On IoT applications: a proposed SDP framework for MQTT

A. Refaey✉, A. Sallam and A. Shami

In this work, the software-defined perimeter (SDP) is considered for the Message Queuing Telemetry Transport (MQTT) protocol framework in the Internet of Things (IoT) applications. In fact, the SDP provides an additional layer of security with or without SSL/TLS by replacing the traditional login method (username/password) with a single-packet authorisation (SPA) process. This will blacken the end devices, by cloaking and causing them to be inaccessible by attackers. Consequently, this prevents the log-in information from being compromised in the absence of encryption. Eventually, the framework is evaluated through an implementation testbed and system proved to be secure against denial of service active and off-line dictionary types of attacks, even with the use of weak login credentials. All the while, achieving measurable efficiency over the traditional use of MQTT.

Introduction: Ericsson predicts that by 2022 the number of connected devices will reach 29 billion of which 18 billion will be related to the Internet of Things (IoT) [1]. This rising number of IoT connected devices is raising legitimate concerns regarding the security aspects of the adopted protocols. In particular, most of the adopted protocols in the IoT were designed for wireless sensor networks (WSNs) but have suddenly become the core of IoT solutions. For example, the Message Queuing Telemetry Transport (MQTT) protocol is adapted to secure and blacken the application-specific purpose end devices in the IoT era [2]. In fact, this protocol lacks the interoperability and minimal authentication features as it was designed for isolated WSNs without the security of Internet connectivity in mind. Precisely, the protocol enables a two-way handshake for client authentication which allows the use of encrypted messages only if SSL/TLS is available on the resource-starved device. This two-way handshake is susceptible to denial of service (DoS), man-in-the-middle, and off-line dictionary types of attacks as the client's authentication credentials (username/ password) are in the clear in instances when SSL/TLS is not available. Therefore, both mutual authentication and encryption are required to prevent these types of attacks which necessitates additional resources beyond the capability of the zero configuration and limited resources of IoT end devices.

The software defined perimeter (SDP) is a standalone framework, made popular by its use by multiple organisations within the Department of Defense and Intelligence communities. In 2014 and emendate in 2018, the Cloud Security Alliance outlined the initial protocol for the SDP specifications [3]. Adopting the SDP framework will replace the traditional login credential method with a single-packet authorisation (SPA) process. The SPA packets are encrypted and authenticated with a Hash-based Message Authentication Code signature, in addition, they contain a one-time-use random field. Indeed, a malicious actor cannot spoof an SPA packet by itself, as it would need to compromise all associated SDP credentials by undermining all five layers of the SDP security. In addition, the SDP Gateway logs all fully validated SPA packets, making the broker immune to both impersonated and replayed SPA packets. Also, it enables a less complex and more secure process for the limited resource end-devices by moving the authentication process to the gateways. In the next section, details on the SDP framework is provided, followed by a virtualised testbed to introduce and evaluate the aforementioned architecture.

Software defined perimeter (SDP): The design of the SDP framework allows the authorised clients (devices or users) to access the public Internet by ensuring a secured connection to systems and services while denying access to everything else. Therefore, it mitigates network security threats by incorporating the idea of dynamically provisioned perimeters. Precisely, through the existing fixed perimeter security approaches any network-connected system is accessible. However, in the SDP, only authorised devices have access to the network-connected systems, which are otherwise completely hidden from public view, through extensive but light-weight validation processes. Furthermore, the SDP is software-based which qualifies it to be a scalable security solution for integration into the IoT era where the perimeters here could function as the end-devices. There are five distinct layers of security control which characterise SDP, and they are as follows:

i *Single-Packet Authorisation*: SPA is used to initiate communication between the initiating host (IH) controller (on the end-device) and the accepting host (AH) controller (on the gateway);
ii *mTLS or IKE*: Mutual TLS or Internet key exchange (IKE) is used prior to further validation to enable two-way authentication between end-devices and the gateway;
iii *Device Validation*: this step is required to ensure the device is running a trusted software and possesses a private key (not expired or revoked);
iv *Dynamic Firewall*: is applying access control rules based on the access rights assigned to the previously approved and validated end-devices;
v *Service Binding (SerB)*: after the validation and authentication processes are completed for the end-device (client) and access has been granted, a service-binding layer creates an encrypted TLS tunnel to protect the service's access.

In the following section, the benefits of using the SPA for the MQTT are explained.

Single-packet authorisation (SPA): Herein, the system starts with a single packet authorisation which is based on RFC-4226. The first packet is sent to the SDP controller which is installed on the MQTT server (Broker) by the end-device (subscriber or publisher) looking to access the network or a specific service. Consequently, the broker has to cryptographically verify the device's identity. The benefits of this process can be summarised as follows:

- *The MQTT servers (Broker) are blackened*: The server will not respond to any connections until the end-device (Subscriber or Publisher) has provided an authentic SPA and been authorised with the use of protocol;
- *Mitigates DOS on TLS*: It allows the MQTT server to discard the TLS DOS attempt before entering the TLS handshake. Precisely, the server discards all suspected packets and only accepts the SPA packets which contain a one-time-use unique and random field;
- *Attack detection*: The first packet received by the server is expected to be an SPA or a similar construct, otherwise, it is viewed as an attack. In particular, the SDP is capable of identifying an attack based on a single malicious packet.

SDP security workflow: The SDP workflow as illustrated below gives maximum protection to the systems and patches most of the vulnerabilities found in the legacy security systems. The SDP workflow goes into the order explained in Fig. 1

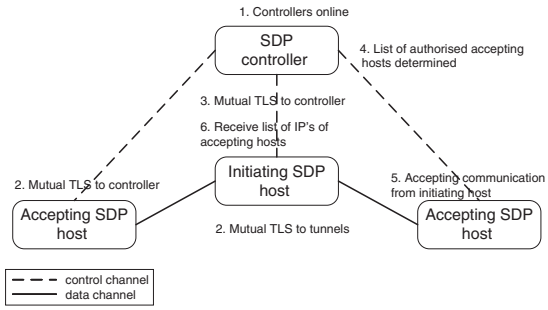

Fig. 1 *SDP workflow*

SDP-MQTT platform: An MQTT network consists of three main components, a broker, publisher, and subscriber. The broker is responsible for managing the transmissions between subscribers and publishers. It can also buffer the messages while subscribers are offline. The publisher publishes a message for a particular name/topic, and the subscriber listens for a message with that name/topic. A simplified MQTT model is presented in Fig. 2. To defend against the DoS Attack, the SDP modules were embedded in the MQTT network nodes. In particular, the SDP initiating host (IH) was installed in the MQTT subscriber and the SDP accepting host (AH) module was installed in the MQTT Broker. Additionally, a separate machine is added to the network to run the SDP Controller (CTRL).

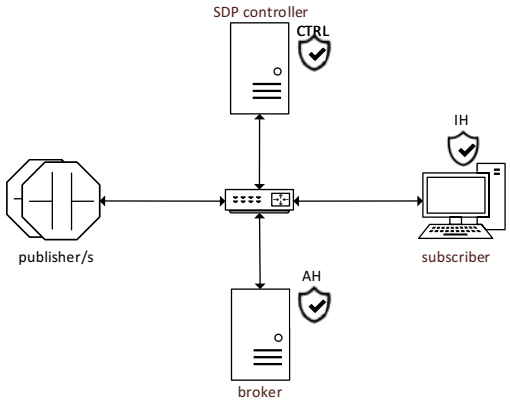

Fig. 2 *SDP platform in MQTT network*

In this work, the testbed consists of a set of Ubuntu 16.04 LTS virtual machines hosted by a physical machine running linux Ubuntu 18.04 and VirtualBox 5.2. The MQTT network comprises five MQTT publisher VM running Mosquitto clients 3.1 connected to another MQTT broker/server VM running Mosquitto server 3.1. Additionally, another VM was created to run an MQTT subscriber client. The SDP platform used in this testbed utilise the modules of the Open Source SDP project developed by Waverly. The fwknop module was installed on the broker to represent the SDP AH. Again, the fwknop module was installed on the subscriber VM, but with different settings to represent the SDP IH on the legitimate client. Furthermore, the SDPController module was installed on a separate VM to represent the SDP Controller.

This new architecture provides solid support for MQTT security and privacy. More specifically, whenever a new subscriber/client tries to access the broker, it has to follow a strict authentication procedure as shown in Algorithm 1. This is regardless of the user and password authentication built-in MQTT protocol. Furthermore, the fact that the firewall at the broker has a drop-all policy makes SDP worthwhile to repel different types of malicious attacks successfully such as flooding attacks.

Algorithm 1: MQTT subscriber authentication in SDP

1: Assumes AH is already initialised/authorised
2: **if** SPA packet is valid **then**
3: The CTRL establishes an mTLS secured connection between itself and the IH
4: The CTRL verifies the IH a certificate present at the subscriber
5: The CTRL sends all the information about the IH's authorised services to the AH
6: The AH sets up the corresponding firewall rules to allow this connection for τ period of time
7: **if** $\tau > 0$ **then**
8: The subscriber attempt to connect to the broker
9: **while** Connection tracking is valid **do**
10: Transmit packets
11: **end while**
12: **else**
13: The AH removes the corresponding firewall rules
14: **end if**
15: **else**
16: Drop the packet
17: **end if**

Performance overhead of SDP: The proposed architecture ensures a seamless integration between the SDP and MQTT. In particular, the SDP authentication time is the only delay that added to the connection setup of the MQTT different components. Fortunately, this authentication procedure is only required once for each component unless it loses the connection for some reason. It is worth mentioning that, it takes 4.182067 and 2.068458 s for the AH and IH to start up, respectively. In this architecture, each publisher is sending messages at steady rate. The message rate is set to 1 message per second per publisher asynchronously. The message payload is fixed for each publisher and set to approximately 30 bytes. This test does not try to reach the maximum message throughput. However, the goal is to show how the network throughput would differ with and without the SDP platform. In this way, the number of publishers give global message throughput (Table 1).

Table 1: Performance evaluation results

Criterion	Without SDP	With SDP
connection setup time, s	0.086	2.154458
average network throughput, %	0.44	73.6

On the other hand, to launch a DoS attack, the broker was hammered by a flood of TCP packets. These packets have an SYN flag to establish a TCP three-way-handshake using a fake IP address. The server unaware of the attack, therefore, it responds to each attempt with an SYN-ACK packet. However, the malicious client does not send the expected ACK. This attack was initiated using a command-line oriented TCP/IP packet assembler/analyser called hping3.

To demonstrate the effect of the DoS attack on the MQTT Broker CPU usage, the CPU load average was logged every second using the uptime Linux utility. The load average is a cumulative value that represents the average system load calculated over a given period of time: 1, 5, 15 min. It is not just a measure of the number of processes in runnable state, but also the number of processes in the uninterruptible sleep state. Thus, the CPU load would be greater than 100% if the CPU overloaded. The following command was used to collect the CPU load average values.

while true; *do uptime* $\gg$ *cpu.log*; *sleep* 1; *done*

Fig. 3 shows that the load variations of the broker CPU usage under the DoS attack when the SDP was disabled and when the SDP was enabled. The attack occurs after 60 s. When the SDP was disabled, the CPU usage increased quickly and reached 100% after 251 s. On the other hand, when the SDP was enabled, the CPU usage increases slowly and reach 39%. Then, the CPU usage almost never changes until the end of the experiment duration.

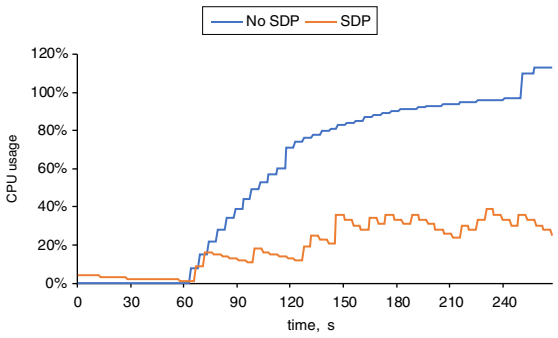

Fig. 3 *CPU usage*

Conclusion: Indeed, the MQTT protocol represents an ideal messaging protocol for IoT and M2M communications. However, it is incapable of providing routing for end-devices in vulnerable and low bandwidth networks. Herein, an SDP framework was introduced to provide solid support for MQTT security and privacy. This framework was evaluated through an implementation testbed and proved its robustness against a DoS attack while maintaining the lightweight requirements.

Acknowledgments: The authors thank Ms. Juanita Koilpillai from Waverley Labs for her help to complete this work.

Submitted: *10 July 2019* E-first: *19 September 2019*
doi: 10.1049/el.2019.2334
One or more of the Figures in this Letter are available in colour online.

A. Refaey (*Manhattan College, NY 10471-4099, USA*)

✉ E-mail: ahmed.hussein@manhattan.edu

A. Sallam and A. Shami (*Electrical and Computer Engineering, Western University, Canada N6A 3K7*)

A. Refaey: Also with Electrical and Computer Engineering, Western University, Canada N6A 3K

References

1 Zaidi, A., Hussain, Y., Hogan, M., *et al*.: 'Cellular IoT Evolution for Industry Digitalization'. Ericsson white paper, GFMC-19:000017 UEN, January 2019

2 Locke, D.: 'MQ telemetry transport (MQTT) v3. 1 protocol specification'. IBM developerWorks, Markham, ON, Canada, Tech. Lib., 2010

3 Cloudsecurity alliance (CSA) standards: 'SDP Specification v2.0 – Cloud Security Alliance: Cloud Security Alliance', November 2018