FlyTrap: Decentralised Blockchain Security & Auditing Architecture for IoT and MQTT Brokers

Konrad M. Dryja

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Master in Science
of the
University of Aberdeen.

Department of Computing Science

2020

Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed:

Date: March 15, 2020

Abstract

An expansion of the title and contraction of the thesis.

Acknowledgements

Much stuff borrowed from elsewhere

Contents

Abreviations

AAA Authentication Authorization Accountability.

ACL Access Control List.

CCPA California Consumer Privacy Act.

GDPR General Data Protection Regulation.

IoT Internet of Things.

MQTT Message Queuing Telemetry Transport.

PII Personal Identifiable Information.

RFID Radio-Frequency Identification.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

Chapter 1

Introduction

1.1 Overview

1.1.1 Internet of Things

Internet of Things, also known as IoT, is a growing field within technical industries and computer science. It's a notion first first coined in Ashton [1] where the main focus was around RFID (radio-frequency identification) tags - which was a simple electromagnetic field usually created by small-factor devices in a form of a sticker capable of transferring static information, such as a bus timetable or URL of a website (e.g. attached to a poster promoting a company or an event). Ashton argued the concern of data consumption and collection being tied to human presence at all times. In order to mine information, human first was required to find relevant data source which then could be appropriately evaluated. But, as it was accurately pointed out, people have limited resources & time and their attention could not be focused constantly on data capture. Technologist suggested delegating the task to machines themselves; completely remove the people from the supply chain. A question was asked, whether "things" could collect data from start to finish. That paper is known to be the first mention of IoT and a building stone, de facto defining it as an interconnected system of devices communicating with each other without the need of manual intervention.

With time and ever expanding presence of smartphones, personal computers and intelligent devices, the capabilities of those simple RFID tags were also growing beyond just a simple static data transmission functionalities. Following the observation by Moore et al. [10], the size of integrated circuits was halving from year to year, allowing us to put more computational power on devices decreasing in size. They were now not only capable of acting as a beacon, but actively process the collected information (for example, temperature) and then pass it along to a more powerful computer which then could make decisions on whether to increase or decrease the strength of radiators at home - all without any input from the occupants. Eventually, IoT found their way to fields and areas such as households (smart thermostats or even smart kettles), physical security (smart motion sensors and cameras) or medicine (smart pacemakers). This number is expected only to grow in the future. Inside CISO white paper [6], scientists speculate that we might see 50 million of those devices by the end of 2020

1.1.2 Security of data

The growing presence of smart-devices significantly increased the convenience and capabilities of "smart-homes" - at the same time IoT also started handling more and more sensitive data -

especially considering the last example from the previous paragraph. Scientist from University of Massachusetts successfully performed an attack on a pacemaker [7], reconfiguring the functionality, which - if performed with malicious intents - could have tragic consequences. But even less extreme situations, such as temperature readings at home, are nowadays heavily regulated by data protection laws. Examples being the General Data Protection Regulation (GDPR) introduced by European Commission [5] or California Consumer Privacy Act by California State Legislature [3]. Collection of data is required to be strictly monitored and frequently audited in case of a breach - which also includes restrictions on collection of Personal Identifiable Information (PII, as per GDPR). Those and more put an obligation on every company willing to exchange user data to govern the data appropriately and ensure its security - which includes data collected by Internet of Things devices.

1.1.3 MQTT

IoT are usually low-power with limited computational power - mostly to decrease the required maintenance and ensure long-lasting life, without the need of replacing the power source (which is often a fixed battery) - meaning that only minimum amount of work should be performed on the "thing" itself, instead sending it off to a centralised structure (e.g., a server hosted on the cloud) for further processing. One of the popular choices includes an intermediary, a broker, relaying communication between clients connected to it. That way, Peer-to-Peer connection is not required and can be wholly delegated to separate backend server. Popular choice for the broker is MQTT (Message Queuing Telemetry Transport)[1] standard defining the exact shape and form of TCP packets, handling unexpected timeouts & reconnects along with distributing channels of communication onto different topics containing separated information. From there, clients can either subscribe (i.e. consume) or publish (which can also be used for issuing commands) the data. Unfortunately, the OASIS standard introduces limited security capabilities (offering only username/password authentication) and no auditing or logging.

1.1.4 FlyTrap

This project will be aiming to develop a novel approach - further referred as **FlyTrap** - for handling security in systems utilizing MQTT brokers and their implementations, focusing on platform-agnostic solution hosted within containerized environment. It will not depend on the exact software implementing the broker, but rather will aim to work with any broker that fully implements MQTT v5.0 standard. Furthermore, to ensure decentralised operation resistant to data breaches, downtime and full transparency, Ethereum[2] platform would be used as a data layer: capturing relevant interaction as publicly available transactions. In order to limit the quantity of data put on the blockchain (as computational and storage power there is limited), I will also introduce several rules dictating logging of only specific events. The system's purpose is to fully incorporate **Authentication, Authorization** and **Accountability** (AAA) framework to IoT devices communicating through MQTT.

[1] https://mqtt.org/
[2] https://ethereum.org/

1.2 Motivation

1.2.1 MQTT

MQTT v5.0 (as per the specification[3]) does not dictate nor specify any requirements regarding the security. It does offer an option of restricting some topics only to specific users, defined in access control lists (ACLs). The users then are required to provide a password when initiating a connection with the broker. Although, the basic username/password authentication is known to be cumbersome, only offering limited security. This also puts a burden on system administrators to maintain those ACLs in some centralised system, which then again is at risk of breaches or leakage. Moreover, placing the burden on a singular MQTT broker creates a single point of failure, where system downtime could halt the entire architecture.

1.2.2 Blockchain

By decentralising the data layer of the AAA framework and in process placing it on distributed ledger, I can ensure maximised uptime and complete transparency of performed transactions. Events such as permission changes, failed authentication attempts will be recorded as separate transaction which then could be audited by anyone knowing the public address of the system. This then could be handed over to authorities or auditing corporations to ensure that data is passed in a lawful manner. Utilising Blockchain technologies also opens an opportunity to require payment (in the form of crypto currency) from potential consumers of data effectively expanding the business model.

1.2.3 Legislature

The rise of awareness of necessity of data protection also encouraged governments to introduce legal requirements (such as GDPR or CCPA) of data governance and face heave fines in case of non-compliance. MQTT standard and their implementation at the moment would be considered non-compliant, due to effectively no way to trace past operations. General Data Protection Regulation requires entities handling user data to maintain proper retention of data and purge if requested by the data owner. MQTT at its current state is not capable of either, as messages are removed from the broker as soon as they are consumed (with small exceptions), leaving no trace of “who” accessed “what” (not to mention questions such as “why” they accessed it).

1.3 Goals

The project can be divided onto four main goals and two extras, leaving some field for maneuvering in case of road blocks or difficulties resulting from the challenges faced in the dissertation. By having flexible targets, I will be able to stop sooner in case of overestimating the schedule, or carrying on with extra work, should I find myself meeting the targets quicker than expected.

Main Goals:

1. Design structure of blockchain network, relevant data models that would be placed on the blockchain and deploy on the Ethereum platform, capable of recording transactions and allowing for modification of ACLs, i.e. which wallets are permitted to access specific resources on the MQTT brokers.

[3] https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

2. Design rules that would be used for capturing the transactions. For example, rule stating that if client makes more than 5 consecutive, failed authentication attempts would be placed on a blacklist and that action would be added onto the blockchain as a transaction.

3. Design containerised software acting as a secure proxy between brokers and connecting clients. This will handle both authentication and log performed action as an immutable transaction on a blockchain network. Logging would only be performed if the requested operation triggers some pre-defined rules.

4. Perform evaluation of the designed solution using an off the shelf MQTT broker and a range of experimental scenarios with simulated network of MQTT clients.

Extra Goals:

1. Create public API for the auditors to freely access the contents of blockchain and thus transactions containing information about suspicious operations.

2. Generalise the implementation of the framework so it can be deployed with any broker following the MQTT standard.

What project is NOT trying to be:

- Design a new blockchain platform from scratch. Rather existing solution - Ethereum - is going to be used.

- Write / modify operating system of IoT devices.

- Design a new MQTT Broker. The system is going to be built on-top of MQTT layer.

1.4 Report Structure

The dissertation is going to be divided onto 7 chapters, each describing following aspects of the project:

Chapter 1 **Introduction** chapter will outline the main motivation behind the project and introduce the notions used a building block in the design. It will also list goals and no-goals defining success.

Chapter 2 In **Background & Related Work**, similar research and state of the art will be described along with outlining the differences between them and this project. And thorough explanation of used software will also be attached, such as what is blockchain, Ethereum, MQTT.

Chapter 3 **Requirements & Architecture** will include analysis of both functional and non-functional requirements, main use-cases that are driving the project and high-level overview of the architecture explaining how each element addresses each of the requirements.

Chapter 4 **Design** will be an expansion to architecture, providing an explanation on how each of the elements connects to another.

Chapter 5 **Implementation** will talk about the process of implementation the design into software. It will include notions such as followed processes, used frameworks and sample code snippets.

Chapter 6 Inside **Testing & Evaluation** a comparison between state-of-the-art software, vanilla and FlyTrap will be performed. Tests checking for performance impact and whether common attack can be detected / stopped will also be run.

Chapter 7 **Discussion & Future Work** will include conclusions of the project, elements that were left-over, but beneficial for future iteration and all blockages encountered throughout.

Chapter 2

Background & Related Work

2.1 Background

In this section, I will list all technologies that are used in this project along with discussing other papers which were trying to address security with IoT devices by also trying to include blockchain technology.

2.1.1 MQTT

When designing architecture with the main target being communication of many (even couple of thousands a second) clients constantly exchanging data, scalability and availability needs to be kept in mind. The first and obvious solution would be to directly connect data consumers and data produces, by making them communicate in Peer-to-Peer fashion, removing the need for any extra infrastructure. This might work perfectly fine with small systems (disregarding issues such as dynamic DNS or static IP), but as number of clients requesting access to data increases, the total capacity of the sensor would eventually be capped - since IoT usually are of limited power and computation capacity. Imagine a scenario where a single temperature sensor constantly getting bombarded with requests for current readings, it might be able to cope up to 5 incoming requests every second, everything else would cause malfunction or significantly slower response times.

Then there is also an issue of security. By allowing clients to connect to our IoT devices, we are opening an extra attack vector. What if the client doesn't want to only access the temperature readings, but perhaps inject a worm which would intercept other sensors (such as cameras). Recently "smart nannies", responsible for alerting the parents when the child is crying and also relieving the adults from having to be constantly nearby, gained popularity. A direct camera feed could be accesses via smart phone, no matter where. This eventually led to exploitation, as it was found that many of those devices were vulnerable to remote access by third parties[11].

MQTT aims to address those issues (and not only), by moving the communication to a separate entity, which operates in a publish-subscribe fashion. This would mean that IoT devices only have to publish information that is available to them (e.g. temperature readings), allowing to completely remove remote access, effectively mitigating this particular attack vector. Furthermore, the MQTT brokers can be further placed behind load balancers and such to further enhance their availability.

In short, MQTT, fully expanded to Message Queueing Telemetry Transport is an open protocol, certified by OASIS and ISO[2], responsible for publisher-subscriber architecture. It's important to point out that MQTT is not a piece of software or a server, but rather a set of standards

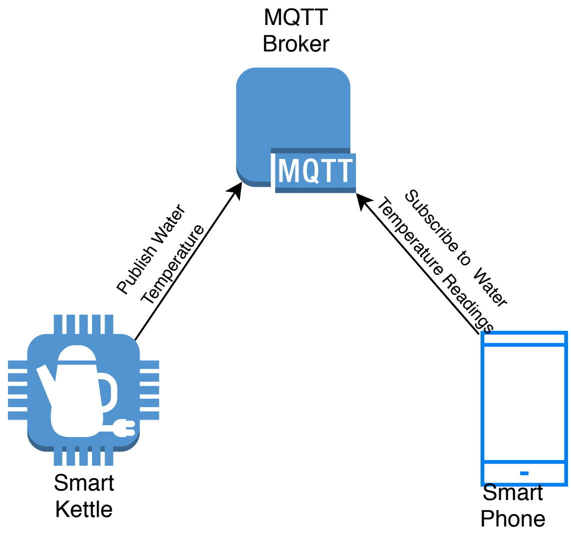

Figure 2.1: MQTT Broker Architecture

defining what potential clients can expect (what kind of responses and data) while connection to brokers following the standard. Figure 2.1 briefly shows how MQTT-compatible broker can relay information between clients. Smart Phone and Smart Kettle don't have to be online at the same time in order to receive information, nor Smart Phone is even permitted to initiate direct connection to Smart Kettle. The broker's responsibility is to track connected subscribers (which must specify topic of their interest) and maintain connection until subscribe advertises session termination or abruptly disconnects (e.g. loss of power or unreliable connection).

In MQTT architecture, Client ID identifies each of the connecting entities (publisher / subscriber) and Topic identifies a bridge between publishers and subscribers connected to the same topic. For example, a Smart Kettle could be publishing temperature readings under topic called "UK/Aberdeen/Kettle" - then, Smart Phone would need to request the same topic to receive those readings.

Message persistence

This will be discussed in depth, when I will describe the process of publishing and subscribing, but it's worth pointing out that by default, the messages are not saved nor cached on the broker. That is, if Kettle publishes the temperature reading, but there is no subscribers listening to this information, the message will perish. This is not ideal, for situation where smart device could wake up only every couple of minutes and then go to low-power mode again. To address this, MQTT Messages can be enriched by "Retain" flag. If such flag is present, the broker will keep the message and send it straight away to any new subscribers requesting given topic. This is also useful for issuing commands to IoT devices - for example, a phone could send command to turn off the lights with "Retain" flag set. Then, the smart light switch could check for retained messages every couple of minutes, removing the need of constant connection.

Implementation

MQTT by itself is only a collection of standards instructing implementors on what patterns should be followed and the structure of particular messages, thus it's not shipped with any piece of software. It assumes operation on TCP layer of network (although newer versions also allow for

WebSocket support [9]), thus also allowing for encrypted connection via Transport Layer Security. Every exchanged message is a TCP packet, following strict convection - which in case of deviation is discarded as corrupted.

Two of the implementations that I have considered during this project are Mosquitto[1] by Eclipse and Moquette[2]. The former written in C and the former in Java, although there is many, many more. In a paper by de Oliveira et al. [4], scientists compare Moquitto and RabbitMQ, arguining their choice by the offered cloud infrastructure with greater scalability opportunities. Moreover, there are solutions that are paid, whereas the considered approaches are free and open source allowing for better understanding of operations. The paper is concluded with the finding that hardware and network latency has a far greater impact on the performance, rather than choice of the individual broker, which leaves the decision mostly down to offered extra features.

Mosquitto also offers a Docker container [8] in which the broker can be run, allowing for further isolation and removal of extra dependencies.

Publishing

The most popular method of passing MQTT messages is still under Transport layer, as TCP packets. This allows for slightly higher freedom (compared to stricter protocols, such as HTTP), at the cost of more sophisticated parsing. MQTT standard is composed of several message types with the most important being:

- CONNECT - used to initiate the connection
- PUBLISH - used by the client to publish messages and by the broker to publish messages to subscribers
- SUBSCRIBE - used by the client to request subscription to a given topic
- UNSUBSCRIBE - used by the client to request removal of subscription to given topics
- Along with relevant *ACK counterparts (e.g. CONNACK) used to indicate successful transmission of the message

As shown on figure 2.2, the publishing flow starts with the CONNECT messages. Inside, there are several flags included, such as Quality of Service requested (MQTT can periodically send heartbeat ping to clients to check if they are still alive), requested version of MQTT protocol (at the moment, v5.0 and v3.1). This part is also referred as “Variable header”. The second part, known as “Payload” consists of client ID.

Then, once the client has established its identity to the broker, the broker responds with CONNACK message, which contains bit informing whether further connection is allowed or not. From this point, the client is cleared to start publishing session.

Usually, for every message to be published, there is one PUBLISH packet. Newer version of MQTT allow for spreading larger messages across multiple packets, although this will not be covered in this paper. The PUBLISH packet contains mostly two properties - topic to be published on and the actual payload. Each of the properties is prepended with 8 bytes indicating

[1] https://mosquitto.org/

[2] https://github.com/moquette-io/moquette

the length. From this fact, we can derive the maximum possible size of individual payload - 65535 characters (pure ASCII, no Unicode, which may take more than 1 bytes per character). Same as with CONNECT, each message is responded to with PUBACK, acting as a receipt for receiving the payload.

The client can continue to publish extra messages without having to connect again, as long as the TCP session has not been terminated. Should the client want to disconnect, it should follow standard TCP flow, i.e. issue FIN/ACK packet to the broker. For situation, where the connection has been terminated abruptly, there are options such as Will flag (message to pass in case of sudden disconnection) or Keep Alive (to indicate how long should the connection be kept alive for before assuming the client has lost connection).

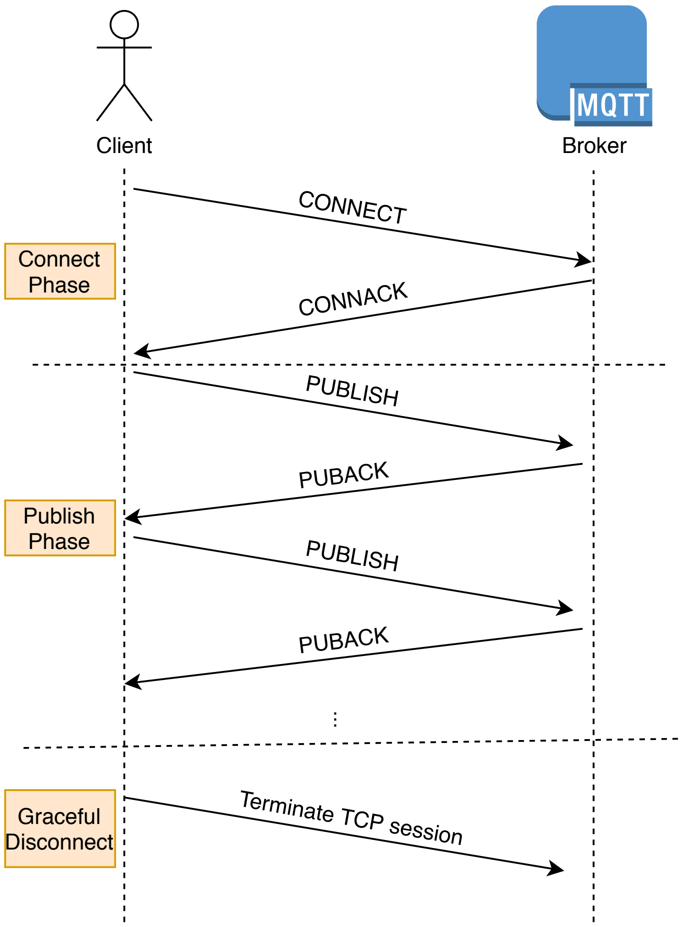

Figure 2.2: Publishing flow with MQTT

Subscribing

Subscribing flow is quite similar to Publishing, with some minor differences. Following figure 2.3, first and foremost a connection needs to be established by instating standard TCP/TLS session and then sending CONNECT packet. The contents follow the same standard, i.e. containing information such as Client ID or even optional parameters in a form of "key: value" (particularly useful for this project).

After successful connection, Client can proceed to send request for subscription. Similar with PUBLISH packet, client specifies type of the packet in the variable header and then requested topic

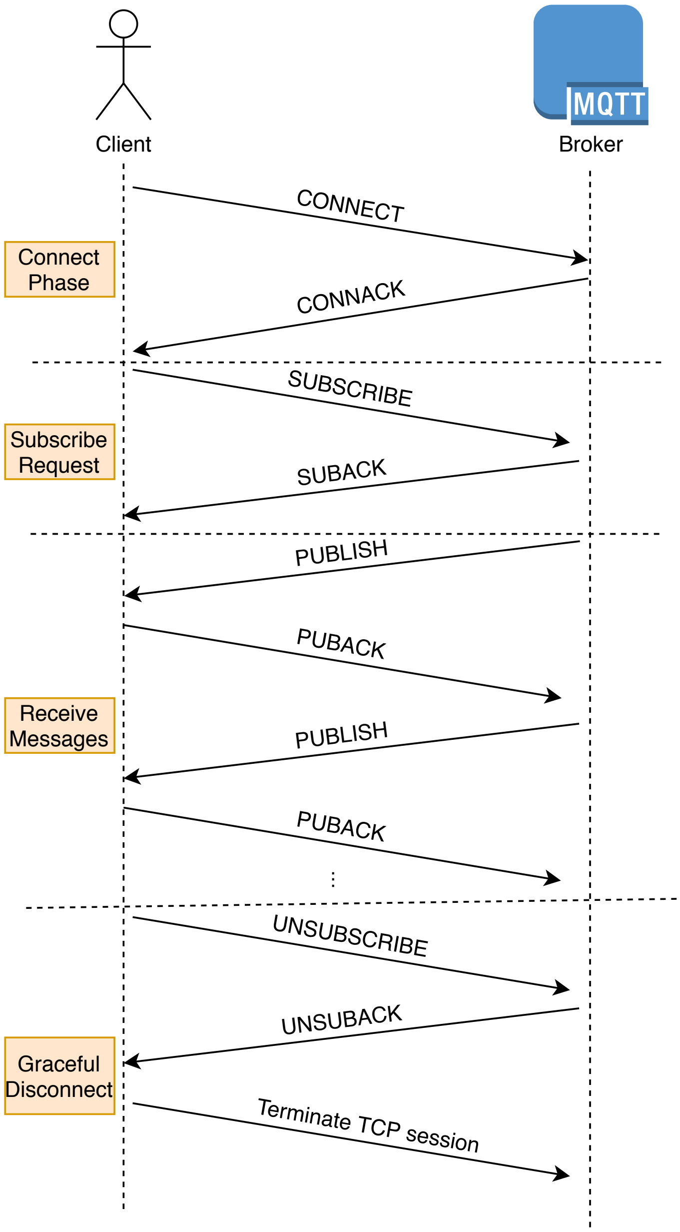

Figure 2.3: Subscribing flow with MQTT

for subscription in the payload. The extra element is the QOS flag - Quality of Service. MQTT has 3 levels of QOS:

1. 0 - No response to PUBLISH messages
2. 1 - PUBLISH messages will be followed by PUBACK
3. 2 - More granular control over PUBLISH, with extra packets such as PUBREC (Publish Received), PUBREL (Publish Release) and PUBCOMP (Publish Complete).

Once the SUBSCRIBE message has been processed and approved by the broker, it will issue SUBACK message and remain connect to the client. From this point, any message that is published on the topic specified in SUBSCRIBE packet will be published (as PUBLISH packet) to every client currently subscribed to it. Of course, depending on requested QOS, the broker might then await for PUBACK message (or even issue extra messages such as PUBREC, PUBREL, PUBCOM). The diagram demonstrates a simple exchange with QOS set to 1.

82	0C	00	01	00	07	F	L	Y	T	R	A	P	00
1	2	3	4	5	6	7	8	9	10	11	12	13	14

Table 2.1: Example SUBSCRIBE to topic FlyTrap packet

To close off MQTT, I also wanted to overview an example packet and dissect it byte by byte to demonstrate exactly what kind of information is included - this can be seen in table 2.1

1 Control field, specifies type of the message (CONNECT, SUBSCRIBE etc.)

2 Remaining length of the message. Can be expanded to 2 bytes.

3-4 Packet ID

5-6 Payload length

7-13 Payload. Corresponding hex encoding of characters, replaced with actual characters for clarity

14 Requested QOS

2.2 Related Work

This section will talk about other scientific papers which had similar goal in mind, by combining blockchain technologies with IoT or even MQTT brokers.

Chapter 3

Requirements & Architecture

In this chapter I will outline base requirements for the project along with sample stories that would later dictate the workflow. In the second part, I will also include overview of the architecture proposed for the system, correlating each element with relevant requirement and explaining how they would address the use-cases.

3.1 Requirements

3.1.1 User stories

1. As a government regulator, I'd like to overview access history to specific MQTT topics, to make sure the data is handled in GDPR-compliant manner.

2. As a government regulator, I'd like to verify why / when / who accessed given resource at a specific time, such that I can issue fines for potential non-compliance and inspect data breaches.

3. As a topic owner, I'd like to restrict people that can publish / subscribe to them, to maintain their confidentiality.

4. As a topic owner, I'd like to collect payments from people willing to access my data.

5. As a topic owner, I'd like to block access to my information from requests coming outside requested country, to comply with GDPR requirements.

6. As a broker owner, I'd like to collect payments from people willing to publish their data on my system, to keep the system profitable.

7. As a broker owner, I'd like to secure a distributed network of brokers (with varied implementations), to increase system's availability.

8. As a broker owner, I'd like to block access to the system to malicious clients performing denial-of-service attacks, to avoid system downtime.

9. As a data consumer, I'd like to publish/subscribe my messages on low-power devices, such that I can utilise my IoT sensors.

10. As a data consumer, I'd like to access the broker from over a hundred parallel sensors, each publishing data independently.

3.1.2 Functional Requirements

The user stories can then be further formulated into the following functional requirements:

(FR1) The system will provide an interface to manage access to the topics along with inspecting the audit trails.

(FR2) The system can connect to any Ethereum node, be it a public endpoint or a locally running, closed network. This will provide flexibility of either using transparent and with 100% uptime resource or a closed node with reduced costs.

(FR3) The system should provide a way to collect payments in ETH from clients attempting to gain access to relevant resources. This payment would then in process be transferred to the resource owner's Ethereum wallet.

(FR4) The system should offer an option to specify an exact amount of ETH required to publish or subscribe - with possibility of separating the costs and also setting the cost to 0 (=free).

(FR5) The system should be capable of fending of primitive denial-of-service attacks by blocking continuous, failed attempts to connect.

(FR6) The operations performed by clients will be of limited complexity, such that they can be executed on devices with limited computational power.

(FR7) The system can answer crucial GDPR questions, such as who accessed given resource, why did they have access, when they accessed it and what exactly was accessed.

(FR8) The system should offer an option to restrict the client's country that can access the resource, which the would be verified using GeoIP lookup, as various countries have various data protection laws.

3.1.3 Non-functional Requirements

In addition to the functional, it is also important to mention following non-functional requirements, as system is intended for end-users (potentially non-technical) and due to incorporation with blockchain can introduce performance overhead.

(NFR1) The system should provide **an overhead of no more than 2 seconds cumulative** per MQTT session. This is important, as the intention is to provide an add-on on top of the existing MQTT brokers. This might further compromise the current efficiency, so the system should aim to minimise the added latency

(NFR2) The system should be agnostic of the used MQTT broker, as long as the broker **fully implements MQTT v5.0 standard**. As pointed out earlier, there is a variety of brokers available to use, such as Mosquitto or Moquette. FlyTrap shouldn't rely on the implementation of a broker, but rather only on the standard utilised.

(NFR3) The system should be capable of extending any MQTT broker with **Authentication, Authorization, Accountability** framework. This is to ensure that data can only be accessed by authenticated entities, which are authorized to access requested resources - and in case of a breach or other disaster, keep them accountable to their actions.

(NFR4) The system should only be based on **Free and Open-Source Software**. Since the ultimate aim is to provide increase security, keeping the source open would allow any potential users to inspect its operation. Furthermore, third party security audits can happen without the system owner's intervention.

(NFR5) The system should be capable to run inside **virtualised container**, to ensure that it's platform agnostic.

3.2 Architecture

Chapter 4

Design

4.1 Secure Proxy

In order to enable FlyTrap to make decisions on whether the requests for publishing or subscribing should be accepted or denied, a secure proxy needs to be established between the clients and the MQTT Broker. As the communication between the broker and the consumers happens on Transport Control Layer, it is possible to insert a middleman which would be capable of inspecting the packets flowing through, dissecting it for relevant information and finally make a decision about their future journey - all without the client ever knowing that someone has intercepted the connection.

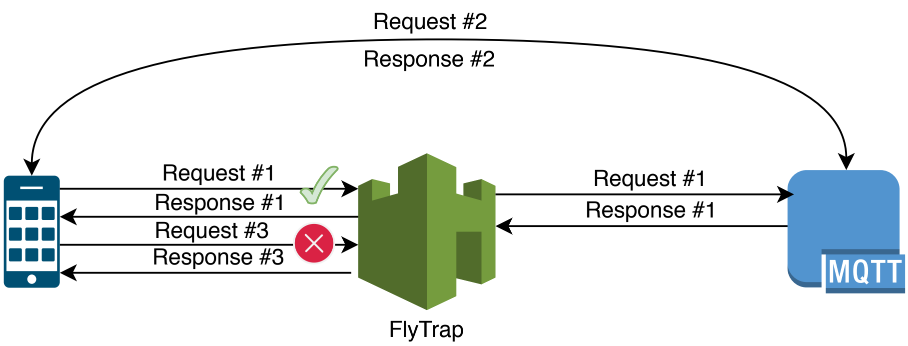

Figure 4.1: FlyTrap acting as a proxy

Figure 4.1 demonstrates all 3 possibilities when client attempts connection to a broker. In the Request #1, FlyTrap will dissect the packet and confirm that the phone indeed can be allowed to access specific topic and then start bidirectional proxy with the broker, passing the TCP packets between two. Request #2 shows that same packet can be used for vanilla MQTT Broker without FlyTrap, thus decoupling the client and secure proxy, as the former can be used without the need to change the latter. Finally, for the third request, it is found that the client cannot access the requested resource and will be presented with CONACK response, with access denied flag set, terminating the connection.

Although this solution enough will not be sufficient. As easily as FlyTrap can tap into the connection, same can be assumed for potential malicious actors, which could be listening on the flowing through packets. The solution will support an extension to standard TCP - Transport Layer Security, or TLS for short, responsible for encrypting the TCP packets, greatly reducing the threat of man-in-the-middle attacks.

TLS sessions can be summarized in the following steps:

1. Initiate standard TCP session
2. ClientHello with client's ciper capabilities
3. ServerHello and exchange of the cipher suite, along with server's certificate
4. Key exchange and change of cipher spec
5. Encrypted session starts

It's important to point out, that due to step 3 requiring server's certificate, FlyTrap will need to either obtain copy of broker's certificates or generate a new pair, ensuring that the connecting client's will trust it.

Chapter 5

Implementation

Bibliography

[1] Ashton, K. (1999). An introduction to the internet of things (iot). *RFID Journal*.

[2] Banks, A., Briggs, E., Borgendale, K., and Gupta, R. (2019). Mqtt version 5.0. *OASIS Standard*.

[3] California State Legislature (2018). Ab-375 privacy: personal information: businesses.

[4] de Oliveira, D. L., Veloso, A. F. d. S., Sobral, J. V., Rabêlo, R. A., Rodrigues, J. J., and Solic, P. (2019). Performance evaluation of mqtt brokers in the internet of things for smart cities. In *2019 4th International Conference on Smart and Sustainable Technologies (SpliTech)*, pages 1–6. IEEE.

[5] European Commission (2018). 2018 reform of eu data protection rules.

[6] Evans, D. (2011). The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper*, 1(2011):1–11.

[7] Halperin, D., Heydt-Benjamin, T. S., Ransford, B., Clark, S. S., Defend, B., Morgan, W., Fu, K., Kohno, T., and Maisel, W. H. (2008). Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 129–142.

[8] Light, R. (2017). Mosquitto: server and client implementation of the mqtt protocol. *Journal of Open Source Software*, 2(13):265.

[9] Mijovic, S., Shehu, E., and Buratti, C. (2016). Comparing application layer protocols for the internet of things via experimentation. In *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, pages 1–5. IEEE.

[10] Moore, G. E. et al. (1965). Cramming more components onto integrated circuits.

[11] Pultarova, T. (2016). Webcam hack shows vulnerability of connected devices. *Engineering & Technology*, 11(11):10–10.