2018 IEEE 32nd International Conference on Advanced Information Networking and Applications

Improving Security on IoT Applications based on the FIWARE Platform

Caio Thomás Oliveira*, Rodrigo Moreira*, Flávio de Oliveira Silva*, Rodrigo Sanches Miani* and Pedro Frosi Rosa*
*School of Computer Science (FACOM)
Federal University of Uberlândia (UFU), Uberlândia, Brazil
Email: caiothomas@ufu.br, rodrigo.moreira@ufu.br, flavio@ufu.br, miani@ufu.br, prfrosi@ufu.br

***Abstract*—Internet of Things (IoT) has increased its presence in many environments. However, this means a greater exposure of sensitive data, rising potential security threats. Thus, security becomes a key requirement for the protection and prevention of cyber attacks to IoT applications and devices. FIWARE Platform, for example, has an architecture of components responsible for interconnecting devices to IoT applications, decreasing complexity and providing a standard set of services to developers. The IDAS 5 version of the platform presents several security gaps, so this work aims to solve some of them by incorporating end-to-end security services using encryption and access control in all NGSI requests (RESTFul API). The main contribution of this paper is the implementation of the DTLS 1.2 protocol in NodeJs to support the LWM2M/CoAP protocol and its addition to the IoT Agent. This paper also allowed the IoT Agent of the FIWARE Platform support TLS communication to the MQTT protocol. Through an experimental evaluation, it was possible to validate the implementation. Our preliminary results show that the encrypted requests had a small increase regarding latency, but this cost is compensated by the increase of security in FIWARE based IoT Applications. The source code of this work is open on the GitHub and it can be used to support security services in other IoT communication protocols.**

I. Introduction

IoT is an emerging paradigm which aims to contribute to several kinds of application domains, so its impact is significant in the industrial and academic scenario [1]. The variety of applications developed upon this context ranges from health, cities, agriculture, transportation, military, commercial, domestic, among others[2], [3].

Given such impact, many companies and researchers concentrate their efforts on expanding the development of IoT applications. They seek answers to the challenging questions regarding the hardware and software technologies used in their projects [4], [5]. It is possible to note that when deploying real IoT systems, there are a significant number of challenges, among them are related aspects of vulnerabilities of the use of the open network in devices which show problems with privacy, security, authentication, and authorization [1], [3].

Security in IoT plays an essential role due to two main reasons: several things may interact together in a complex manner, through many security techniques and according to different policy requirements [6] and IoT devices can have different operating environments and, usually, limited computational power. Given an individual IoT application, it is necessary to adjust the security requirements of transmitted data through encryption/decryption and authentication. Privacy is another concern, especially in situations where devices are collecting sensitive data, and people do not want to disclose their information.

FIWARE [7] is a Future Internet platform which enables working in the IoT context through a set of middleware and services that aim to integrate devices, protocols and information. Despite being a robust platform, some project components have little or no security mechanisms developed so far. Hence, the primary objective of this work is to increase the security of IoT applications based on the FIWARE platform by proposing security mechanisms to ensure confidentiality and access control at the application layer.

With that in mind, mechanisms of confidentiality and access control have been incorporated in IoT Agent Node Lib [8]. An IoT Agent in the context of FIWARE is a component, also called Generic Enabler (GE), that lets groups of devices to send their data for and be managed from a FIWARE NGSI Context Broker by using their native protocols. We can summarize our contributions as follows:

- provide the integration of the IoT Agent to a new element named IoT Broker-FI;
- introduce the support for the Transport Layer Security (TLS) in the IoT Agent for the Message Queuing Telemetry Transport (MQTT) protocol;
- introduce the support for Datagram Transport Layer Security (DTLS) 1.2 in the IoT Agent for the Lightweight Machine-to-Machine/Constrained Application Protocol (LWM2M/CoAP);
- propose a strategy for allowing Next Generation Service Interface (NGSI) requests to be encrypted using Secure Hypertext Transfer Protocol (HTTPS) and integrated with IoT Broker-FI and IoT Agent;
- evaluate the impact of each proposed solution using a simple application.

Support for DLTS 1.2 has been tested to a compatible server, LESHAN Lightweight M2M and the impact of incorporating security mechanisms are experimentally validated through the combined use of IoT Broker-FI, IoT Agent MQTT, IoT Agent LWM2M/CoAP and the components of the FIWARE platform, Orion Context Broker and PEP Proxy.

The new GE, called IoT Broker-FI is also a contribution of this work. The IoT Broker-FI, fully compatible with the latest

1550-445X/18/$31.00 ©2018 IEEE
DOI 10.1109/AINA.2018.00104

version FIWARE Backend Management (IDAS R5) [8], allows the aggregation of data from different devices in a single entity in the context and communicates with the IoT Agent in a secure manner

This paper is organized as follows: Section II presents some concepts of the FIWARE framework, its current state in relation to security; Section III presents a view of how security is implemented in different platforms used in IoT; Section IV presents the proposal for enhancing security; and, Section V presents our experiments and results analysis in the Section VI. Finally, Section VII presents the concluding remarks and future work.

II. FIWARE Platform: A brief security analysis

The FIWARE is a public and open source platform available on the Internet. It is based on the open infrastructure platform named OpenStack which proposes an enhancement for IoT functionalities. In the FIWARE platform, the applications and services are composed by the Generic Enablers (GEs) which can be instantiated to be executed.

These GEs use the Next Generation Service Interface (NGSI-09/NGSI-10) as the standard for communication between applications. The NGSI standard was developed by the Open Mobile Alliance (OMA) which provides a powerful interface for use in various web services through Representational State Transfer (REST) [9].

FIWARE has several GEs which can be used for several applications in different domains. To implement a standard application you can use the main components: IoT Agent, NEC IoT Broker, PEP Proxy and Orion Context Broker. In summary, the IoT Agent consists of a gateway that manages the devices. The PEP Proxy redirects only authorized NGSI requests. The NEC IoT Broker is used to perform the association and composition of the IoT Agents, that is, it is possible to compose data from different sensors which can communicate through different protocols associated with a particular entity. The Orion Context Broker stores the data collected by the devices.

The IoT Agent is software modules responsible to convert specific IoT device protocols such as LWM2M/COAP, MQTT, and SIGFOX to the NSGI calls that enable the communication with other FIWARE GEs. The IoT agent represents an abstraction layer between the devices and the FIWARE platform. The IoT Agent supports different ways to communicate with the device in order to produce a context based on the sensing information. In the *active* attributes mode, the device will continuously notify that IoT Agent about the new measurements of a specific attribute. In the *lazy* attributes mode, updates of the device will only be transferred to the IoT Agent when requested. Finally, the *command* attribute mode allows the IoT Agent executes commands in the device [10].

Figure 1 depicts the FIWARE architecture view and the GEs necessary to be used in a standard scenario.

In Figure 1 we can see that the IoT Agent is in the IDAS 5 version, while the NEC IoT Broker is in the IDAS 4 version. As there have been changes in the NGSI interface,

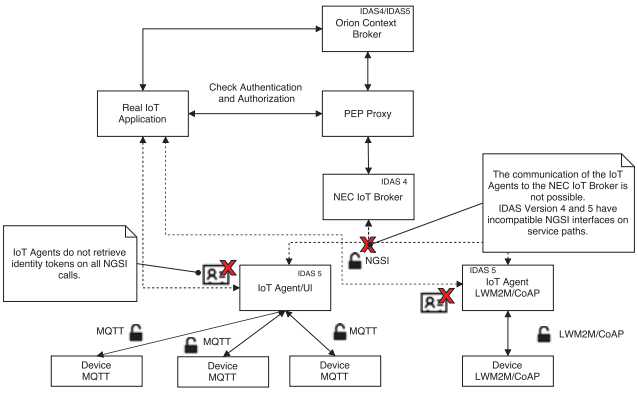

Fig. 1. FIWARE architecture for a standard scenario.

the communication between them was not feasible due to compatibility issues. Among the changes, we can mention the mandatory inclusion of headers in the request for service and subservice. Thus, IoT Agents require the use of service information, but the NEC IoT Broker does not have such functionality making association and composition of various devices impracticable.

FIWARE provides the PEP Proxy to authorize NGSI requests sent from the IoT Agent to the Orion Context Broker. However, as we can see in the IoT Agent, the token recovery implementation to perform authentication and authorization was inadequate for the NGSI requests. The requests that do not implement the token retrieval capability are *UpdateContext*, *RegisterContext*, *SubscribeContext* and *unsubscribeContext*. *UpdateContext* and *RegisterContext* requests are essential because they are executed whenever a device is connected to the IoT Agent. This makes it impossible for the PEP Proxy to work correctly. This eliminates the ability to provide authentication and authorization of NSGI requests.

The security mechanisms adopted in the IoT Agent-UL for the MQTT protocol present some issues. First, in the IoT Agents, only the messages sent from the devices had client authentication in the MQTT Broker. Besides, this IoT-Agent version does not support encryption. This could lead to an active attack by intercepting packets and reading plain data such as topics, username and password.

Another significant problem is that the IoT Agent considers the control of packets accepted by MQTT Broker through defined topics. Thus, MQTT Broker filters the topics which are incorrect, however, when the packets are not encrypted another user can use the same topic to send altered data. This would make it possible to exchange confidential information or execute a command on the device without significant challenges. Hence, we build on an encryption mechanism that ensures the external elements do not view the pattern of threads accepted by the MQTT protocol.

Similarly, LWM2M/CoAP has no encryption on its connections implemented in NodeJs for the IoT Agent. This gap allows an attacker to view the information in the CoAP

package and execute commands on these devices. To overcome this, we implemented DTLS 1.2 in LWM2M/CoAP that had an adequate and expected behavior.

III. Related Work

IoT platforms have become a trend and offer a set of middleware which are used to develop an IoT application running in cloud computing. Middleware has gained prominence due to the important role in simplifying the development of new services and the incorporation of technologies [11]. This makes it essential for external IoT applications to use their API without having to be aware of protocols and semantic definitions [1]. Several proposals are implemented by the industry, in this section a comparison of their resources is presented.

AWS IoT [12] is an API which allows devices to authenticate and connect to AWS IoT by using MQTT, HTTP or WebSockets protocols. The AWS IoT device gateway authorizes devices to communicate with AWS IoT securely by using a publishing template. AWS IoT provides access control and confidentiality using the Transport Layer at all points of connection of the distributed system. The security protocol is TLS using X.509 certificates, and the authentication method is the Amazon SigV4 [13].

Kaa [14] is a middleware platform which allows the construction of complete IoT solutions. The platform supports encryption for devices through TLS.

The IBM Watson IoT [15] platform consists of a set of tools and components through cloud computing. It supports only the MQTT and HTTP protocols to communicate the gateway and devices. For applications, secure APIs are offered to use data analysis tools.

The Azure IoT [16] supports the MQTT, HTTP, and Advanced Message Queuing Protocol (AMQP) protocols. Also, it allows you to authenticate the devices to send the data.

Table I provides a view of the features which are implemented by the presented platforms. We can see that they all run in cloud computing and allow you to manage the data through your APIs. Besides, they provide documentation to implement the communication of available messaging protocols. The commonly used protocols are MQTT and HTTP. FIWARE offers more protocols such as LWM2M/CoAP which can be used on devices with low network resources and the SigFox that makes it possible to use its own communication network. However, the access control is not properly implemented on your gateway and does not present end-to-end confidentiality among its components.

In this work the FIWARE platform is used and the section IV describes the main components which are used to develop a real IoT application.

IV. Enhancing FIWARE Platform Security

This section addresses the security mechanisms of confidentiality for protocols MQTT, LWM2M/CoAP and NGSI in the IoT Agent. Also, it introduces the new IoT Broker-FI component which aims to update the NEC IoT Broker (version IDAS 5) of the FIWARE platform.

Figure 2 illustrates an overview of the proposed FIWARE architecture required to support security aspects involving authentication, authorization, and confidentiality.

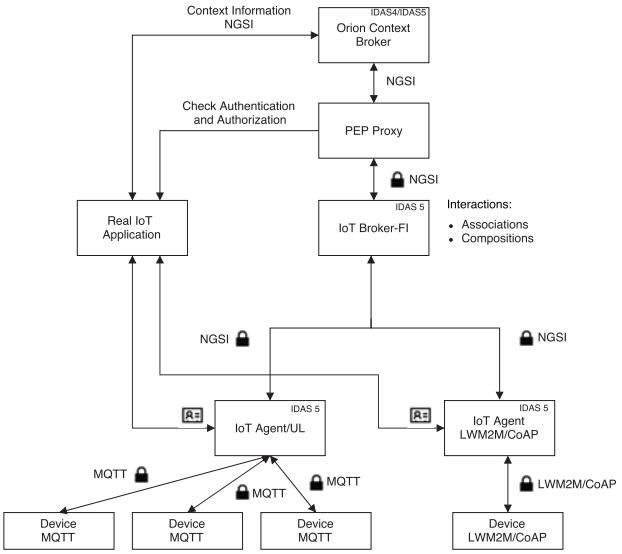

Fig. 2. Overview of the proposed architecture.

According to the proposed architecture, the IoT Agents must perform the authentication and authorization to retrieve the identification information which will be attached to the validated request header. PEP Proxy will be used to implement the authorization and authentication of the tokens sent by the IoT Agent and also check the request token to redirect to the Orion Context Broker. Since NEC IoT Broker is incompatible with the IDAS 5 version of the IoT Agents and it does not have secure NGSI communication [17], there is a need to develop a new component which has a purpose of realizing associations and compositions of IoT Agents with different protocols using a secure communication.

The secure communication capabilities are addressed in Section IV-A, access control in Section IV-B and the features implemented in IoT Broker in Section IV-C.

A. Confidentiality

Confidentiality ensures that only authorized users will have access to classified information [18]. In our context, developing confidentiality requires that a third party could not read the packets exchanged by sender and recipient.

The current version of the IoT Agent is IDAS 5 and it has been developed in Node Js. IoT Agent consists of several libraries, and they are used to provide a standard of agents for different protocols. IoT Agent-NodeLib is a library which performs the registration of devices, services and it delivers NGSI functionality regardless of the used protocol. Thus, IoT Agents for each protocol might import this library to support different protocol versions.

TABLE I
COMPARISON OF IOT PLATFORM FUNCTIONALITIES.

Requirements	AWS IoT [12]	Kaa [14]	IBM Watson IoT [15]	Azure IoT Hub [16]	FIWARE [7]
Virtualization (Cloud Computing)	Yes	Yes	Yes	Yes	Yes
Device Protocols	MQTT, HTTP, WebSockets	MQTT, HTTP	MQTT, HTTP	AMQP, MQTT, HTTP	MQTT, HTTP, LWM2M/ CoAP, SigFox
Gateway	Yes	Yes	Yes	Yes	Yes
Access control	Yes	Yes	Yes	Yes	Not all NGSI calls implemented on the gateway (IoT Agent).
Confidentiality	TLS	TLS	TLS	TLS	No
API Data Management	Yes	Yes	Yes	Yes	Yes
Documentation	Yes	Yes	Yes	Yes	Yes

Figure 3 shows the stack of protocols needed to implement packet encryption (confidentiality) on the IoT Agent. Note that the device sends information using MQTT or LWM2M/CoAP, and such data is converted to NGSI by using the HTTP or HTTPS protocols. Thus, one of the goals of this paper is to develop the security mechanism using such existing protocols and adapting it to our needs. With that in mind, we use the following cryptographic protocols for encrypting network packets: TLS [19] for the Transmission Control Protocol (TCP) and DTLS [20] for the User Datagram Protocol (UDP).

The implementation of the security features in the FIWARE platform will be detailed according to the proposed architecture of Figure 2 and the protocol stack of Figure 3.

First, there is a need to encrypt NGSI messages between the IoT Agent, IoT Broker-FI and the PEP Proxy. Therefore, it is necessary for both applications to talk to each other through HTTPS servers, which is the implementation of TLS over the HTTP protocol. In doing so, the NGSI calls can be transmitted over an encrypted connection.

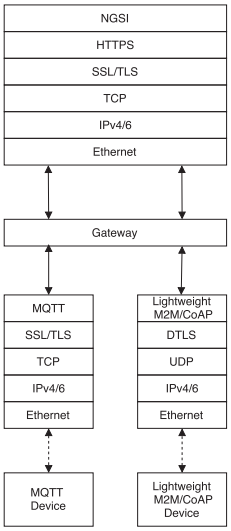

Fig. 3. Stack of protocols required to implement secure communication in the IoT Agent.

MQTT protocol consists of a messaging-oriented application protocol which provides communications flow control control with delivery guarantee [21], [22]. MQTT uses the publish/subscribe paradigm for message exchange. This paradigm uses a MQTT Broker which is responsible for receiving, queuing, and relaying messages received from publishers to subscribers [23]. Since security should be presented in all of the components (device, MQTT Broker server and IoT Agent), a study in the MQTT library was required to implement security at the IoT Agent. The MQTT library used in the IoT Agent already implements TLS resources with certificates. The idea here is to provide an IoT Agent version with the Pre-shared Key (PSK) implementation.

To implement the IoT Agent, some changes were made in order to connect and establish the certificate settings in the component configuration files. For example, establishing the handshake in every message exchanged between sender and recipient might generate an overhead in the network. To prevent this, the MQTT device is able to set up the connection time with the MQTT Broker.

Lightweight M2M/CoAP is a protocol specified by the Open Mobile Alliance for constrained devices in IoT [24]. The stack LWM2M use the CoAP protocol to transfer the data between the client and server. This makes use of a light and compact protocol which allows the device management based in simple object of resource model through CoAP call abstraction. CoAP protocol uses UDP as the transport layer which uses the DTLS protocol to perform the encryption of its messages. DTLS can be used to have the same guarantees as of the TLS over TCP used by MQTT. The DTLS is similar to TLS and uses a PSK key on the device. According to the OMA specification [9], the DTLS 1.2 protocol was not implemented in the NodeJS FIWARE Lightweight M2M protocol [25].

Regarding M2M communication, the LWM2M/CoAP protocols require different CoAP requests, depending on credential types. According to the specification, the LWM2M supports three different types of credentials, namely: Certificates, Raw public keys and PSK [26]. Thus, when a client initiates communication, the first step is to register for the LWM2M server. In case of confidentiality, client and server should establish secure channel before exchanging of information. Upon establishing secure communication, as defined in the specification [26], the DTLS client and server must remain for a long term, by avoiding compromising the security context.

We used the Neustar library [27] to implement the DTLS functionality over the Lightweight M2M. This library implements the CoAP protocol by using security layer DTLS encapsulation [28]. The CoAP/DTLS protocols, developed by Neustar, use ARM mBed which is an open source library for TLS development. The PSK key is stored on the client and

server, being that only the server has a certificate which will be used to be sent to the client to establish secure channel.

Some modifications were made in the Neustar library to fully support the Lightweight M2M protocol. Originally, the Neustar library closes a socket after a connection. However, according to the LWM2M/CoAP specification, the connection must remain open for sending requests to manipulate the objects [26]. Therefore, in our implementation, the socket remain open and keep the security state, such as session keys and security parameters established in first COAP request.

With Neustar library, we made some CoAP protocol updates to reuse the open socket when establishing secure communication session. This is a specification driven approach, which is useful for the exchange of data, by reducing new handshakes made for constrained devices. To determine socket incoming messages, the NodeJs Event Library has been used to trigger reactive functions. This way, the socket is listening for the incoming messages and it sends them for the specified function to handle the request.

To validate the DTLS 1.2 implementation was also carried out through protocol communication tests with LESHAN. LESHAN is a Java implementation of the OMA Lightweight M2M server and client, which already implements the security layer [29], [30]. The goal of this evaluation is to enabled a Java client (LESHAN) to securely communicate with the developed IoT Agent.

B. Access Control

To prevent unauthorized agents change data on the Orion Context Broker resources is mandatory to implement access control. Authorization and authentication functionalities were not fully supported for the following IoT Agent calls: *UpdateContext*, *RegisterContext*, *SubscribeContext* and *UnsubscribeContext* methods. For this reason, access control to the Orion Context Broker does not work as expected.

The access control requires that the user register their information in a security application such as the OpenStack Keystone. When the IoT Agent is triggered by some device, a *token* requesting is made for the security application along with the user data, password and entity key. This system checks it out if the information match, if so, this *token* is sent to be attached to the NGSI request header.

The *token* allows devices to access private resources of the entities for a specified period of time. The security application is in charge to provide the *token* which defines the access time. Figure 4 shows the sequence diagram depicting the required steps to perform the token retrieval on the IoT Agent.

A careful IoT Agent code refactoring has been done by regarding the necessary access control features implementation. Modifications has been introduced to support NGSI requests for the authentication and authorization functionalities, by regarding access control of the IoT Agents to the PEP Proxy, to retrieve *tokens*. Thus, the PEP Proxy verifies whether a *token* is valid to permit, or not, that a request could be sent to the Orion Context Broker.

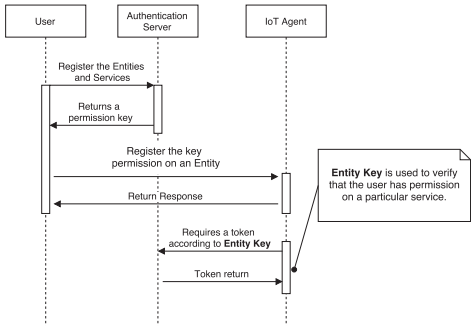

Fig. 4. IoT Agent sequence diagram for retrieving a *token*.

Also, modifications has been introduced for the IoT Agent to check it out the token of the incoming NGSI requests from the Orion Context Broker, in the scenarios which execute commands and request data in a lazy way on the device. Otherwise, such requests can perform functions on devices even if they are not validated. In this case, all requests NGSI must have a valid *X-Auth-Token* header.

C. IoT Broker-FI

NEC IoT Broker for the FIWARE is not compatible with the IoT Agent and Orion Context Broker due to the older NGSI communication interfaces. The NEC IoT Broker requires that every entity should be manually stored in the IoT Discovery, which made it impractical for a real application. For this reason, a new broker capabilities component has been developed.

The new component is named IoT Broker-FI and encompasses the same features of the other one (NEC IoT Broker), but with other functionalities. One of them is related to the way that NGSI reads the requests. This new method aims to read the NGSI request and store the information collected without human intervention, different from the model proposed by NEC IoT Broker where it was necessary to register each entity.

Therefore, IoT Broker-FI development aims to aggregate functions provided by the NEC IoT Broker, to provide compatibility with IoT Agents and also to interact with a large number of providers and consumers automatically. Besides, our IoT Broker yield secure communication on NGSI requests using TLS and was developed in NodeJs following the same architectural standards adopted by FIWARE developers.

V. Experimental Evaluation

To validate the proposed secure communication architecture among the IoT Agent, the IoT Broker-FI, and the Orion Context Broker, we set up two test scenarios. The first one involves creating a scenario with no security requirements and the second one considers the use of encryption for message exchanged between the devices, complemented with authentication and authorization, as proposed in the architecture presented in Figure 2.

A. Evaluation Method

For both MQTT and LWM2M/CoAP protocols, we perform the inspection of the packets to validate the sequence of messages necessary for the communication between the involved components. Also, the packet inspection allowed us to observe the use of encryption in each scenario.

Subsequently, the latency measurement was performed in each scenario. In our context, latency is the communication time between a device and the Orion Context Broker. Along with this path, the message could be sent or received by the device, always going through the IoT Agent and IoT Broker.

For the MQTT protocol the latency benchmark was performed for *active* and *command* attributes. In the case of *command*, the evaluation examined the influence of the handshake time. For this, the first approach considered the realization of the handshake at the moment of the connection of the device with the MQTT Broker. In the second approach, for each message sending through the device, a handshake is performed. For the LWM2M/CoAP protocol the latency measure was conducted considering *active, lazy* and *command* attributes.

We run experiments on a local network using a computer with Intel Core I7 with four cores of 2.00GHz with 8 GB of RAM and Ubuntu operating system. The tool for analyzing the packets traces and collecting the time data is Wireshark. It is important to note that for each protocol, its respective IoT Agent has been used.

B. IoT Agent-UL MQTT

It is possible to store the information device groups which have a *type*, *trust* and *apikey* in the IoT Agent. When the device does not have its registration and such *apikey* exists in IoT Agent-UL MQTT, the referred device begins to share the characteristics of this device group. As a result, the device information is stored in the IoT Agent and the *Register Context* is sent through a NGSI request to store the entity in the Orion Context Broker.

After the *RegisterContext* is completed, the device can execute the updates context on the Orion Context Broker. This mode of operation is part of the *active* scenario where the device actively makes periodic updates of its properties.

C. IoT Agent LWM2M/CoAP

The OMA Lightweight M2M protocol has a set of interfaces which are used by the client and the server (IoT Agent). When a device connects to the server, the client first sends the registration request.

It is important to note that the IoT Agent first makes a *RegisterContext* and then, a CoAP request to receive notifications when the data is changed through the *observe* method. Thus, if an update of the object on the device occurs, the data is sent to the context. In fact, this represents the *active* attribute scenario proposed by FIWARE.

VI. Result Analysis

This section presents a comparative analysis of the latency time for the data exchanged between the device and the Orion Context Broker in one scenario without security and another with the security approach proposed in this work.

Wireshark tool has been used for the measurements. In Figures 5 and 9, the Y-axis presents the latency in seconds (s) and the X-axis indicates the experiment identification. To improve the analysis of each scenario, the experiments were replicated twenty-four times (E1 to E24), and the averaged results are presented using a 95% normal confidence interval.

A. IoT Agent MQTT

The IoT Agent MQTT only supports two different operation modes: *active attributes* or *command*. On both operation modes, the *handshake* happens before the MQTT data exchange. As a default behavior of FIWARE architecture, before each NGSI request a new *handshake* is executed.

Figure 5 presents the latency time considering MQTT *active* attributes in the scenario where the TLS based security is used versus the one with no confidentiality applied to the messages. The average duration with security is $0,07354 \pm 0,00851$ seconds and $0,048199 \pm 0,003693$ seconds without security. In this case, latency time has an average increase of 52.58%, considering that the handshake is performed between two different NGSI requests.

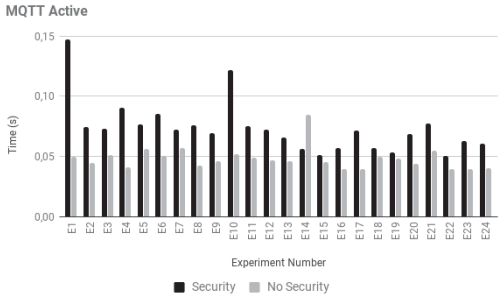

Fig. 5. Comparative Latency to MQTT *active attributes*.

Figure 6 presents the latency time when MQTT *Command* attributes are executed in the device. In this case, the latency time with the use of TLS is in average 44% higher when compared to the MQTT *command* attribute without using TLS. The average time using TLS is $0,08815 \pm 0,01311$ seconds and with no TLS the average time is $0,06119 \pm 0,00621$ seconds. The latency here is bigger when compared to the MQTT *active* attributes. This happens because to send a command to the device it is necessary to use a higher number of NGSI requests to send the *Command* to the device and retrieve the new status in the context. In this case, the MQTT *Command* attribute is 27% higher than the MQTT *active* attribute.

B. IoT Agent LWM2M/CoAP

The IoT Agent LWM2M/CoAP evaluation considered three different modes of operation supported by the LWM2M/CoAP

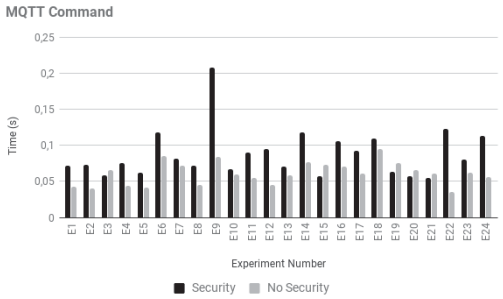

Fig. 6. Comparative latency using MQTT *Command* attributes.

protocol: *Active*, *Lazy* and *Command*. It is important to notice that all requests of the COAP protocol have an acknowledge associated. So, it is possible to generate in each scenario a retransmission of the message if it is not confirmed. In the scenario where no security service is used, it is necessary to open the *socket* before sending any COAP message and it affects the latency time. When a security service based on DLTS is used, the same *socket* defined to send the data between the device and the IoT Agent is also used to transmit the *handshake* messages.

Figure 7 presents the latency comparison in the scenario where the COAP *active* attributes are used. In this case, when an attribute is modified, updated information is sent to the IoT Agent. The result shows that the latency time is in average 31% higher when DLTS is used. The average time using DLTS is $0.02804 \pm 0,00364$ seconds and is 0.02130 ± 0.00447 seconds when no security service is used.

The COAP *Active* attribute scenario presents lower latency time when compared to MQTT *Active* attribute. In both cases, only one attribute was updated during the experiments. COAP protocol allows the update of only one attribute per message while MQTT protocol permits to update several attributes using a single protocol message. So, it is important to consider the number of attributes exchanged between the device and the IoT Gateway. When using COAP, a higher number of attributes can generate a higher UDP based traffic, while MQTT will have a lower TCP based traffic.

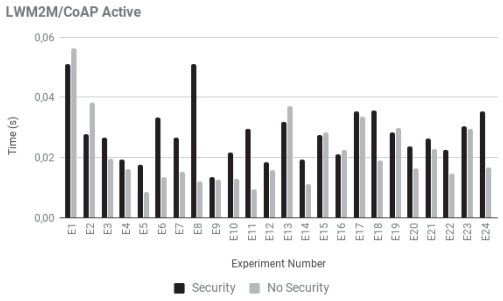

Fig. 7. Comparative Latency to LWM2M/CoAP *Active* attributes.

Figure 8 presents the latency time when using COAP *lazy* attributes. In this case, a NSGI *QueryContext* starts the process to update the value of the attribute. In this case, to each attribute, the IoT Agent will send a COAP GET message to obtain the most recent value. The use of the DTLS based security service presented a 17% higher latency time when no security was used. The average time using DTLS was 0.06006 ± 0.00551 seconds, and with no security, it was 0.05107 ± 0.00263 seconds. As the MQTT protocol does not offer a similar attribute monitoring mode, it is not possible to correlate the COAP results with MQTT.

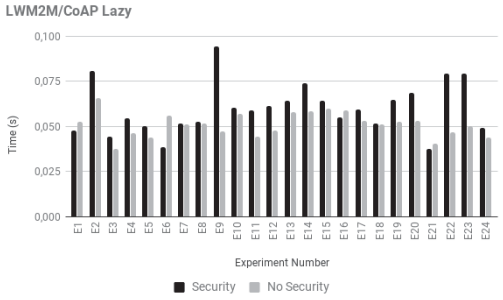

Fig. 8. Comparative Latency to LWM2M/CoAP *Lazy* attributes.

Using LWM2M/CoAP also possible to send a *Command* to the device. Figure 9 presents the comparative latency time when using the *Command* attributes. Using LWM2M/CoAP with no security the average latency time is 0.02845 ± 0.00315 seconds, while when using DLTS to secure the messages the latency time is in average 0.03444 ± 0.00221. In average, the latency time is 21% higher when using DLTS.

To this attribute mode, MQTT latency average time with TLS is 0.08815 seconds and 0.06119 seconds with no security services. When using DLTS LWM2M/CoAP presents an average latency time of 0.02845 seconds and with DLTS based security an average latency time of 0.03444 seconds. Although LWM2M/CoAP results are lower when compared to MQTT, if the number of attributes increases, the latency time will also increase. When using LWM2M/CoAP, the IoT Agent must wait for the response of the device and just after that send the NSGI messages.

The support of security services has its onus and also its bonus. When using TLS and DLTS to support security services, it is possible to have confidentially, avoiding traffic analysis. The deployment of security in a real environment has some associated challenges. In a deployment with an enormous amount of devices, it will be necessary to have a mechanism to manage the certificate lifecycles. Another issue is the resources available in the devices in other to support the extra processing power required by the security services in a class of devices that must use low power consumption. Another type of issue is network failures that may bring a problem to execute the protocol *handshake* to establish the secure channel. All these problems are beyond the scope of this work.

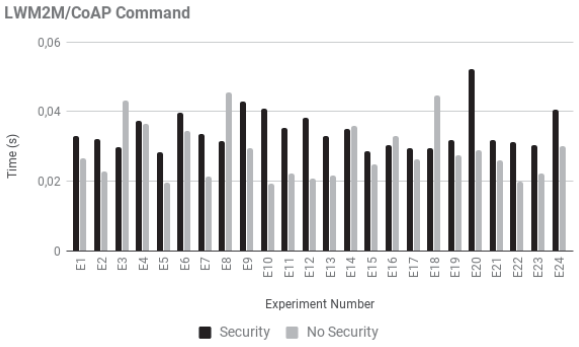

Fig. 9. Comparative Latency to LWM2M/CoAP *Command* attributes.

VII. Concluding Remarks and Future Work

The main contribution of this work is to provide packet encryption in the FIWARE Platform by implementing the DTLS 1.2 NodeJs in the LWM2M/CoAP protocol. We also validated our proposal by using LESHAN which is an OMA Lightweight M2M server and client Java implementation.

MQTT security features have also been incorporated into IoT Agent, by allowing the device to send encrypted messages. The IoT Agent has been refactored so that the authentication and authorization of IoT Agents could be performed on all NGSI requests. The access token of the requests from the Orion Context Broker is also checked. Another contribution is to verify the security cost of the MQTT and LWM2M/CoAP end-to-end protocols in the FIWARE platform.

IoT Broker-FI was developed to be a point of communication between the IoT Agents and the Orion Context Broker. This application performs the mapping of entities with information. There was also a security gap between the IoT Agents and the NEC IoT Broker since the NGSI requests were not encrypted. For this, encryption was implemented in NGSI messages and access control to resources.

Finally, the mechanisms developed were made available to the researchers involved with FIWARE and the entire scientific community and developers [31]. Thus, this work allows other developers to extend and enhance security throughout FIWARE architecture.

Acknowledgment

This work has been partially funded by the National Council for Scientific and Technological Development (CNPq) under the CNPq RHAE Program and by PROPP/UFU.

References

[1] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for internet of things: a survey," *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 70–95, 2016.

[2] L. Tan and N. Wang, "Future internet: The internet of things," in *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, vol. 5, Aug 2010, pp. V5–376–V5–380.

[3] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[4] H. Schaffers, N. Komninos, M. Pallot, B. Trousse, M. Nilsson, and A. Oliveira, "Smart cities and the future internet: Towards cooperation frameworks for open innovation," in *The Future Internet Assembly*. Springer, 2011, pp. 431–446.

[5] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, "Future internet: The internet of things architecture, possible applications and key challenges," in *2012 10th International Conference on Frontiers of Information Technology*, Dec 2012, pp. 257–260.

[6] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelfflé, "Vision and challenges for realising the internet of things," 2010.

[7] (2017, jan) Fiware. [Online]. Available: https://www.fiware.org/

[8] F. catalogue. (20017, jul) Backend device management - idas. [Online]. Available: https://catalogue.fiware.org/enablers/backend-device-management-idas

[9] OMA. (2017, jan) Oma specifications. [Online]. Available: http://www.openmobilealliance.org/wp/

[10] FIWARE, "FIWARE OpenSpecification IoT Backend Device Management R5," Oct. 2016. [Online]. Available: https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSpecification.IoT.Backend.DeviceManagement_R5#IoT_Agent:_LWM2M.2FCoAP

[11] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[12] Amazon. (2017, aug) Aws iot. [Online]. Available: https://aws.amazon.com/pt/iot-platform/how-it-works/

[13] ——. (2017, jul) Security and identity for aws iot. [Online]. Available: http://docs.aws.amazon.com/iot/latest/developerguide/iot-security-identity.html

[14] KaaIoT. (2017, aug) Kaa project. [Online]. Available: www.kaaproject.org

[15] IBM. (2017, aug) Ibm watson iot platform. [Online]. Available: https://www.ibm.com/cloud-computing/bluemix/pt/internet-of-things

[16] Microsoft. (2017, aug) Azure iot hub. [Online]. Available: https://azure.microsoft.com/pt-br/services/iot-hub/

[17] F. Catalogue. (2017, may) Iot broker nec. [Online]. Available: https://catalogue.fiware.org/enablers/iot-broker

[18] D. M. Mendez, I. Papapanagiotou, and B. Yang, "Internet of things: Survey on security and privacy," *arXiv preprint arXiv:1707.01879*, 2017.

[19] T. Dierks, "The transport layer security (tls) protocol version 1.2," 2008.

[20] R. Seggelmann, "Datagram transport layer security," Proceedings of the 17th LinuxTag 2011, Tech. Rep., 2011.

[21] V. Gazis, M. Görtz, M. Huber, A. Leonardi, K. Mathioudakis, A. Wiesmaier, F. Zeiger, and E. Vasilomanolakis, "A survey of technologies for the internet of things," in *2015 International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, 2015, pp. 1090–1095.

[22] R. J. C. Raphael J Cohn, "Mqtt version 3.1.1," OASIS, Tech. Rep., dec 2015.

[23] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Sensing as a service model for smart cities supported by internet of things," *Transactions on Emerging Telecommunications Technologies*, vol. 25, no. 1, pp. 81–93, 2014.

[24] O. M. Alliance. (2017, jul) Lightweight machine to machine requirements. [Online]. Available: http://www.openmobilealliance.org/release/LightweightM2M/

[25] G. FIWARE. (2017, apr) Oma lightweight m2m applications. [Online]. Available: https://github.com/telefonicaid/lwm2m-node-lib/

[26] O. M. Alliance. (2017, jul) Lightweight machine to machine technical specification. [Online]. Available: http://openmobilealliance.org/release/LightweightM2M/V1_0-20151214-C/OMA-TS-LightweightM2M-V1_0-20151214-C.pdf

[27] Neustar. (2017, jul) Implementation coap extended to use dtls. [Online]. Available: https://github.com/neustar/node-coap-dtls

[28] E. Rescorla and N. Modadugu, "Datagram transport layer security version 1.2," 2012.

[29] E. Leshan. (2017, jul) Oma lightweight m2m (lwm2m) implementation in java. [Online]. Available: https://github.com/eclipse/leshan

[30] S. Rao, D. Chendanda, C. Deshpande, and V. Lakkundi, "Implementing lwm2m in constrained iot devices," in *Wireless Sensors (ICWiSe), 2015 IEEE Conference on*. IEEE, 2015, pp. 52–57.

[31] C. Thomas, "Security Improved FIWARE IoT Components," Apr. 2017. [Online]. Available: https://github.com/caiothomas