Is MQTT secure?

Teserakt AG

March 2019

When someone asks if something is secure it is not always easy to answer. People want a yes or a no even if they are incapable of explaining what "secure" means in the context. The last thing they want to hear is "it depends" – yet that's often the least wrong answer.

This post elaborates on what "it depends" involves in the context of MQTT's security, with a focus on confidentiality. Although we will focus on MQTT here, the same ideas apply to any protocol or communication between devices. Let's start with some general background on the notion of security for online services.

"Secure"?

Imagine if a group of people are communicating using a social network or chat program, how then might we define security? There are a number of ways to look at this problem:

1. **Security of the service itself**. Does the social network or chat server have any logic errors or exploitable code that might allow an unauthenticated user, or authenticated user not part of the group, to interfere with the messages of the group?
2. **Transport security to the service**. Most internet packets take multiple hops between a user and the destination server, whether inside the destination social network's network or outside of it. Can we trust these parties not to interfere with packets? Might they record them? Might other parties insert themselves into this path and do the same?
3. **End-to-end security**. This ensures confidentiality and integrity between the communicating participants such that no intermediaries, even the social network or chat server, can view or tamper with communications. All they know are who is communicating with whom.
4. **Sender and recipient untraceability**. This is where we do not know who is sending to whom at all, a.k.a. anonymity. We list this for completeness, but it's hard to implement effectively and for this reason rarely guaranteed. The obvious real world example of this is Tor.

So how do common service providers fare when considering these security criteria?

Most services provide transport security through the use of **TLS** (you might know this as SSL – TLS is the protocol that has replaced SSL but the names are often used interchangeably). Whether you upload a photo on Facebook or Google services, or chat using Slack, you are doing this via TLS and subject to TLS being secure, so are your communications. Since the mass surveillance revelations by Edward Snowden from 2013 onwards, there has been an increasing push to deploy encryption and in particular TLS: Chrome now marks unencrypted sites as insecure, there's Let's Encrypt, "end-to-end encryption" has a Wikipedia page; Snowden made crypto jump from subculture to mainstream, which is probably good, all things considered.

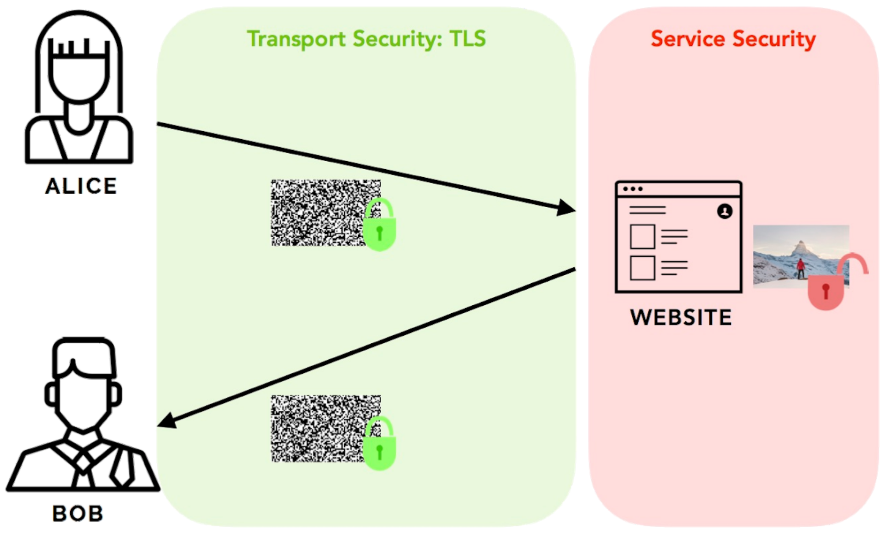
The security of online service providers is, however, never a total guarantee (although we know they employ word class engineers and teams to guarantee the best level of protection they can). Things are even worse when you move away from companies that understand the internet. Most penetration testers can tell you horror stories of out of date software, poorly written and possibly outright insecure code, lack of hardening and no process to manage this for critical production software.

TLS' limitations

Transport security is a great goal, but it cannot protect you if the security of the service in question is compromised, or if some of the service's operators decide to look into your data. So point 2 (transport security) does not mitigate point 1 (service security). Specifically, suppose we have the following sequence of events:

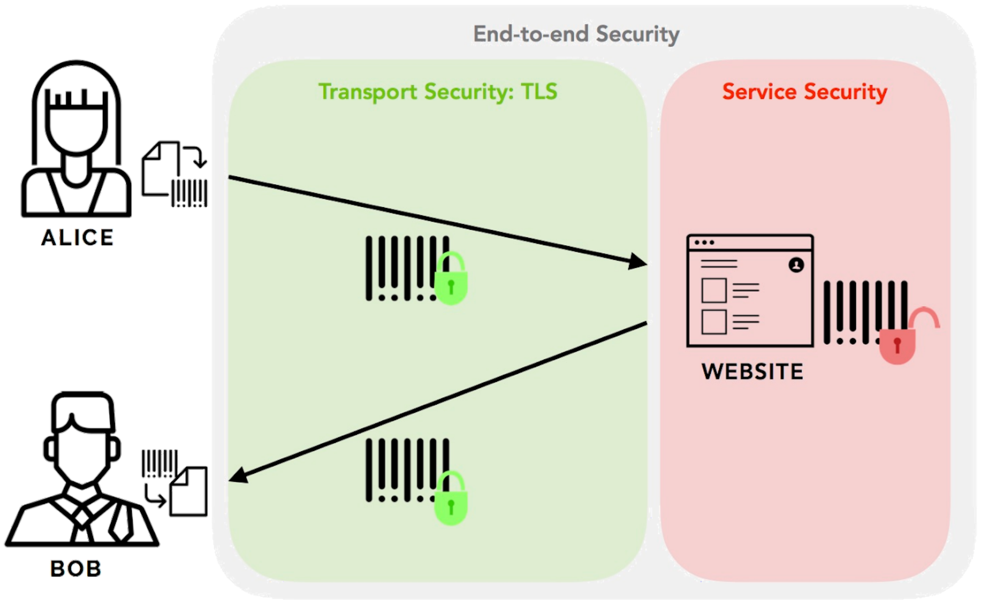
1. User Alice uploads a photo to a social networking website. Her browser negotiates TLS with the service and her photo is secured in transit.
2. The service stores the photo unencrypted in one of their data centres.
3. User Bob wishes to view the photo Alice just posted – perhaps it was a skiing trip to Verbier? He requests the photo from the service and being authorized by Alice already, he receives the photo. His browser likewise successfully negotiates and uses TLS.

We can visualize this in the following diagram:



The communication from Alice to the service and the service to Bob is secure, but the service itself has an unencrypted copy of Alice's photo. So if a logic error exists, or the service is otherwise compromised, that photo can be extracted by the attacker.

End-to-end security comes in to play at this stage: leaving out the details, if Alice encrypts a photo for Bob, then the service in question can no longer view the photo. They can only disrupt its transmission, as the diagram below shows:



The machine-to-machine case

In the most popular machine-to-machine (M2M) protocol, **MQTT**, messages are sent to a server, called the **broker**, before being forwarded to their recipient(s). Most broker software support TLS, so you can get transport security today. This is excellent news and ensures that messages cannot be read by intermediate internet nodes.

What TLS does not protect against is, of course, the **compromise of the broker**. This could happen for a whole number of reasons, from outright logic errors in the software to misconfiguration by the user. For example, a vulnerability such as the one we found in a popular broker software may be leveraged to get access to the host running the broker, thereby granting the attacker's access to all MQTT messages processed by the broker.

As stated above, TLS also leaves data exposed to operators of the broker, whom may not be trustworthy enough for critical applications – for example, if access to a confidential data stream is worth $1M to a competitor, said competitor could bribe or blackmail an operator for a fraction of this cost. Corporate espionage is a very real risk for many companies today and has never been easier. Companies that manage critical national infrastructure are particularly at risk.

Even worse, some devices rely on publicly managed brokers. When sharing a broker with other service providers in this way, while all information may be secure when talking to the broker, the only thing preventing other clients accessing your devices' information is the access control implementation on the broker. If such a public broker is compromised or implements access control poorly, those messages can easily be read and modified. As we've recently observed, access control are not always safe by default, which increases the risk.

Increasingly, vehicles are becoming connected devices too. The data shared by vehicles might range from engine performance information that is of interest to competing manufacturers, to personal information such as owner location and the journeys they make. This information almost certainly comes under the purview of ever stringent data protection law (for example the GDPR).

This is why there is a need for end-to-end security in the **M2M, V2X, and IoT** spaces. As increasingly amounts of sensitive data are sent between autonomous nodes, such as vehicle locations or presence in private homes, it becomes even more important to ensure that even in the event of relay, proxy or broker compromise, only the intended recipients of those messages can read the data.

Intermission – MQTT & TLS

As far as confidentiality is concerned, MQTT only briefly mentions using TLS. The standard says: "The MQTT protocol requires an underlying transport that provides an ordered, lossless, stream of bytes from the Client to Server and Server to Client.", and then states that, although the underlying transport is TCP/IP, TLS and WebSocket are "also suitable".

In the MQTT standard's Security section, we read that "it is strongly recommended that Server implementations that offer TLS SHOULD use TCP port 8883". However, the standard does not recommend – let alone strongly – that TLS be used. Instead, the same section notes that "[as] a transport protocol, MQTT is concerned only with message transmission and it is the implementer's responsibility to provide appropriate security features. This is commonly achieved by using TLS." Fair enough.

(The above excerpts are taken from MQTT 3.1.1's specification, but the latest one, 5.0, includes almost identical text.)

Cryptography of things

Implementing state-of-the-art cryptography in resource-constrained environments can present a technical challenge. In some constrained environments, the device may not be powerful enough to run public-key cryptography and/or to include a TLS implementation. Such environments can for example include some RFID chips or 8-bit AVR processors. But most of the time you'll have symmetric cryptography algorithms such as AES and

SHA-256. Yet crafting a protection scheme compatible with the system constraints isn't always straightforward – for example, how do you add authentication when you can't extend the payload size?

In more extreme cases – legacy hardware, high throughput constraints, and so on – standard crypto may still be insufficient. For such cases, NIST is running a competition that aims to standardize so-called **lightweight** cryptography algorithms. Note that the niche field of lightweight cryptography is all but new, and that many lightweight crypto schemes already exist; they're just not blessed by NIST. (Fun fact: the MSc and PhD theses of a Teserakt founder were about lightweight crypto, back in 2006 and 2009, before it was cool!)

Protecting messages with cryptography is generally a solved problem, and in our experience the trickiest part is the case-specific engineering problem. Specifically, a non-trivial problem is to create a system that remains safe even if the device is partially compromised, if its pseudorandom generator fails, or if its clock is not accurate, yet consumes minimal energy and does not incur significant performance hits.

The hard part

In the same way that you can't solve the halting problem, or that you can't generate randomness deterministically, you **can't create trust with cryptography**, no matter how much cryptography you use – to cite Jon Callas. In machine-to-machine networks, trust often simply consists in the association between a device identity and a cryptographic key. From there, we've got two main classes of problems related to **key management**:

First, problems that relates to **key provisioning and trust**: How do unique keys end up on devices? At what stage of the production chains are they generated and provisioned to the device? Can a device generate its own keys? How would they share their public or secret keys with other parties? What trust model should be used, pre-shared keys, trust-on-first-use, centralized and hierarchical PKI, or P3KI's decentralized model? There are no generic good answers to these questions – again, it depends.

Second, how do you encrypt, or more generally, **how do you protect messages** once every devices has its key? How are session keys derived from identity keys? Should there be a notion of session at all? How can you ensure that previous communications won't be compromised if a key is compromised later (forward secrecy)? What about the opposite (backward secrecy – if a current key is compromised can we recover and protect future messages)? How do you realize secure group messaging among large sets of devices, as opposed to humans? How can you replace humans' operations, and how can you leverage the absence of humans to better automate things? These questions and many others are mostly unexplored territory, and are our focus at Teserakt. We don't claim to have all the right answers (we definitely don't), but are doing our best to find the right trade-offs to provide the highest security at the lowest cost.

Is MQTT secure?

After reading the above you'll now realize that the question is not the right one to ask. MQTT is not insecure, but it's not secure, because it's **not designed to be secure**. A slightly better question to ask could be "Are MQTT broker services secure?", but then again it depends. A better question would be "Are MQTT broker services, and their related software, secure enough for application X?".

An even better question would be "Do you have enough trust in the broker software, in the organization that runs it and its personnel, in said organization's ability to perform security updates (not only of the broker but also of related software, such as hypervisors) to leave all your messages exposed to espionage and tampering today and a few years from now?"

Many organizations will acquiesce, and rightfully so, noting that they care more about availability than confidentiality and integrity. That could for example be the case of weather measurement data.

But many other organizations will reply that no, they think that risk is too high to be neglected or accepted. This might be the right answer when data transmitted relate to private individuals' activity, to critical infrastructure systems, to safety monitoring systems, to proprietary technology (which may be reverse engineered only from metadata), to software updates, to geolocation information, and many other types of information that needs be protected to ensure a sustainable business.

The future

We would like to think the internet was always ready for e-commerce and that the cryptography and security of systems protecting it have always been fit for purpose. However, the truth is that our understanding of secure systems and our expectations have evolved over time. SSL Version 1 never made it outside of Netscape and SSH version 1 is considered badly broken. We have iterated these protocols over the last several decades to arrive at today's level of transport security.

Security in personal messaging has likewise evolved from obscure, niche origins (such as OTR over what was originally called Jabber). The first attempt to bring end-to-end security to this space was TextSecure and RedPhone around 2012, a time in which all other messaging applications relied only on transport security. These applications were merged into Signal. Following the success of Signal, the protocol was licensed by and integrated into WhatsApp – among others – and is now used daily by around 1.5 billion people. Alternative products such as Wire, Telegram, and Facebook Messenger either launched with similar security standards, or rolled them out on their platforms.

Teserakt's vision is that the **same evolution is happening** in the embedded, IoT, V2X, and M2M spaces. Today, messaging protocols such as MQTT and Kafka are at best providing transport security. We want the state of the art of end-to-end-security to apply to the smart home, to critical national infrastructure, to the cars you drive, and to the medical devices you use.

Solving the hard problem with E4

If you need end-to-end security, you should contact us to try our product, E4, which we believe is the best option today for integrating solid encryption and key management in IoT systems. We have solved the hard problems so that you don't have to.