciones dadas para establecer una conexión TLS cifrada con autenticación del broker por parte del cliente.

Para establecer una comunicación bajo TLS lo primero que debemos hacer es generar una autoridad de certificación (*Certification Authority*, CA) de la cual tanto el servidor como el cliente deben fiarse. De tal manera que los certificados que generemos a posteriori del cliente y servidor estén firmados por esta CA. Cuando el cliente reciba un certificado por TLS del servidor podrá decidir si fiarse o no basándose en si ese certificado lo acredita una CA en la que confía. Lo que se pretende es que el cliente, antes de establecer la comunicación con el bróker, determine si otro “es quien dice ser”.

Para resolver el ejercicio 3, hemos creado una CA como se puede observar en las figuras 13 y 14.


```
ejercicio3 > conf > tls > ca >  gen_ca.sh
1  #!/bin/bash
2
3  # rellenar acordemente. password: master
4  openssl req -new -x509 -days 365 -extensions v3_ca -keyout ca.key -out ca.crt
```

Figura 13 Script `gen_ca.sh` que genera los archivos necesarios de la certification authority

```
+ ca git:(main) x ./gen_ca.sh
Generating a RSA private key
.....+++++
..+++++
writing new private key to 'ca.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
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]:ES
State or Province Name (full name) [Some-State]:IB
Locality Name (eg, city) []:PALMA
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UIB
Organizational Unit Name (eg, section) []:GRUPO6
Common Name (e.g. server FQDN or YOUR name) []:MUSI
Email Address []:miquelrc95@outlook.es
```

Figura 14 Rellenado del formulario para generar la CA

Posteriormente generamos las claves y certificados del bróker firmándolos por la CA como se puede ver en la figura 15.

```
ejercicio3 > conf > tls > server > gen_server.sh
1  #!/bin/bash
2
3  openssl genrsa -out broker.key 2048
4
5  openssl req -out broker.csr -key broker.key -new
6
7  openssl x509 -req -in broker.csr -CA ../ca/ca.crt -CAkey ../ca/ca.key -CAcreateserial -out broker.crt -days 365
```

Figura 15 Script gen_server.sh que genera los certificados y claves del servidor firmándolos por nuestra CA generada previamente

Una vez generados los certificados y juegos de claves debemos referenciarlos desde el fichero **mosquitto.conf**. Además, cambiamos el puerto al 8883, que es donde suele correr el MQTT-over-TLS e indicamos que versión de TLS va a usar nuestro servidor. En nuestro caso será la versión 1.2.

```
ejercicio3 > conf > mosquitto.conf
1  # Configuración de Autenticación
2
3  allow_anonymous false
4  password_file ./conf/pwfile
5  acl_file ./conf/aclfile
6
7  # Configuración del TLS
8
9  port 8883
10
11  tls_version tlsv1.2
12  cafile ./conf/tls/ca/ca.crt
13  certfile ./conf/tls/server/broker.crt
14  keyfile ./conf/tls/server/broker.key
```

Figura 16 Configuración final y correcta para el mosquitto en el ejercicio3

Para demostrar que todo esto funciona y no nos estamos tirando un farol hemos preparado dos simulaciones para saber si el tráfico va cifrado o no por la red. Los scripts **sniff1883.sh** y **sniff8883.sh** son utilidades que hemos creado para poder ver el tráfico que circula en susodichos puertos. Mientras que el script **unsafemosquitto.sh** arranca un servidor mosquitto referenciando al fichero **unsafemosquitto.conf** cuyo contenido se muestra en la figura 17.

```
ejercicio3 > conf > ⚙️ unsafemosquitto.conf
1  # Configuracion de Autenticacion
2
3  allow_anonymous false
4  password_file ./conf/pwfile
5  acl_file ./conf/aclfile
6
7  # Configuracion del TLS
8
9  # port 8883
10
11 # tls_version tlsv1.2
12 # cafile ./conf/tls/ca/ca.crt
13 # certfile ./conf/tls/server/broker.crt
14 # keyfile ./conf/tls/server/broker.key
15
```

Figura 17 Contenido del **unsafemosquitto.conf**. No tiene ninguna opción de TLS configurada

A continuación, se presenta la ejecución del script **unsafe_pub.sh** donde simplemente se publica un mensaje sin usar configuración de TLS alguna mientras nuestro script **sniff1883.sh** ya está corriendo.

```
➔ ejercicio3 git:(main) ✗ sudo ./sniff1883.sh
[sudo] contraseña para miquel:
reportfilename: ./report.xml
tcpflow: listening on any
127.000.000.001.43678-127.000.000.001.01883:
0000: 102f 0004 4d51 5454 04c2 003c 0017 6d6f 7371 2d35 4431 6d69 3445 694c 7156 6b44  ../MQTT...<
..mosq-5Dlmi4EilqVkd
0020: 4161 4f49 6100 0464 696f 7300 0464 696f 73                                AaOIa..dios.
.dios

127.000.000.001.01883-127.000.000.001.43678:
0000: 2002 0000 ...

127.000.000.001.43678-127.000.000.001.01883:
0000: 300c 0004 7465 7374 6265 726e 6174 0..testbernat

127.000.000.001.43678-127.000.000.001.01883:
0000: e000 ..
```

Figura 18 Podemos observar el payload del mensaje MQTT, en el las credenciales del usuario "dios" así como el tópico "test" y el contenido del mensaje: "bernat". Claramente el tráfico no está siendo cifrado.

Una vez hecho esto usamos la configuración de la figura 16 y repetimos el ejercicio: Ahora usando **sniff8883.sh** y el script **safe_pub.sh**.

```
ejercicio3 > safe_pub.sh
1  #!/bin/bash
2
3  | mosquitto_pub -m "bernat" -t test -u dios -P dios --cafile ./conf/tls/ca/ca.crt --tls-version tlsv1.2
```

Figura 19 El mismo código que publica el mismo mensaje en el mismo tópico pero esta vez usando la configuración TLS

```
0000: 1603 0300 2510 0000 2120 c403 74ba 0421 8f6d 0dd1 04aa b9a0 e802 951e 820d 3dae .....%...! ..
t...!M.....=..
0020: 9c70 d71d a7bc 439a 9218 1403 0300 0101 1603 0300 2820 2c4d f6cf ba35 a90a d5ab ..p....C.....
.....( ,M...S....
0040: b1f5 1196 4a93 756b d2a5 ff82 9920 d12e a922 8a70 1e33 5d14 4ccf 7968 35 .....J.uk....
. ....".p.3j .L.yh5

127.000.000.001.08883-127.000.000.001.55896:
0000: 1603 0300 ba04 0000 b600 001c 2000 b046 0588 c946 5432 e7d6 c661 1551 fea0 9ec2 .....
..F...FT2...a.Q....
0020: b886 bfcc 52a3 e8c4 3556 de02 d622 0245 706a 5542 298a d1e9 f135 2be6 e4f7 8522 ....R...SV..
..".EpjUB)....5+...."
0040: 9019 9fbb 025e 89d6 eb10 9b6d 106a 2329 38fa e2c7 d412 a945 711d 9626 0b3f a2bc .....^.....m
.j#)8.....Eq..&.?..
0060: e80f e20b dd27 1de9 96b3 14ef 2499 a933 f594 7629 359b 5b89 3331 a688 6f5b 97aa ...../.....
$.3..v)5.[.31..o[...
0080: bd16 4be0 38ff 6cc8 149a 8851 56ca 598d 59dc 2d69 e16e 67ab 3c14 3ee9 35a5 c927 ..K.B.l....Q
V.Y.Y.-i.ng.<.>.5..
00a0: 9f41 386a 0286 56ef 7e86 a69d dd1d b79b 1b82 e96f eb93 c954 86c9 56d9 b0cf 5d14 .A8j...V.~...
.....o...T..V...].
00c0: 0303 0001 0116 0303 0028 b6e0 191b 3a64 c20f bf3a a045 a22a abea a685 1cfd 5811 .....(..
...d.....E.*.....X.
00e0: 79a5 a2b0 6c35 9013 2955 338d 0087 af78 fee5 y...l5..)U3.
...x...

127.000.000.001.55896-127.000.000.001.08883:
0000: 1703 0300 4920 2c4d f6cf ba35 aaff f45f fa2b fbe4 9741 a0f0 fe5a 481b 49a6 d1c2 ....I ,M...5
....+...A...ZH.I...
0020: a2d1 af5d 09e3 d7c0 8965 2822 833f d618 6ba4 2d41 080b c1b9 877f 8539 0bf5 4c33 ...]......e("
..?.k.-A.....9..L3
0040: 8396 6f70 cb0e f6b5 2008 aald 0e61 ..op....
.a

127.000.000.001.08883-127.000.000.001.55896:
0000: 1703 0300 1cb6 e019 1b3a 64c2 1043 0262 004d c5f9 0165 96f2 1500 9b20 7fea bd06 .....:d.
.C.b.M...e.....
0020: 06

127.000.000.001.55896-127.000.000.001.08883:
0000: 1703 0300 2620 2c4d f6cf ba35 abc6 fa06 f040 26da c450 c73c d9e9 0c12 24d5 de68 ....& ,M...5
.....@&..P.<....$.h
0020: a8f2 3f3e 0c67 b213 bb2f 2d ..?>.g.../-

127.000.000.001.55896-127.000.000.001.08883:
0000: 1703 0300 1a20 2c4d f6cf ba35 acd7 0061 89dc d5cf 3cf8 9727 9adb 816a 0a47 26 ..... ,M...5...
a....<...'...j.G&

127.000.000.001.55896-127.000.000.001.08883:
0000: 1503 0300 1a20 2c4d f6cf ba35 ad42 flca 8aba f723 bbd3 a829 25af 7ab2 fb24 aa ..... ,M...5.B.
....#....)%..Z...$.

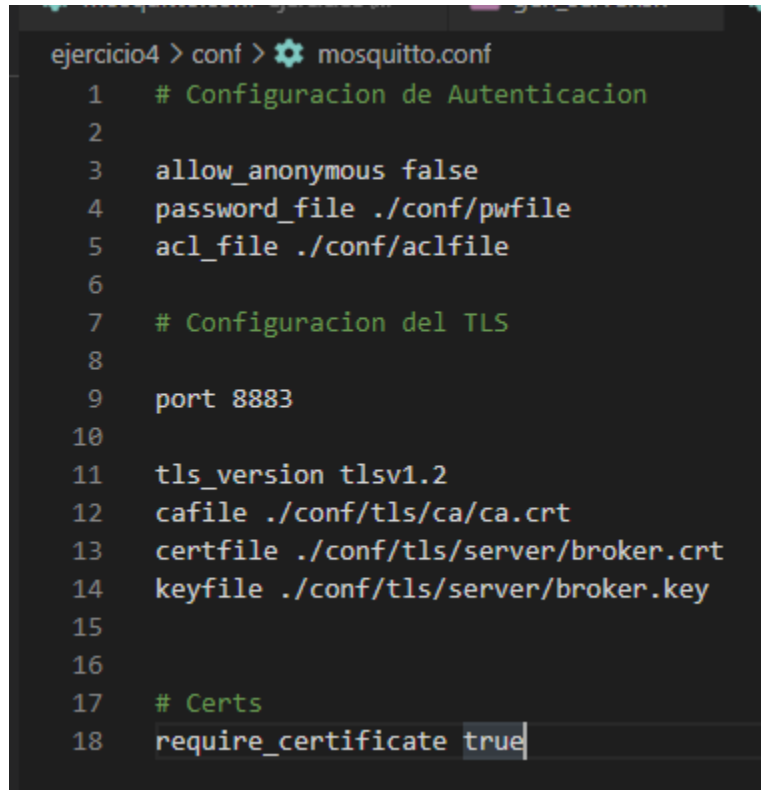
127.000.000.001.08883-127.000.000.001.55896:
0000: 1503 0300 1ab6 e019 1b3a 64c2 1148 3ff3 7033 97fa a371 4acc 4dc6 3758 a4ec 19 .....:d..H?
.p3...qj.M.7X...
```

Figura 20 Output del "sniffer" de tráfico. No entendemos nada ya que todo está cifrado

Ejercicio 4

Ejercicio: (4 puntos) Generar el material necesario y configurar cliente y servidor para establecer una conexión TLS cifrada con autenticación del broker por parte del cliente, y del cliente por parte del broker.

El bróker mosquitto tiene una opción de configuración para que el servidor valide también al cliente. Para ello hay que modificar el fichero **mosquitto.conf** y añadir lo siguiente:



```
ejercicio4 > conf > mosquitto.conf
1  # Configuracion de Autenticacion
2
3  allow_anonymous false
4  password_file ./conf/pwfile
5  acl_file ./conf/aclfile
6
7  # Configuracion del TLS
8
9  port 8883
10
11  tls_version tlsv1.2
12  cafile ./conf/tls/ca/ca.crt
13  certfile ./conf/tls/server/broker.crt
14  keyfile ./conf/tls/server/broker.key
15
16
17  # Certs
18  require_certificate true
```

Figura 21. Añadimos la nueva opción: `require_certificate true` para requerir que los clientes se autenticuen con sus certificados

Desde este momento, el servidor mosquitto requerirá un certificado a todos los clientes que se intenten conectar.

Por tanto, igual que se ha hecho anteriormente con el servidor, se crea un certificado y un par de claves para el cliente y se firman por la autoridad de certificación que creamos antes.

```
ejercicio3 > conf > tls > client > gen_client.sh
1  #!/bin/bash
2
3  openssl genrsa -out client.key 2048
4
5  openssl req -out client.csr -key client.key -new
6
7  openssl x509 -req -in client.csr -CA ../ca/ca.crt -CAkey ../ca/ca.key -CAcreateserial -out client.crt -days 365
```

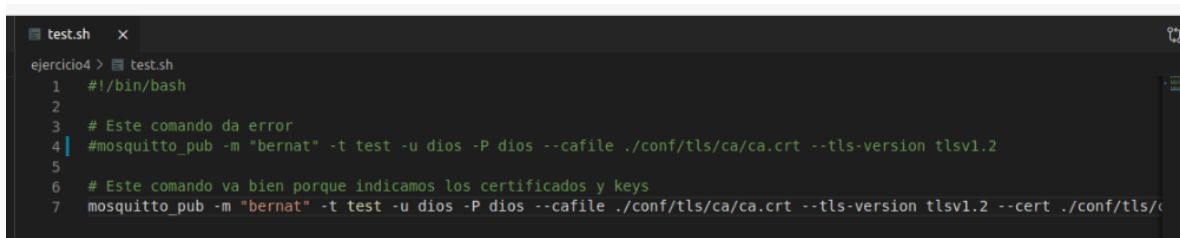
Figura 22: Código `gen_client.sh` que genera el certificado a usar desde los clientes de mosquito

```
→ client git:(main) x ./gen client.sh
Generating RSA private key, 2048 bit long modulus (2 primes)
.....+++++
.....+++++
e is 65537 (0x010001)
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]:ES
State or Province Name (full name) [Some-State]:IB
Locality Name (eg, city) []:PALMA
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UIB
Organizational Unit Name (eg, section) []:GRUP06
Common Name (e.g. server FQDN or YOUR name) []:client
Email Address []:miquelrc95@outlook.es

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:master
An optional company name []:master
Signature ok
subject=C = ES, ST = IB, L = PALMA, O = UIB, OU = GRUP06, CN = client, emailAddress = miquelrc95@outlook.es
Getting CA Private Key
Enter pass phrase for ../ca/ca.key:
→ client git:(main) x
```

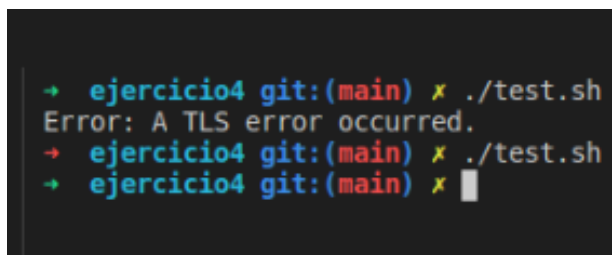
Figura 23. Rellenado del formulario de la generacion de los certificados de los clientes

Y finalmente hacemos una simulación de test: Probamos de publicar con una configuración en la que **PRIMERO NO** enviamos los datos de nuestro certificado como clientes y otra en la que **SÍ**. Esto se ve ilustrado por las figuras 24 y 25.



```
test.sh x
ejercicio4 > test.sh
1  #!/bin/bash
2
3  # Este comando da error
4  #mosquitto_pub -m "bernat" -t test -u dios -P dios --cafile ./conf/tls/ca/ca.crt --tls-version tlsv1.2
5
6  # Este comando va bien porque indicamos los certificados y keys
7  mosquitto_pub -m "bernat" -t test -u dios -P dios --cafile ./conf/tls/ca/ca.crt --tls-version tlsv1.2 --cert ./conf/tls/c
```

Figura 24: Código del script test.sh donde tenemos las dos instrucciones necesarias para poder verificar si nuestra configuración funciona



```
→ ejercicio4 git:(main) x ./test.sh
Error: A TLS error occurred.
→ ejercicio4 git:(main) x ./test.sh
→ ejercicio4 git:(main) x
```

Figura 25: Ejecutar el script sin enviar nuestro certificado resulta en "Error: A TLS error occurred.". Mientras que la segunda vez el comando finaliza de manera satisfactoria