An Assessment Platform of Cybersecurity Attacks against the MQTT Protocol using SIEM

Mohamed Hadded, Gaspard Lauras, Jérôme Letailleur, Yohann Petiot, Anouk Dubois

An Assessment Platform of Cybersecurity Attacks against the MQTT Protocol using SIEM

Mohamed Hadded, Gaspard Lauras, Jérôme Letailleur, Yohann Petiot and Anouk Dubois

IRT SystemX, 2 Bd Thomas Gobert, 91120 Palaiseau, France

{mohamed.elhadad, firstname.surname}@irt-systemx.fr

***Abstract*—The industry of shared self-driving is increasingly interested in the Message Queuing Telemetry Transport (MQTT) solution to develop and evaluate their autonomous and shared mobility services. This solution would have the advantage of making data exchange easier between autonomous vehicles themselves and between vehicles and infrastructure. Nevertheless, there are a number of security threats due to the design and the use of the MQTT protocol. Some of these threats are denial of service (DoS), spoofing, information disclosure and data corruption. These security issues can be caused by external attackers as well as internal entities that are successfully authenticated. This paper analyzes the impact of these attacks on the performance of MQTT protocol with TLS in terms of CPU/RAM usage and latency. For that, we provide in this paper an in-depth overview of cybersecurity attacks that can disrupt the MQTT protocol and we also present an evaluation platform using Security Information and Event Management (SIEM) architecture that automatically collects and aggregates system events from the server to assess the impact of the cyber attacks. The results indicate that these attacks have highly negative influence on the performance of broker. These results will contribute in the future to implement new countermeasures to improve cybersecurity of MQTT protocol.**

Autonomous and shared mobility services, IoT, Cybersecurity, Attacks, MQTT, Broker, Pub/Sub, TLS, SIEM.

I. Introduction and motivation

In the context of connected driving, the reliable and secure transfer of a large quantity of data between the connected cars and the infrastructure (edge, cloud platform, traffic supervision center, etc) is a main challenge. New generations of cellular technologies meet the needs of connected and autonomous driving in terms of bandwidth and low latency [1]. However, at the application level, it is necessary to consider a secure messaging technology to allow the various actors (connected vehicles, road infrastructure, service providers, edge and core infrastructure, etc) to publish and consume data with very low latency. The Message Queuing Telemetry Transport (MQTT) protocol is currently considered to be the ideal messaging solution for Vehicle to Everything (V2X) communication over 4G/5G cellular networks [2]–[5].

The MQTTs publish/subscribe protocol is positioned at the application layer of TCP/IP stack. Being a protocol adopted initially in Internet of Thing (IoT), it has also become recently suited for sharing information between connected vehicles and infrastructure [2]. Many car manufactures have attempted to implement a connected car service using MQTT. For instance, BMW Mobility Services has announced their use of MQTT as the messaging platform, especially HiveMQ, to provide reliable messaging for their car-sharing application [6]. Therefore, communications security is a critical and delicate aspect, since the transmitted and processed information are related to real-time and sensitive context.

Nevertheless, there are a number of security threats due to the design and the use of the MQTT protocol [7]. Some of these threats are denial of service (DoS) [8], [11], spoofing, information disclosure, escalation of privilege and data corruption. These security issues can be caused by external attackers as well as internal entities that are properly authenticated [9]. Indeed, with valid authentication information but without any authorization to access a topic, an attacker can flood the broker with unauthorized sub/pub requests which lead to a significant consumption of the broker's CPU resources during the verification of these requests [11]. The detection of these attacks is therefore a crucial element to guarantee the security of the MQTT protocol against possible malicious acts, and in particular to allow the rapid implementation of countermeasures. The goal of this paper is first to provide an overview related to cybersecurity attacks that can disrupt the MQTT protocol and we also discuss their impacts. Moreover, we present an evaluation platform using Security Information and Event Management (SIEM) architecture to collect and aggregate the system event logs and assess the impact of these attacks on the performance of MQTT broker (e.g. Mosquitto). The main contributions of our paper are:

- We cover cyber-attacks that can disrupt the MQTT protocol and compare them in terms of vulnerabilities exploited by attackers, sending methods, etc.
- We present an evaluation platform using Security Information and Event Management (SIEM) architecture that automatically collects and aggregates system events from the server.
- Finally, we implement two DoS attack scenarios and evaluate their impacts on broker's performance in terms of memory and CPU computing resources as well as the latency.

The remainder of this paper is organized as follows: Section II presents an overview of MQTT protocol. In Section III, we describe the security vulnerabilities that can affect the performance of MQTT broker. Then, in Section IV, we present

the impact assessment platform. In section V, we give the experimental results and analyze the impact of SlowTT attack and last will payload flooding attack on the performance of MQTT broker. Finally, in Section VI, we conclude the paper and highlight some future work.

II. Overview of Message Queuing Telemetry Transport (MQTT)

MQTT is a lightweight M2M communication (machine to machine) protocol implementing a pub/sub messaging model of publishing messages and subscribing to topics. In MQTT, the central point is the MQTT broker in charge of relaying messages from publishers (senders) to subscribers (destination clients). If a device wants to publish a message, it sends it to the broker under a specific topic which describes the content of the message (for example: the weather, temperature, home, etc). The broker then transmits this message to each client that has previously subscribed to this topic.

The MQTT supports three QoS levels which define the effort that is made to ensure that the data reaches the broker.

- QoS 0 "At most once": the message is sent only once. Once the message is sent by the client it is rejected from the outbound message queue and therefore it is not re-transmitted by the sender. In addition, the receiver does not confirm receipt of the message.
- QoS 1 "At least once": the message is sent at least one time to the receiver until delivery is guaranteed. It might be delivered multiple times if the sender does not receive an acknowledgment after a period of time.
- QoS 2 "Exactly once". Each message is guaranteed to be delivered only once by the subscriber. This level provides the safest and slowest quality of service level.

The use of one level or another depends on the characteristics and reliability needs of the system. Logically, a higher level of QoS requires a higher exchange of verification messages with the client and, therefore, a higher load on the system.

The MQTT protocol has various security mechanisms that can be adopted to protect communications [20]. This includes, in particular, SSL/TLS protocols, authentication by user/password or by certificate and Access Control List (ACL). Since the MQTT protocol does not provide security on its own, its specification recommends TLS as a transport option to secure the protocol using port 8883. The devices with low latency and resource constraints, especially in industrial IoT deployments, can benefit from using TLS session resumption to reduce the re-connection cost. It is also possible to configure the broker to accept anonymous connections. All of these features must be considered when setting up an MQTT system, and the risks of each must be understood, as well as their impact on the effectiveness of the system.

III. DoS attack models in MQTT

In this section, we present the different DoS attacks that can disrupt the MQTT protocol. Generally, the main goal of DoS attacks is to overwhelm broker resources and thus to deny access to legitimate clients. DoS attacks accomplish this by forcing the victim broker to perform complex computing operations making it inaccessible to its intended clients. Several DoS attack models can target this protocol which exploit network configuration vulnerabilities of the MQTT protocol and its different access levels available for clients (QoS 0, QoS 1, QoS 2) as well as the different control packet types available in MQTT (CONNECT, PUBLISH, SUBSCRIBE, etc).

A. Last will payload flooding attack

This attack can be easily launched and it exploits the Last Will and Testament feature available in MQTT protocol [12]. This feature is used to notify other subscribers about an unexpected disconnected client. The last will message is set by the publisher client when it connects to the broker. It is stored by the broker until publishing client disconnects ungracefully. If the client is abruptly disconnected, the broker sends the message on behalf of the publisher to all its clients that are subscribed to the topic which was specified in the last will message. The last will payload flooding attack consists so in sending a massive number of CONNECT messages to the broker with a greater size of the last will payload (more than 10000 characters) in order to increase the size of the packets, which will have a strong impact on the bandwidth at the victim broker. Additionally, it will also increase the resources needed to process messages, which may causes the overall RAM and CPU usage to be higher than normal. This attack can have very serious consequences, if the attackers connect and disconnect frequently from the broker. This will force the broker to publish the message and so increase more the impact of the attack on server resources, preventing it from processing new TCP connections.

B. Unauthorized subscription flooding attack

It is DoS attack in which an attacker with valid credentials information but no authorization to access various topics seeks to make the broker resource unavailable to legitimate clients [8]. This can be happened by flooding the broker with invalid subscribe or publish requests. This will result in consumption of broker CPU and RAM resources in verifying individual requests.

C. ACK-PSH flooding attack

It is a kind of DDoS attack which consists of sending PSH-ACK packets towards the broker without establishing a TCP handshake [10]. This means that the broker will detect this packet as not belonging to a session and respond with a RST packet. This attack can be carried out by sending classic ACK packets. Its goal is to disrupt MQTT network activity by saturating bandwidth and resources on MQTT broker.

D. SlowTT attack

As described in [13], slow denial of service attack against MQTT called SlowTT aims to saturate and block the available TCP connections of the broker as much as possible in order to prevent other legitimate clients from establishing MQTT sessions and therefore from being able to send or receive

messages due to lack of connections sockets. Once the communication with the broker is initiated by the attacker, it exploits the network configuration parameters adopted by the MQTT protocol, especially the KeepAlive parameter, to keep connections alive for an indefinite time. In addition, SlowTT is able to keep connections open for a long time even with lower keepAlive values by exploiting PINGREQ and PINGRESP packets to simulate legitimate behavior. This attack can be summarized into two major steps:

- Initiate connection with the MQTT broker by using the CONNECT packet.
- Sending PING packets to keep the connection alive for an indefinite time

As a result, maintaining the TCP connections alive prevents the legitimate clients from using them as soon as they are released by the broker.

E. Connect Flooding attack

The attacker sends a large number of CONNECT packets to the victim broker to overwhelm it with the processing of authentications requests [8]. Moreover, attackers can exhaust the CPU or memory resources on the target broker by sending a considerable number of CONNECT packets that have a higher QoS level 2 making services unavailable for legitimate clients.

F. Elevation of privilege attack

Through unauthorized publishing/subscription, the attackers can gain privileges that should not be available to them [12]. In fact, the wildcard feature (e.g. #) available in the MQTT protocol allows the clients to subscribe to multiple topics simultaneously and learn about the list of topics available on the broker and log every message exchanged. As a result, all data published by MQTT clients will be vulnerable to access by the attackers due to wildcard topics.

G. OpenSSL infinite loop attack

This vulnerability has been discovered recently and tracked as CVE-2022-0778 with a CVSS score of 7.5 [17]. The problem comes from a bug that arises when parsing malformed security certificates that contain invalid elliptic curve public keys, resulting in what's so called an "infinite loop." This issue can be exploited by the attackers to trigger a DoS and remotely crash TLS servers consuming client certificates. Table 1 summarizes and compares the different attacks presented above.

IV. Attacks Effects Evaluation Platform

In order to be able to measure the impact of attacks on an MQTT system, we have deployed a realistic architecture using MQTT protocol version 3.1 through open-source broker and client software tools (e.g. Mosquitto, Paho MQTT client) running on virtual machines. As shown in Figure 1, this architecture includes a broker, a legitimate client as well as an attacker. For this purpose, Proxmox Virtual Environment 7.2-4 [14] running on Debian machine was used to host the virtual machines and deploy the broker and client software. Each virtual machine was configured with 2 CPU, 4GB RAM and running Ubuntu 20 OS. This means that it allows to install different operating systems (Windows, Linux, Unix, etc.) on a single computer or a cluster of machines and it also allows deployment and management of containers. Moreover, it provides a REST API for third-party tools. One physical machine is used to configure and manage MQTT network deployed on Proxmox using command-line tools. We first installed the MQTT broker on the broker host VM. Then, we set up a normal network (10.0.10.0/24) of one MQTT publisher node and one MQTT subscriber node running on the MQTT Client VM. The subscriber client is subscribed on different topics on the MQTT broker. We set up one publisher, publishing data across different topics. Similarly, we set up an attacker client running on separate Ubuntu OS 20 virtual machine connected to the broker network which is configured to launch two different types of attacks on the MQTT broker, namely last will payload flooding attack and SlowTT attack. In addition, a separate physical machine running Ubuntu OS 20 installed in another network (192.168.60.180/23) used to set up and configure the MQTT network.

The two open-source MQTT broker and client implementations deployed in this study were namely: Paho MQTT client implementation from [15] since it supports the three MQTT QoS levels, SSL/TLS protocols and Mosquitto broker implementation from [16] in version 2.0.14 as an MQTT broker. To ensure the reliability of identities and the security of information exchange between broker and clients, we generated the certificates required for TLS 1.3 using OpenSSL 3.0.2 as this release has addressed the infinite loop issue presented in OpenSSL 3.0 [17]. The broker was configured with Access Control List (ACL) feature to allow only authorized clients to publish and/or subscribe to various topics. Moreover, Anonymous connections with the broker was disabled to authorise only authenticated clients to publish and/or subscribe to topics.

In order to collect and aggregate system events from the broker to assess the impact of the cyber attacks, another separated VM machine was configured to host cyber risk monitoring platform using SIEM. This platform was designed to monitor an MQTT broker and it allows to collect a certain number of network and system logs. The objective of this platform is to cover a large part of the behaviors of the TCP/IP model on the network side and for the system. The SIEM architecture includes mainly three components. At the Linux userland level, the platform allows to collect all the application logs. At the kerneland level, some endpoint of the infrastructure contains a kernel probe that analyzes the system calls launched by the userland applications. Finally, this SIEM allows to monitor a large number of system behavior via prometheus metrics. These metrics are used to monitor statistics produced by Linux such as disk space used, CPU load, memory consumption, etc.

As shown in Figure 1, one instance of node_exporter is deployed at the broker (endpoint) to expose system metrics via a REST interface. Subsequently, a very light ETL (Extract,

TABLE I
FEATURES COMPARISON BETWEEN ATTACKS

Attacks	Vulnerabilities	User	Sending methods	Vectors
Last will payload flooding attack	CONNECT Last will payload	Internal	Flooding	Dispatching a large number of CONNECT packets accompanied by malicious last will payload.
ACK-PSH flooding attack	TCP	External Internal	Flooding	Sending a massive amount of PSH/ACK packets towards the broker without without establishing a TCP handshake.
Unauthorized Sub/Pub flooding attack	ACL PUBLISH SUBSCRIBE	Internal	Flooding	Sending a large number of invalid subscribe or publish requests using valid credentials to the unauthorized topics in the broker.
SlowTT attack	CONNECT PING	Internal	Low rate Periodic	Sending PING requests everytime before the KeepAlive timer is expired to maintain a MQTT connection alive.
Connect flooding attack	CONNECT	External Internal	Flooding	Dispatching a large number of MQTT CONNECT packets contained a higher size of the payload.
Elevation of privilege attack	PUBLISH SUBSCRIBE	Internal	Low rate	Sending subscribing requests to subscribe to multiple topics simultaneously using wildcards.
OpenSSL infinite loop attack	CONNECT	External Internal	Flooding	Sending subscribing requests to subscribe to multiple topics simultaneously using wildcards.

Transform, and Load) called vector regularly extracts the metrics to send them to our SIEM. This SIEM contains a certain number of databases including Grafana Mimir which allows to store all the time series. They are then made available through a Grafana web interface.

V. EXPERIMENTAL RESULTS AND ANALYSIS

A. MQTT attack traffic generation

As shown in the Figure 1, the MQTT attack traffic was generated from one separate VM machine based on the two DoS attack scenarios, namely SlowTT and last will flooding described in the section III.

The MQTT attacker's machine was configured to generate and transmit a massive number of CONNECT messages to the broker with a varying sizes of the last will payload, where a last will message is sent to notify other subscribers if the client disconnects abruptly. The last will flooding attack was generated using a python script that has been written based on the Eclipse Paho MQTT Python client library. We also wrote other python scripts for initializing and running the publishers and subscribers nodes using the same MQTT client library. As the last will payload flooding is generated from one attack source connected to the network, a multi-processing and multi-threading based approach were adopted in the attacker side to increase its CPU computing power and thus to maximize the impact on the target broker's resources.

The SlowTT attack also was generated from the same MQTT attacker's machine. To do so, we wrote a Python script to send MQTT CONNECT packets in order to establish a connection with the victim broker. After setting the KeepAlive parameter to $240s$, the SlowTT attacker sends updates to the broker using PING packets with a varying periodicity and before the KeepAlive expires. The reason is to keep the connection alive as well as to achieve a realistic message publishing rate.

B. Attack impact analysis

Three performance metrics were measured to assess the impact of SlowTT and last will flooding attacks against the MQTT broker: These three metrics include: latency, memory utilization (RAM) and CPU utilization. Latency refers to the amount of time it takes for data from sending out by a publisher to receiving by a subscriber [19]. Latency was measured using the MQTT broker latency measure tool given in [18]. On the other hand, CPU and memory utilization measures were extracted from Graphana as a csv data. Moreover, we executed each attack for $300s$ and all the measurements for each performance metric were repeated 20 times to ensure the reliability and validity of the results and the mean value for each set of 20 measurements was provided in the next.

In Figures 2 and 3, we study the impact of the last will DoS attack on the performance of the MQTT broker in terms of memory and CPU utilisation, respectively. As a reminder, in the last will DoS attack, the malicious client attempts to disrupt the broker integrity by sending a large number of CONNECT packets contain malicious last will payload.

We performed measurements for both scenarios with and without attacks to identify the influence of malicious traffic

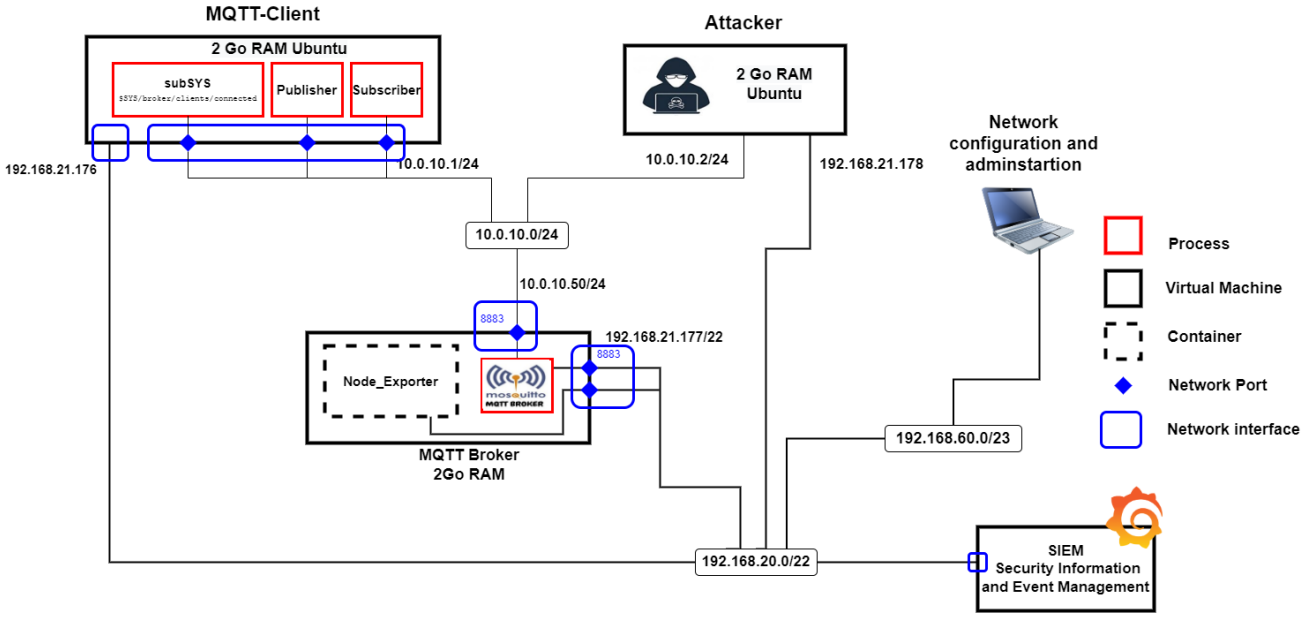

Fig. 1. MQTT broker deployment to assess SlowTT and last will payload flooding attacks.

flow on the server resources consumption. The last will payload flooding attack was launched using 100, 200 and 300 threads. Moreover, the size of the payload was fixed in these results to 20000 characters. Figure 2 indicates that the amount of memory remained constant during the normal traffic flow. However, it was increased drastically during the attack due to the high number of characters included in the last will payload. We observe that the amount of memory used increases from $500Mo$ to $2000Mo$ at $10s$ with 300 threads and from $500Mo$ to $1700Mo$ with 200 threads. These results show that the attack caused the maximum impact on the memory utilization as the RAM idle percentage of the broker reached zero especially where the number of threads is 300.

The results in Figure 3 show the CPU used by the MQTT broker during both normal and malicious traffic flows. These results also show that the computing resources in terms of CPU of the broker are vulnerable to the last will DoS attack. Especially, we can note that the broker is suffering from high CPU load during this attack and it can be potentially exploited to slow down or completely halt the broker.

Figure 4 highlights the last will flooding attack impact on the latency. We can note from this figure that the latency is increasing significantly as the number of threads and payload sizes increase. For instance, we can note that the latency has almost reached 12ms where the number of threads is 300 and the size of last will payload is 10000 characters. Therefore, we can conclude from these results that the last will flooding attack can lead to serious damages especially in very critical environments (industrial, healthcare, smart cars, etc.) where MQTT is used as the messaging protocol to exchange data between IoT devices in real time. The loss of sensitive data in the case where the broker is slow down can arise critical situations.

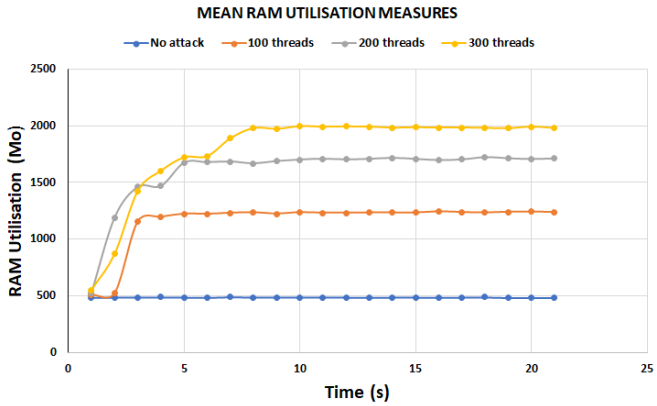

Fig. 2. Last will payload flooding attack impact on the memory utilization with a payload size of 20000 Characters

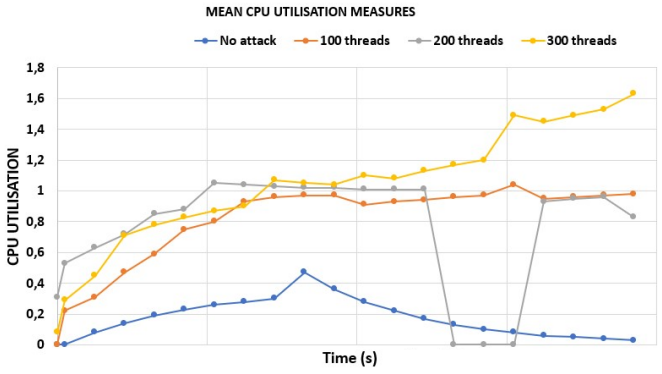

Fig. 3. Last will attack impact on the CPU utilization with a payload size of 20000 Characters

Figures 5, 6 and 7 highlight the SlowTT attack impact on the memory, CPU usage and latency. The experiments configuration was the same as the previous attack; only the attack script changed and the number of threads was fixed to 1024 in these results, as we noted that there is no significant impact of SlowTT attack on the performance of broker for a lower number of threads. We can note from Figure 5 that the memory usage is increased after launching the SlowTT attack and remained constant after a short period of time. Figure 6 also indicates that the broker is vulnerable to SlowTT attack but it has less impact on CPU utilization (more than 50% less) compared to the last will flooding attack. The results of latency given in 4 and 7 show also that SlowTT attack caused less impact as the maximum latency achieved is almost $4ms$ which is 60% less compared to the maximum impact caused by the last will flooding attack. Figure 8 shows the Grafana dashboard related to MQTT broker system metrics collected through Node exporter container. Through a selection control, it is possible to select and monitor different other system and network metrics and time period.

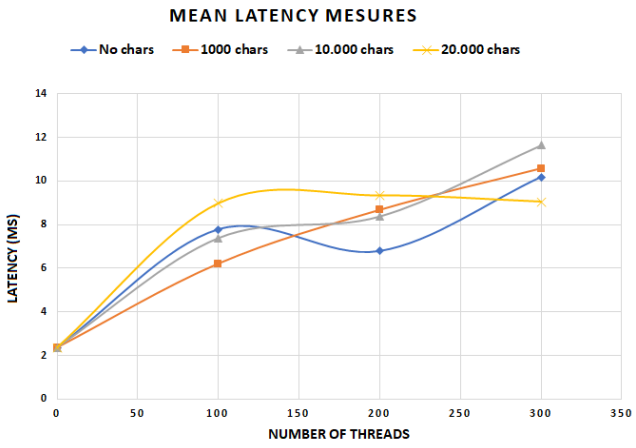

Fig. 4. Last will flooding attack impact on the latency.

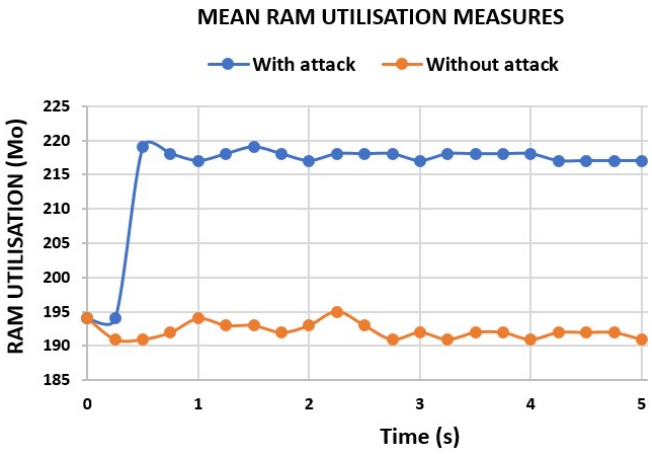

Fig. 5. SlowTT attack impact on the memory utilization.

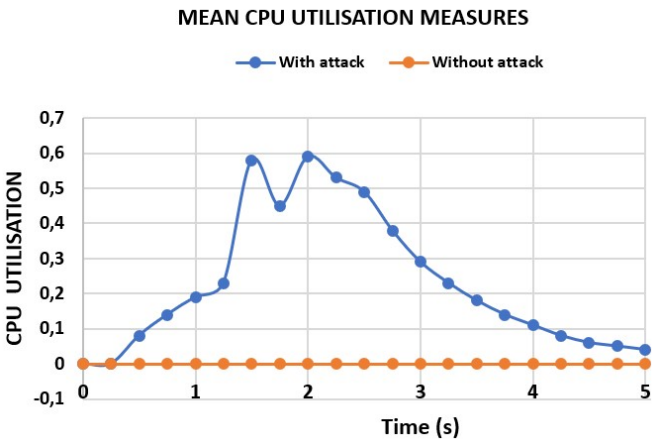

Fig. 6. SlowTT attack impact on the CPU utilization.

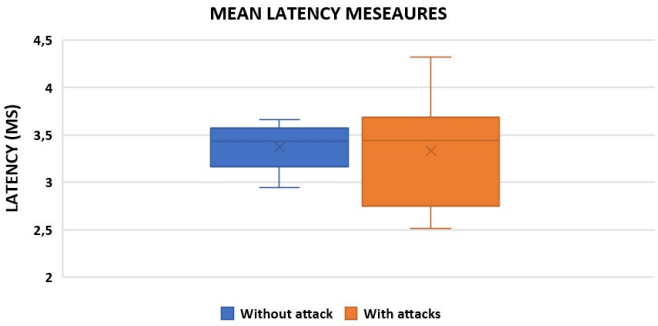

Fig. 7. SlowTT attack impact on the latency.

VI. Conclusion

In this work, we have discussed and compared several DoS attack models that can disrupt the MQTT protocol and we have also presented an evaluation platform using Security Information and Event Management (SIEM) architecture that automatically collects and aggregates system events from the server to assess the impact of the cyber attacks. Overall, the results show that the MQTT protocol is vulnerable to several DoS attack scenarios. As a conclusion to this research work, the last will payload flooding attack can consequently drastically impact the performance of MQTT broker in terms of CPU/RAM usage and end-to-end latency. Moreover, the slowTT DoS attack can also impact the performance but with less damage compared to the last will payload flooding attack.

The detection of these attacks is a crucial element to guarantee the security of the MQTT protocol against possible attacks, and in particular to allow the rapid implementation of adequate countermeasures. Therefore, new detection solutions that can detect and report attacks are henceforward necessary to improve the security of MQTT protocol. As a future work, we plan to implement a Reinforcement Learning (RL) based detection approach to detect and filter out malicious traffic flows in MQTT protocol using system key performance indicators.

Fig. 8. Grafana dashboard used to monitor MQTT broker and also to analyze the server performance metrics.

ACKNOWLEDGEMENT

This research work has been carried out in the framework of IRT SystemX, Paris-Saclay, France, and therefore granted with public funds within the scope of the French Program "Investissements d'Avenir".

REFERENCES

[1] H. Abou-zeid, F. Pervez, A. Adinoyi, M. Aljlayl, Cellular V2X Transmission for Connected and Autonomous Vehicles: Standardization, Applications, and Enabling Technologies, EEE Consumer Electronics Magazine, Vol. 8, No. 6, Nov. 2019.

[2] Ian Skerrett, How 5G and MQTT Accelerate Vehicle-2-Everything (V2X) Adoption, https://www.hivemq.com/blog/how-5g-and-mqtt-accelerate-v2x-adaption, Jun. 2021.

[3] F. Pinzel, J. Holfeld, A. Olunczek, P. Balzer, and O. Michler, "V2v and v2x-communication data within a distributed computing platform for adaptive radio channel modelling," in 2019 6th International Conference on Models and Technologies for Intelligent Transportation Systems (MTITS), pp. 1-6, 2019.

[4] R. Sedar, F. Vázquez-Gallego, R. Casellas, R. Vilalta, R. Muñoz, R. Silva, L. Dizambourg, A. E. F. Barciela, X. Vilajosana, S. K. Datta, J. Härri and J. Alonso-Zarate, Standards-Compliant Multi-Protocol On-Board Unit for the Evaluation of Connected and Automated Mobility Services in Multi-Vendor Environments, Sensors, Vol. 21, no. 6, Mar. 2021.

[5] MQTT in an IoV scenario, https://www.emqx.com/en/blog/mqtt-for-internet-of-vehicles

[6] BMW Mobility Services, https://www.hivemq.com/case-studies/bmw-mobility-services/

[7] F. Chen, Y. Huo, Z. Zhu and D. Fan, A Review on the Study on MQTT Security Challenge, IEEE International Conference on Smart Cloud (SmartCloud), Washington, USA, Nov. 2020.

[8] N. F. Syed, Z. Baig, A. Ibrahim and C. Valli, Denial of service attack detection through machine learning for the IoT, Journal of Information and Telecommunication, Vol. 4, No. 4, May. 2020.

[9] S. Andy, B. Rahardjo, B. Hanindhito, Attack scenarios and security analysis of MQTT communication protocol in IoT system, International Conference on Electrical Engineering, Computer Science and Informatics (EECSI), Yogyakarta, Indonesia, Sep. 2017.

[10] Mohammed Abdulaziz Al-Naeem, Prediction of Re-Occurrences of Spoofed ACK Packets Sent to Deflate a Target Wireless Sensor Network Node by DDOS, IEEE Access, Vol. 9, 2021.

[11] U. Morelli, I. Vaccari, S. Ranise , E. Cambiaso, DoS Attacks in Available MQTT Implementations: Investigating the Impact on Brokers and Devices, and supported Anti-DoS Protections, The 16th International Conference on Availability, Reliability and Security (ARES), pp. 1-9, Aug. 2021.

[12] A. J. Hintaw, S. Manickam, M. F. Aboalmaaly, and S. Karuppayah, MQTT Vulnerabilities, Attack Vectors and Solutions in the Internet of Things (IoT), IETE Journal of Research, DOI: 10.1080/03772063.2021.1912651, May. 2021.

[13] I. Vaccari, M. Aiello, E. Cambiaso, SlowTT: A Slow Denial of Service against IoT Networks, Information 2020, vol. 11, no. 09, Sep. 2020.

[14] https://www.proxmox.com/en/

[15] https://pypi.org/project/paho-mqtt/

[16] Eclipse Mosquitto. Eclipse Mosquitto. Accessed: Mar. 23, 2022. [Online]. Available: https://mosquitto.org/

[17] Openssl infinite loop DoS vulnerability, CVE-2022-0778 (15 March 2022), https://www.openssl.org/news/secadv/20220315.txt

[18] MQTT broker latency measure tool, https://github.com/hui6075/mqtt-bm-latency

[19] R. Banno, K. Ohsawa, Y. Kitagawa, T. Takada, and T. Yoshizawa, Measuring Performance of MQTT v5.0 Brokers with MQTTLoader, in Proceedings of the 2021 IEEE 18th Consumer Communications and Networking Conference (CCNC 2021), Jan. 2021.

[20] M. Amoretti, R. Pecori, Y. Protskaya, L. Veltri, F. Zanichelli, A, Scalable and Secure Publish/Subscribe-Based Framework for Industrial IoT, IEEE Transactions on Industrial Informatics, Vol. 17, No. 6, 2020.