

# Sfruttamento delle vulnerabilità del protocollo MQTT tramite un attacco DoS slow rate su dispositivi IoT

Gianluca Boschi, Francesco Carli, Paola Petri

December 2020

# Indice

<b>1</b>	<b>Descrizione del problema</b>	<b>2</b>
1.1	Connessione: CONNECT & CONNACK . . . . .	3
<b>2</b>	<b>Implementazione</b>	<b>4</b>
<b>3</b>	<b>Risultati</b>	<b>6</b>
3.1	Singolo client . . . . .	6
3.2	SlowITe DoS Attack . . . . .	8
<b>4</b>	<b>Conclusioni</b>	<b>11</b>
4.1	Appendice . . . . .	11

# Capitolo 1

## Descrizione del problema

MQTT é un *protocollo applicativo* di tipo **publish-subscribe**, basato sullo stack TCP/IP, utilizzato frequentemente nelle comunicazioni tra i dispositivi IoT. Nel protocollo sono coinvolte tre entità, un **publisher**, un **subscriber** e un **broker**; il publisher pubblica dei messaggi su un certo topic, che saranno inoltrati e visualizzati da tutti i subscriber che si sono registrati a tale topic. Il broker è simile ad un server: esso ha ruolo di mediatore tra publisher e subscriber: riceve i vari messaggi e li filtra per inoltrarli ai giusti client. Il broker mantiene quindi i topic e i messaggi, ma non permanentemente.

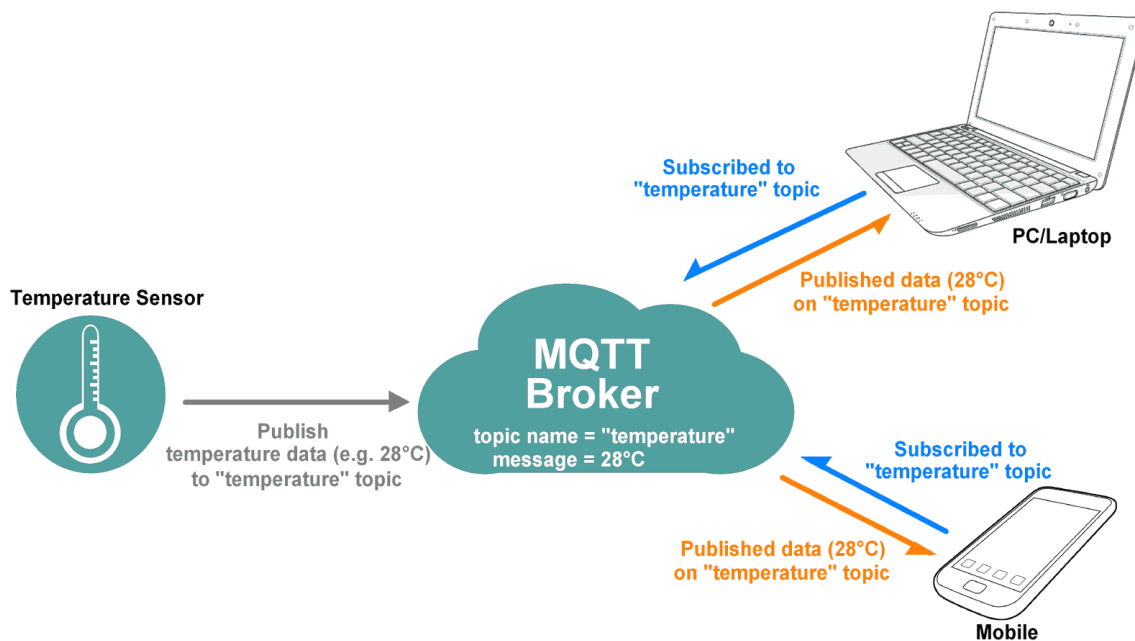


Figura 1.1: Funzionamento del protocollo MQTT.

In questo progetto ci focalizziamo sullo sfruttamento di una vulnerabilità di questo protocollo che lo espone ad attacchi DoS. Il tipo di attacco DoS che andiamo ad analizzare è SlowITe: esso adotta un approccio slow rate che utilizza una piccola quantità di banda per mantenere le comunicazioni attive il più a lungo possibile. Lato “server” (broker) utilizzeremo una macchina con Eclipse Mosquitto, un server MQTT; lato client eseguiremo il software SlowITe. SlowITe avvia varie connessioni client verso il broker fino a saturarlo per tentare di rendere il servizio irraggiungibile. Questo avverrà tramite la modifica del parametro **keep alive**, che regola la durata di una connessione attiva e che è contenuto nella richiesta di connessione, inviata dal client al broker nel messaggio *CONNECT*. Sarà anche necessaria una terza entità (client) che tenterà di accedere al broker MQTT per verificare il raggiungimento dello stato di DoS, ossia uno stato a seguito del quale il broker MQTT risulterà non disponibile.

## 1.1 Connessione: CONNECT & CONNACK

Durante la richiesta di connessione del client, vengono scambiati due diversi tipi di messaggi *MQTT*: *CONNECT* e *CONNACK*. Il client si connette al broker inviando un messaggio *CONNECT*, nel quale specifica il parametro keep alive, il suo ID, e altre informazioni opzionali. Il broker conferma la richiesta del client con un messaggio *CONNACK*, che contiene un flag con l’esito di tale richiesta. Dopo la connessione il client sarà in grado di pubblicare messaggi, i quali conterranno un topic e un payload.

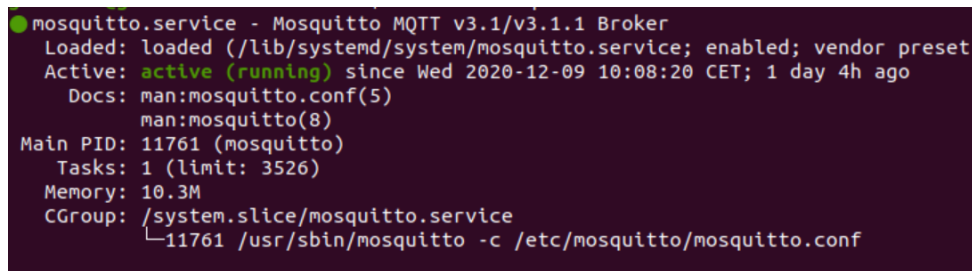
# Capitolo 2

## Implementazione

Per implementare il software di attacco SlowIte è stata utilizzata la libreria Paho MQTT Client<sup>1</sup> di Python<sup>2</sup>.

Lo schema implementativo utilizzato è il seguente:

1. Settare il broker Mosquitto su una macchina Ubuntu Linux e renderlo disponibile online



```
● mosquitto.service - Mosquitto MQTT v3.1/v3.1.1 Broker
   Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset:
   Active: active (running) since Wed 2020-12-09 10:08:20 CET; 1 day 4h ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
   Main PID: 11761 (mosquitto)
      Tasks: 1 (limit: 3526)
     Memory: 10.3M
    CGroup: /system.slice/mosquitto.service
            └─11761 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf
```

Figura 2.1: Broker Mosquitto in esecuzione.

2. Scrivere il codice dell'attacco SlowIte usando la libreria Paho MQTT Client.
3. Scrivere un codice per verificare il corretto funzionamento del sistema con un solo client e i tempi di connessione e disconnessione.
4. Avviare l'attacco SlowIte verso il broker.
5. Scrivere il codice per un publisher e un subscriber per verificare il raggiungimento dello stato di DoS sul broker.

---

<sup>1</sup><https://pypi.org/project/paho-mqtt/>

<sup>2</sup><https://www.python.org/>

6. Verificare la saturazione del broker tramite analisi dei pacchetti su Wireshark.
7. Verificare i risultati ottenuti e confrontarli con i risultati ottenuti da Ivan Vaccari, Maurizio Aiello e Enrico Cambiaso riportati nel paper "*SlowITe, a Novel Denial of Service Attack Affecting MQTT*".

### **Strumenti utilizzati**

- Hardware: Intel(R) Core(TM) i5-6267U CPU @ 2.90GHz (with SSE4.2)
- OS: Linux 4.15.0-108-generic
- Application: Dumpcap (Wireshark) 2.6.10 (Git v2.6.10 packaged as 2.6.10-1 ubuntu18.04.0)
- Libreria lato client: <https://pypi.org/project/paho-mqtt/>

# Capitolo 3

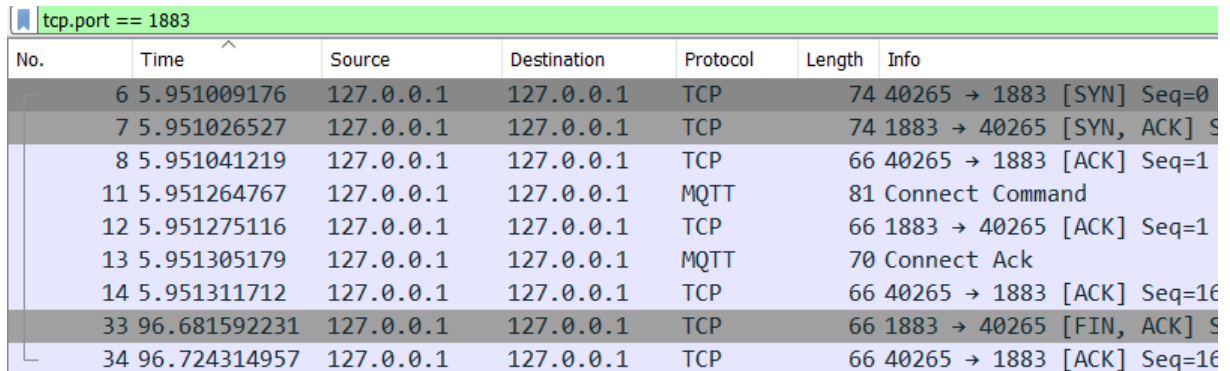
## Risultati

### 3.1 Singolo client

Abbiamo testato il funzionamento del broker Mosquitto facendo connettere un singolo client tramite il metodo *connect()*, i cui parametri sono: l'indirizzo IP del broker, la porta (porta di default 1883) e il parametro **keep alive**.

Settando il valore del *keep alive* a 60, abbiamo verificato, tramite Wireshark, la correttezza del *three way handshake*, della richiesta di connessione tramite *CONNECT* e della successiva notifica tramite *CONNACK*.

Nel protocollo MQTT il broker mantiene la connessione attiva per un tempo pari a  $1.5 \cdot keepalive$  (in questo caso  $60s \cdot 1.5 = 90s$ ), dopodiché invierà un pacchetto di FIN per chiudere la connessione.



The image shows a Wireshark packet capture for the filter 'tcp.port == 1883'. The table below represents the data shown in the packet list pane.

No.	Time	Source	Destination	Protocol	Length	Info
6	5.951009176	127.0.0.1	127.0.0.1	TCP	74	40265 → 1883 [SYN] Seq=0
7	5.951026527	127.0.0.1	127.0.0.1	TCP	74	1883 → 40265 [SYN, ACK] S
8	5.951041219	127.0.0.1	127.0.0.1	TCP	66	40265 → 1883 [ACK] Seq=1
11	5.951264767	127.0.0.1	127.0.0.1	MQTT	81	Connect Command
12	5.951275116	127.0.0.1	127.0.0.1	TCP	66	1883 → 40265 [ACK] Seq=1
13	5.951305179	127.0.0.1	127.0.0.1	MQTT	70	Connect Ack
14	5.951311712	127.0.0.1	127.0.0.1	TCP	66	40265 → 1883 [ACK] Seq=16
33	96.681592231	127.0.0.1	127.0.0.1	TCP	66	1883 → 40265 [FIN, ACK] S
34	96.724314957	127.0.0.1	127.0.0.1	TCP	66	40265 → 1883 [ACK] Seq=16

Figura 3.1: Pacchetti scambiati nella connessione tra client e broker con keep alive = 60.

Nella Tabella 3.1 sono state raggruppate le tempistiche relative allo scambio dei pacchetti. In particolare è interessante notare quanto tempo trascorre tra il primo SYN, che rappresenta l'inizio della connessione, e l'ACK<sub>FIN</sub> che indica la chiusura della connessione: questo tempo corrisponde a circa 90 secondi, confermando le nostre previsioni.

	<b>Tempo</b>
$SYN_{1^o} \rightarrow CONNECT$	0,000255591 sec
$CONNECT \rightarrow CONNACK$	0,000040412 sec
$CONNACK \rightarrow FIN$	90,73028705 sec
$CONNACK \rightarrow ACK_{FIN}$	90,77300970 sec
$SYN_{1^o} \rightarrow ACK_{FIN}$	90,77330577 sec

Tabella 3.1: Ritardi tra i diversi pacchetti.

SYN<sub>1</sub>: primo pacchetto SYN della connessione (quello relativo al *three way handshake*).

ACK<sub>FIN</sub>: rappresenta l'acknowledgement relativo al FIN di chiusura connessione TCP

Sono state effettuate anche delle misurazioni in termini di utilizzo di banda e dimensione dei dati scambiati nel protocollo:

Dimensione totale comunicazione	629 byte
Durata della comunicazione	90.77330577 sec
Banda utilizzata	55.435 bps

Tabella 3.2: Dati sulla comunicazione.

È importante sottolineare che la dimensione totale della comunicazione, espressa in bytes, varierà in base al client ID utilizzato: dato che esso potrà essere una stringa, la lunghezza di quest'ultima potrà influenzare la dimensione del pacchetto *CONNECT* e, di conseguenza, quella dell'intera comunicazione.

La banda utilizzata è stata calcolata tramite la semplice formula:

$$Banda = \frac{DimensioneComunicazione}{DurataComunicazione} \quad (3.1)$$



## 3.2 SlowITe DoS Attack

L'attacco consiste nel tentare di saturare il broker instaurando un grande numero di connessioni: superato un certo numero di sessioni attive sarà raggiunto lo stato di DoS e il broker non sarà più disponibile. L'obiettivo di questo attacco è anche quello di identificare il numero massimo di client simultanei che il broker MQTT è in grado di gestire. Nel codice relativo all'attacco sono stati creati 1024 client diversi, salvati successivamente in un vettore. La scelta del numero di client da generare è data dalle connessioni che il broker MQTT Mosquitto riesce a gestire di default, che sono circa 1024 (per maggiori dettagli: GitHub/Mosquitto). Ai client sono stati assegnati degli ID progressivi ( $client_1$   $client_2$  ...  $client_{1024}$ ) i quali hanno dimensione massima di 10 byte. I client richiedono in sequenza l'apertura di una connessione, tramite un ciclo for, con il metodo *connect()*, nel quale viene settato il parametro *keep alive* da input. Lo script infine mostra il progresso nell'instaurare le connessioni, che saranno mantenute attive fino alla terminazione dello script o per tutta la durata del tempo  $keepalive \cdot 1,5$ . Per terminare l'attacco prima di aver esaurito il timeout dettato dal keepAlive, l'utente può interrompere l'esecuzione dello script premendo un qualsiasi tasto.

La non disponibilità del broker è stata verificata tramite altri due client che tentano di fare operazioni di tipo *publish* o *subscribe* e tramite l'analisi su Wireshark del pcap ottenuto dal monitoraggio della rete effettuato durante l'esecuzione del codice d'attacco.

È stato verificato che la soglia per lo stato di DoS sul broker è di circa 1015 client connessi simultaneamente: oltre questo valore il broker non riesce a gestire altre richieste di qualsiasi tipo. Individuare il numero massimo di connessioni simultanee che il broker riesce a gestire è stato immediato tramite l'analisi del pcap: nel momento in cui il broker non è più disponibile, esso cessa di inviare pacchetti di tipo *CONNACK*. Filtrando il pcap in modo da visualizzare solo questo tipo di pacchetti, si può osservare che il numero totale di connessioni accettate è 1015.

CONNECT	CONNACK	Connessioni non accettate
1024	1015	9

Tabella 3.3: Dati risultati dall'analisi con Wireshark.

tcp.port==1883 and mqtt					
Source	Destination	Protocol	Source Port TCP	Destination Port TCP	Info
127.0.0.1	127.0.0.1	MQTT	1883	56275	Connect Ack
127.0.0.1	127.0.0.1	MQTT	50323	1883	Connect Command
127.0.0.1	127.0.0.1	MQTT	1883	50323	Connect Ack
127.0.0.1	127.0.0.1	MQTT	38169	1883	Connect Command
127.0.0.1	127.0.0.1	MQTT	1883	38169	Connect Ack
127.0.0.1	127.0.0.1	MQTT	59461	1883	Connect Command
127.0.0.1	127.0.0.1	MQTT	1883	59461	Connect Ack
127.0.0.1	127.0.0.1	MQTT	43739	1883	Connect Command
127.0.0.1	127.0.0.1	MQTT	1883	43739	Connect Ack
127.0.0.1	127.0.0.1	MQTT	42221	1883	Connect Command
127.0.0.1	127.0.0.1	MQTT	1883	42221	Connect Ack
127.0.0.1	127.0.0.1	MQTT	52953	1883	Connect Command
127.0.0.1	127.0.0.1	MQTT	1883	52953	Connect Ack
127.0.0.1	127.0.0.1	MQTT	37055	1883	Connect Command
127.0.0.1	127.0.0.1	MQTT	56911	1883	Connect Command
127.0.0.1	127.0.0.1	MQTT	41239	1883	Connect Command
127.0.0.1	127.0.0.1	MQTT	44741	1883	Connect Command
127.0.0.1	127.0.0.1	MQTT	50241	1883	Connect Command
127.0.0.1	127.0.0.1	MQTT	58171	1883	Connect Command
127.0.0.1	127.0.0.1	MQTT	48037	1883	Connect Command
127.0.0.1	127.0.0.1	MQTT	59663	1883	Connect Command

Figura 3.2: Porzione del pcap che mostra il momento in cui il broker raggiunge lo stato di DoS. Si noti che dal pacchetto 12193 in poi il broker non invia più *CONNACK*.

No.	Time	Source	Destination	Protocol	Length	Info
13	4.040715204	127.0.0.1	127.0.0.1	MQTT	70	Connect Ack
25	4.041393571	127.0.0.1	127.0.0.1	MQTT	70	Connect Ack
37	4.042010500	127.0.0.1	127.0.0.1	MQTT	70	Connect Ack
52	4.042743006	127.0.0.1	127.0.0.1	MQTT	70	Connect Ack
61	4.043092056	127.0.0.1	127.0.0.1	MQTT	70	Connect Ack
73	4.043640605	127.0.0.1	127.0.0.1	MQTT	70	Connect Ack
85	4.044182491	127.0.0.1	127.0.0.1	MQTT	70	Connect Ack
97	4.044754603	127.0.0.1	127.0.0.1	MQTT	70	Connect Ack
110	4.045550723	127.0.0.1	127.0.0.1	MQTT	70	Connect Ack
121	4.046097565	127.0.0.1	127.0.0.1	MQTT	70	Connect Ack
133	4.046666959	127.0.0.1	127.0.0.1	MQTT	70	Connect Ack
146	4.047262662	127.0.0.1	127.0.0.1	MQTT	70	Connect Ack
160	4.049846739	127.0.0.1	127.0.0.1	MQTT	70	Connect Ack
169	4.053054760	127.0.0.1	127.0.0.1	MQTT	70	Connect Ack
181	4.053626850	127.0.0.1	127.0.0.1	MQTT	70	Connect Ack
193	4.054208610	127.0.0.1	127.0.0.1	MQTT	70	Connect Ack
205	4.054855542	127.0.0.1	127.0.0.1	MQTT	70	Connect Ack
12193	4.060606703	127.0.0.1	127.0.0.1	MQTT	70	Connect Ack

Frame 13: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface lo, id 0

```

0000  00 00 00 00 00 00 00 00 00 00 00 00 00 45 00  .....E
0010  00 38 82 04 40 00 00 06 ba b9 7f 00 00 01 7f 00  -8-@-
0020  00 01 07 5b 8d 5d de ab 8a 1a 0b c7 58 4d 00 18  -[-]---X
0030  02 00 fe 2c 00 00 01 01 00 0a d1 b2 85 b5 d1 b2  .....
0040  85 b5 20 02 00 00  .....

```

Pacchetti: 12275 - visualizzati: 1013 (8.3%)

Figura 3.3: Pacchetti *CONNACK*.

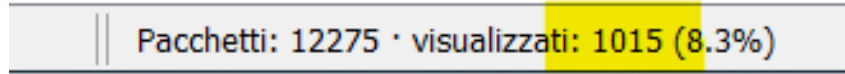


Figura 3.4: Pacchetti *CONNACK* totali.

Con questi risultati abbiamo quindi rilevato: un numero totale  $N_{tot} = 1024$  client che hanno richiesto una connessione, un numero  $N_m = 1015$  di client che hanno stabilito una connessione e un numero  $N_e = 9$  di client che non hanno ricevuto il *CONNACK* e che quindi non sono riusciti a connettersi al broker poiché saturo.

Abbiamo anche testato la corretta chiusura delle connessioni stabilite allo scadere del timeout keep alive  $\cdot 1.5$  ed abbiamo rilevato i dati riportati nella Tabella 3.4.

Tempo di arrivo del primo pacchetto della comunicazione (SYN)	4,625 sec
Tempo di arrivo dell'ultimo pacchetto della comunicazione (ACK)	97,599 sec
Durata totale	92,974 sec
Dimensione totale della comunicazione	649.981 byte
Banda utilizzata per l'attacco	55.927,98 bps

Tabella 3.4: Dati relativi ai pacchetti della comunicazione.

La non disponibilità del broker è stata verificata tramite altre due entità client, una con il ruolo di *publisher* e una con il ruolo di *subscriber*. L'entità *subscriber* si sottoscrive ad un topic e il *publisher* pubblica un contenuto su quel topic. Prima dell'attacco viene normalmente creato il topic scelto dal *subscriber* e viene stampata a video il messaggio del *publisher*. Durante l'attacco vengono eseguiti nuovamente il codice del publisher e quello del subscriber: in questo caso però il broker non sarà disponibile e quindi nessun contenuto verrà pubblicato.

# Capitolo 4

## Conclusioni

In questo lavoro è stato studiato ed esaminato l'argomento dell'IoT, focalizzandosi in particolare sul protocollo MQTT e sullo sfruttamento delle sue vulnerabilità: in particolare sono state evidenziate delle problematiche riguardanti attacchi DoS che sfruttano il parametro KeepAlive. Nel documento sono stati riprodotti attacchi DoS in locale al fine di monitorare l'andamento della rete e il funzionamento delle varie entità coinvolte; proprio da queste simulazioni è stato stabilito il numero massimo di connessioni MQTT contemporanee, che rappresenta inoltre la soglia per il raggiungimento dello stato di DoS sul broker.

### 4.1 Appendice

Il codice sorgente dell'attacco e dei client utilizzati è disponibile al link:

[https://github.com/gianluca2414/MQTT\\_SlowITe](https://github.com/gianluca2414/MQTT_SlowITe)