SAPIENZA
UNIVERSITÀ DI ROMA

Analisi delle problematiche di sicurezza del protocollo MQTT

Facoltà di Ingegneria dell'informazione, informatica e statistica

Corso di Laurea in Informatica

Candidato

Edoardo Di Paolo
Matricola 1728334

Relatore

Prof. Angelo Spognardi

Anno Accademico 2019/2020

Analisi delle problematiche di sicurezza del protocollo MQTT
Tesi di Laurea. Sapienza – Università di Roma

Questa tesi è stata composta con LaTeX e la classe Sapthesis.

Email dell'autore: dipaolo.1728334@studenti.uniroma1.it

Dediche.

Indice

Capitolo 1

Introduzione

Negli ultimi decenni il numero di dispositivi collegati ad Internet è cresciuto esponenzialmente. Ormai, nel 2020, non si hanno più solamente computer o cellulari in rete ma anche elettrodomestici, macchine industriali e strumenti medici, tutti dispositivi che qualche anno fa erano offline. Tutto ciò fa riferimento all'IoT, l'*Internet of Things*.
Parallelamente allo sviluppo di queste nuove tecnologie, sono stati studiati nuovi protocolli affinché i dispositivi potessero essere utilizzati in maniera efficiente. Ad esempio ci sono alcuni sensori i quali permettono di misurare temperatura ed umidità di una stanza e funzionano attraverso l'uso di una semplice pila; dunque è necessario andare a ridurre il costo energetico della connessione così da aumentare la durata di utilizzo del dispositivo.
Alcuni esempi di protocolli possono essere: MQTT, CoAP, AMQP e WebSocket. Ovviamente tutti i protocolli hanno la possibilità di essere integrati con TLS (*Transport Layer Security*) così da poter garantire una maggiore sicurezza nello scambio dei dati; questo, però, potrebbe andare a gravare sui consumi del dispositivo poiché dovrebbero essere effettuati più calcoli affinché avvenga il trasporto dei dati. Inoltre, con l'aumento di questi nuovi dispositivi sono aumentate anche le possibili minacce relative all'IoT. Un esempio è il malware Mirai [6] che, nel 2016, ha infettato milioni di dispositivi rendendoli parte di una botnet la quale, successivamente, ha attaccato attraverso un DDoS il fornitore di servizi DNS Dyn così da rendere inaccessibili milioni di siti web. A causa, anche, di questa tipologia d'attacco, sempre più comune, i protocolli sviluppati devono presentarsi sicuri e robusti.

Negli ultimi anni l'utilizzo del protocollo MQTT è sicuramente cresciuto di gran misura.

Figura 1.1. Numero dispositivi MQTT 2018.

La Figura 1.1 rappresenta il numero di dispositivi che una ricerca della parola "MQTT" produceva nel 2018 sul sito Shodan [1]. Se effettuiamo la stessa ricerca adesso, abbiamo un numero maggiore come si vede dalla figura sottostante.

Figura 1.2. Numero dispositivi MQTT 2020.

La maggior parte di questi dispositivi girano sulla porta standard di MQTT, la 1883; inoltre molti di questi non richiedono alcuna autenticazione per poter connettersi. Effettuando una ricerca ancora più specifica attraverso la stringa "*port:1883*", abbiamo ancora più risultati come si vede dalla Figura 1.3.

Figura 1.3. Numero dispositivi MQTT 2020, port:1883.

In questa relazione viene descritto lo studio effettuato su uno dei maggiori protocolli nell'IoT: MQTT. Nello specifico durante l'attività di tirocinio siamo andati alla ricerca di possibili anomalie e/o violazioni del protocollo da parte dei broker server i quali dovrebbero rispettare ogni standard definito per MQTT.

Nel **Capitolo 2** sarà analizzato il protocollo nel dettaglio assieme alle sue principali caratteristiche e funzionalità.

Nel **Capitolo 3** viene analizzata la libreria che è stata scritta ai fini del tirocinio. In questo capitolo sono anche descritti i differenti broker su cui sono stati effettuati i test e vengono riportati i risultati più interessanti.

Capitolo 2

MQTT

2.1 Descrizione del protocollo

MQTT, acronimo di *Message Queuing Telemetry Transport*, è un protocollo di tipo *publish-subscribe* il quale permette il trasporto di messaggi tra diversi dispositivi tramite TCP/IP. Il protocollo è molto utilizzato in ambito IoT per la sua semplicità e anche per la banda la cui richiesta è davvero bassa.

MQTT presenta un modello di architettura differente, ad esempio, dal tipico client/server del protocollo HTTP; infatti adotta il meccanisco *publish-subscribe* (pub/sub) attraverso l'utilizzo di un *broker*. Il funzionamento del modello pub/sub avviene attraverso la pubblicazione e la sottoscrizione da parte del client a diversi topic, che possono essere paragonati a dei canali di comunicazione. In MQTT il publisher e il subscriber non comunicano mai direttamente e non sono neppure consapevoli della presenza dell'uno e dell'altro: il collegamento è gestito dal broker il cui compito è quello di filtrare i messaggi che riceve e distribuirli ai vari subscribers. In questo capitolo sono analizzate le principali caratteristiche che il protocollo MQTT mette a disposizione.

2.1.1 Topic

Nel protocollo MQTT un topic non è altro che una stringa codificata in UTF-8 che il broker utilizza per filtrare i messaggi da inviare successivamente ad ogni client sottoscritto a quel topic. I topic sono case-sensitive, quindi bisogna prestare attenzione alle lettere maiuscole o minuscole, e la lunghezza minima dev'essere di un carattere. Inoltre, MQTT mette a disposizione del client dei *wildcard*, i quali permettono di connettersi simultaneamente a più topic; uno è rappresentato dal simbolo +, mentre l'altro dal simbolo #. Il primo è chiamato

single-level wildcard, mentre il secondo *multi-level* wildcard. Ci sono dei topic, a cui non si può pubblicare alcun messaggio, riservati allo stato del sistema e sono quei topic che cominciano per $SYS/.
Alcuni esempi di topic validi possono essere i seguenti:

1. *casa/luci/sala* - topic specifico per le luci della sala;
2. *casa/+/sala* - include tutti i topic dei dispositivi che fanno riferimento alla sala (luci comprese);
3. *casa/#* - include tutti i topic che fanno riferimento alla casa.

2.1.2 Connessione al broker

Come scritto nell'introduzione di questo capitolo, la connessione avviene solamente fra client e broker. Per client intendiamo un qualsiasi dispositivo il quale permette di gestire una connessione ad un broker, che si comporta come un server.

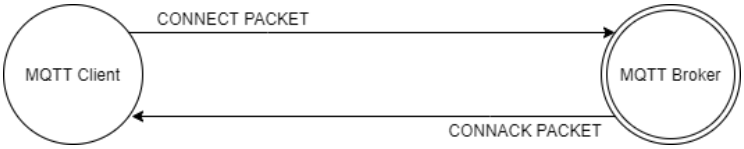

Figura 2.1. Flusso connessione al broker MQTT.

Come si può vedere dalla Figura 2.1, la connessione avviene tramite lo scambio di due pacchetti: *connect*, inviato dal client al broker, e *connack* inviato dal broker al client. Il pacchetto connect è così strutturato:

Tabella 2.1. Struttura pacchetto connect.

Parametro	Descrizione
clientId	rappresenta l'identificativo del client che chiede di connettersi
cleanSession	valore booleano il quale specifica se la connessione è persistente o meno
username	rappresenta l'username necessario affinché avvenga la connessione
password	la password associata all'username
willRetain	viene letto solo se *willFlag* è true
willQos	viene letto solo se *willFlag* è true
willFlag	permette di notificare un messaggio specificato dal client nel payload in caso di disconnessione anomala
keepAlive	rappresenta in secondi l'intervallo massimo in cui broker e client possono non inviarsi messaggi

Il pacchetto connack, invece, è così strutturato:

Tabella 2.2. Struttura pacchetto connack.

Parametro	Descrizione
sessionPresent	flag riferita al cleanSession del pacchetto connect
returnCode	stato della connessione

Per quanto riguarda il *returnCode* descritto nella Tabella 2.2, questo è un valore che va da 0 a 5:

- 0: connessione accettata;
- 1: connessione rifiutata, versione del protocollo non valida;
- 2: connessione rifiutata, clientId non valido;
- 3: connessione rifiutata, server non disponibile;
- 4: connessione rifiutata, username o password errati;
- 5: connessione rifiutata, non autorizzati.

2.1.3 Sottoscrizione e pubblicazione

Come scritto più volte, MQTT è un protocollo che adotta un meccanismo chiamato *publisher/subscribe*; perciò due aziono fondamentali che un client può compiere sono quelle chiamate *publish* e *subscribe*.

Subscribe

L'azione subscribe permette al client di sottoscriversi a un determinato topic passato nel pacchetto.

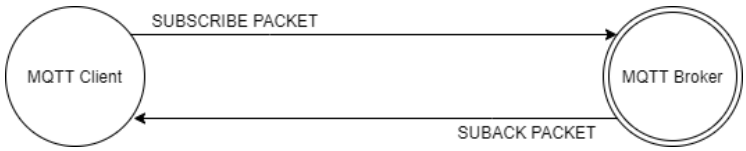

Figura 2.2. Flusso subscription ad un topic in MQTT.

Come si nota dalla Figura 2.2, la sottoscrizione avviene semplicemente con lo scambio di due pacchetti: *subscribe*, dal client al broker, e *suback*, dal broker al client. Il primo pacchetto rappresenta la richiesta di sottoscrizione ad un

determinato topic, il secondo invece conferma la sottoscrizione a quel topic. La struttura del subscribe packet è la seguente:

Tabella 2.3. Struttura pacchetto subscribe.

Parametro	Descrizione
packetId	l'identificatore del pacchetto
payload	una lista di QoS e topic a cui sottoscriversi

Il pacchetto suback, invece, è così strutturato:

Tabella 2.4. Struttura pacchetto subscribe.

Parametro	Descrizione
packetId	l'identificatore del pacchetto a cui rispondere
returnCode	lista di *returnCode* che rappresentano lo stato della sottoscrizione

Anche in questo caso il *returnCode* può assumere diversi valori, riassunti in questa lista:

1. 0: successo, massimo QoS 0;
2. 1: successo, massimo QoS 1;
3. 2: successo, massimo QoS 2;
4. 128: sottoscrizione fallita.

All'azione di sottoscrizione corrisponde anche un'azione di disiscrizione dal topic che avviene attraverso il pacchetto *unsubscribe*. Quest'ultimo è così strutturato:

Tabella 2.5. Struttura pacchetto unsubscribe.

Parametro	Descrizione
packetId	l'identificatore del pacchetto a cui rispondere
topics	lista di topic da cui disiscriversi

Publish

La pubblicazione di un messaggio in MQTT è più complicata rispetto alla fase di sottoscrizione al topic. Prima di tutto, bisogna introdurre il concetto di QoS, *Quality of Service*. Il QoS è paragonabile ad un contratto che viene stipulato tra client e broker; esistono tre diversi livelli: QoS 0, QoS 1, QoS 2.

Nel Quality of Service di livello 0 il payload del pacchetto viene pubblicato immediatamente. In questo caso non c'è garanzia sull'effettiva pubblicazione del messaggio.

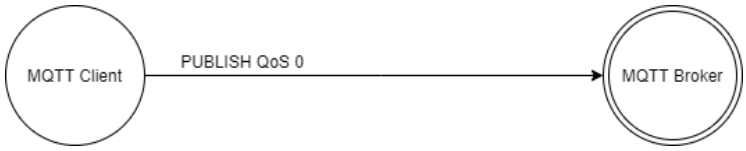

Figura 2.3. Flusso publish con QoS 0.

Nel caso del Quality of Service di livello 1 viene garantita la pubblicazione ad almeno un destinatario.

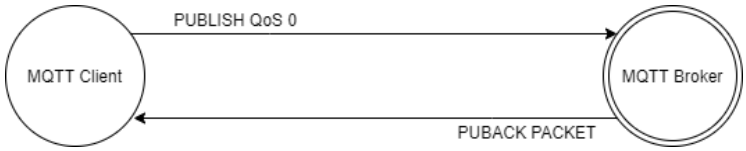

Figura 2.4. Flusso publish con QoS 1.

Come si vede dalla Figura 2.4, il client riceve dal broker il pacchetto *puback* solo dopo aver pubblicato il pacchetto publish. Il pacchetto puback è così strutturato:

Tabella 2.6. Struttura pacchetto puback.

Parametro	**Descrizione**
packetId	l'identificatore del pacchetto publish

Questo meccanismo permette al client di inviare nuovamente il pacchetto publish nel caso in cui, dopo un certo intervallo di tempo, non abbia ancora ricevuto il puback dal broker; in questo caso, nel pacchetto publish, il campo *dup* sarà impostato come true. Inoltre, il client mantiene in memoria il pacchetto publish finché non riceve il puback.

Infine abbiamo il QoS di livello 2 il quale dà maggiore affidabilità ma è più lento rispetto ai precedenti livelli.

Come si vede dalla Figura 2.5 abbiamo più messaggi scambiati fra client e broker. Questo meccanismo assicura che il messaggio sia ricevuto solamente una volta dai destinatari; il livello 2, perciò, dà un maggiore livello di sicurezza al client. Nel caso in cui il pacchetto vada perso, dopo un certo intervallo di tempo il client trasmetterà nuovamente il pacchetto e resterà in attesa della risposta da parte del broker.

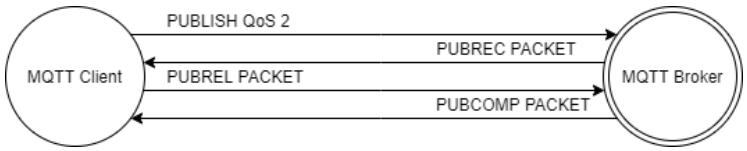

Figura 2.5. Flusso publish con QoS 2.

La struttura dei pacchetti è la stessa che si può vedere nella Tabella 2.6.

Adesso che sono stati analizzati i differenti QoS, si può analizzare nel dettaglio il publish, il pacchetto più complicato del protocollo. Il pacchetto ha i seguenti campi:

Tabella 2.7. Struttura pacchetto publish.

Parametro	**Descrizione**
packetId	l'identificatore del pacchetto
topic	il topic a cui si vuole pubblicare il messaggio
qos	il Quality of Service del pacchetto
retainFlag	booleana per il *retained*
payload	il contenuto del messaggio
dup	booleana che indica se il messaggio è un duplicato

Dopo aver ricevuto il pacchetto il broker processa il messaggio in base al QoS; una volta processato invia il pacchetto a tutti i client sottoscritti al topic.

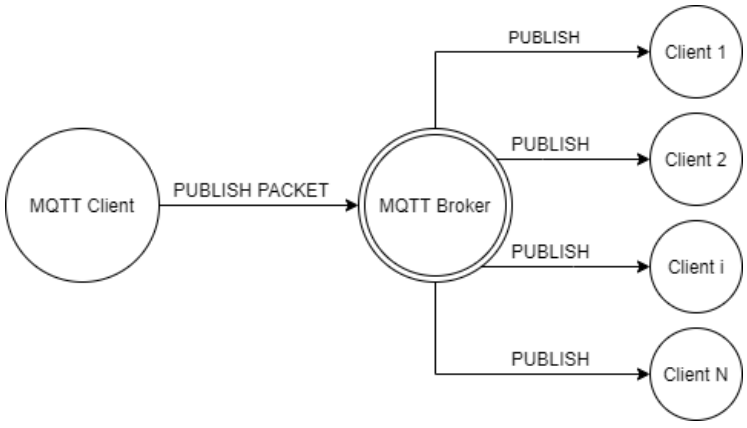

Figura 2.6. Esempio di pubblicazione di un pacchetto.

2.1.4 Altre caratteristiche

Retained message

Un *retained message* è semplicemente un pacchetto che ha impostato il valore true della *retainFlag*. Di solito il broker una volta che riceve un messaggio da pubblicare su un topic, in cui nessun client è iscritto, lo scarta; questo avviene quando il retain è *false*. Nel caso di un *retained message* il broker salva in memoria il pacchetto che ha ricevuto per poi pubblicarlo una volta che un client si sottoscrive al topic.

Last and will testament

Questo meccanismo, indicato anche con *LWT*, ci permette di gestire meglio le disconnessioni improvvise da parte di un client. Quando il dispositivo si connette al broker, nel pacchetto connect (Tabella 2.1), viene specificato il payload nel caso in cui avvenga una disconnessione imprevista; questo payload viene salvato dal broker il quale lo utilizzerà in caso di imprevisto.

Clean session o persisted session

Questa funzionalità viene specificata dal client nel momento della connessione; infatti nel pacchetto *connect* (Tabella 2.1) viene specificata una variabile *cleanSession*. Nel caso di una persisted session il client riceverà dal broker tutti i messaggi pubblicati sui topic a cui si era sottoscritto che sono stati inviati mentre era offline.

Capitolo 3

Implementazione ed esperimenti

Per ricercare possibili anomalie prodotte dalle librerie broker MQTT, c'è stata la necessità di implementare una libreria client la quale potesse permettere di gestire a basso livello i pacchetti e il loro flusso; infatti le librerie come *paho* per python e *mqtt.js* per nodejs non permettono di lavorare a basso livello.

3.1 La libreria client

La libreria client è stata scritta in python e utilizza la libreria *twisted* [7]. Sono stati implementati tutti i pacchetti che MQTT mette a disposizione, inclusi tutti i livelli QoS.
Un esempio di pacchetto costruito è quello publish; i vari campi sono descritti nella Tabella 2.7, qui possiamo vedere il *fixed header* del pacchetto.

Tabella 3.1. Fixed header publish packet.

bit	7	6	5	4	3	2	1	0
byte 1	Packet				Dup	QoS level		Retain
	0	0	1	1	x	x		x
byte 2	Remaining Length							

Oltre a questo header, abbiamo il *variable header* che contiene il nome del topic a cui si sta pubblicando e l'id del pacchetto. Infine c'è campo per il *payload*, ovvero il contenuto del messaggio da pubblicare.

Il codice python per la gestione di questo pacchetto è il seguente:

```
def publish(self, topic, message, dup=False, qos=0, retain=False,
                                    messageId=None):
    header = bytearray()
    varHeader = bytearray()
    payload = bytearray()

    header.append(0x03 << 4 | dup << 3 | qos << 1 | retain)
    varHeader.extend(_encodeString(topic.encode("utf-8")))

    if qos > 0:
      if messageId is None:
        varHeader.extend(_encodeValue(random.randing(1, 65535)))
      else:
        varHeader.extend(_encodeValue(messageId))

    payload.extend(_encodeString(message.encode("utf-8")))
    header.extend(_encodeLength(len(varHeader) + len(payload)))

    self.transport.write(header)
    self.transport.write(varHeader)
    self.transport.write(payload)
```

Come si può vedere, il codice rispecchia la costruzione del pacchetto come definito nell'OASIS standard [5]. Innanzitutto vengono definiti i tre differenti campi: header (fixed header), varHeader (variable header) e infine payload. Si può notare come il messageId, che è l'identificatore del pacchetto, venga utilizzato solamente nel caso in cui il qos sia diverso da 0; in questo caso, infatti, dopo il publish dovrebbe esserci un ulteriore scambio di pacchetti tra broker e client che coinvolge l'id del pacchetto.
Infine viene aggiunto il payload, codificato in utf-8, e viene esteso il fixed header aggiungendo la lunghezza totale tra varHeader e payload. Il trasporto vero e proprio del pacchetto dal client al broker, invece, è gestito grazie alla libreria twisted attraverso il metodo *write*.

In maniera del tutto analoga sono stati implementati tutti gli altri pacchetti. Attraverso questa libreria client si ha accesso alla costruzione del pacchetto, cosa non possibile nelle librerie come paho; ciò ha permesso di creare dei test particolari che hanno dato risultati differenti in base al broker utilizzato.

Per facilitare l'esecuzione dei test da eseguire, si è creato il programma in modo che potesse prendere dei file *json* i quali contenevano il flusso che la

libreria doveva seguire. Questo ha permesso di poter studiare in maniera più dettagliata i diversi comportamenti da parte dei server.
Un esempio di test è il seguente:

```
[
    {
       "type": "subscribe",
       "params": {
           "topic": "test/topic"
       }
    },

    {
        "type": "publish",
        "params": {
            "topic": "test/topic",
            "message": "pacchetto \#1",
            "qos": 0,
            "dup": false,
            "retain": false,
            "packetId": 1
        }
    }
]
```

In questo caso la libreria client gestiva due pacchetti: il primo di tipo subscribe al topic *test/topic*, il secondo, invece, di tipo publish al topic *test/topic* con QoS 0 e packetId 1. Ovviamente prima veniva inviato il pacchetto di tipo subscribe e solo successivamente quello di tipo publish.

3.2 I broker studiati

Sono stati presi come caso di studio differenti broker, che saranno introdotti molto brevemente in questa sezione. Ricapitolando, il broker ha la funzionalità di filtrare i messaggi che riceve dai differenti client e successivamente inviarli ai destinatari.

3.2.1 Mosquitto

Mosquitto [2] è uno fra i broker più famosi ed utilizzati per quanto riguarda MQTT. È leggero, open source e si adatta molto bene sia con dispositivi di poco consumo che con normali server. Supporta tutte le versioni del protocollo.

3.2.2 EMQ X

Anche EMQ X è un broker molto utilizzato. È scritto in *Erlang* ed è open source; permette di gestire milioni di connessioni simultanee anche con un unico server. Oltre a supportare MQTT, supporta anche CoAP, MQTT-SN (*MQTT for Sensors Networks*) e LwM2M.

3.2.3 HiveMQ Community Edition

HiveMQ Community Edition è un altro broker scritto in Java ed open source. Supporta MQTT a partire dalla versione 3.x fino all'ultima rilasciata che è MQTT 5. È molto utilizzato anche nei sistemi di automazione e in macchinari industriali che necessitano di un sistema real-time.

3.2.4 Moquette

Moquette [3] è un broker meno conosciuto scritto in java e open source. Supporta tutti i livelli di QoS e tutte le versioni del protocollo.

3.2.5 Aedes

Aedes [4] è un broker non troppo conosciuto scritto in NodeJS. Non supporta l'ultima versione di MQTT, ma è pienamente compatibile con la versione 3.1 e la 3.1.1. Ha molte librerie col quale può essere integrato.

Bibliografia

[1] Hron, M. Are smart homes vulnerable to hacking? https://blog.avast.com/mqtt-vulnerabilities-hacking-smart-homes (2018).

[2] Light, R. A. Mosquitto: server and client implementation of the mqtt protocol. *Journal of Open Source Software*, **2** (2017), 265. Available from: https://doi.org/10.21105/joss.00265, doi:10.21105/joss.00265.

[3] Moquette. Moquette. https://github.com/moquette-io/moquette (2020).

[4] MoscaJS. Aedes. https://github.com/moscajs/aedes (2020).

[5] Open, O. OASIS Open. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Table_3.3_- (2020).

[6] Wikipedia. Mirai (malware) — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Mirai%20(malware) (2020).

[7] Wikipedia. Twisted (software) — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Twisted%20(software) (2020).