Enforcement of Security Policy Rules for the Internet of Things

Ricardo Neisse, Gary Steri, Gianmarco Baldini
European Commission Joint Research Centre, Ispra, Italy,
{ricardo.neisse, gary.steri, gianmarco.baldini}@jrc.ec.europa.eu

***Abstract*—According to the European Union data protection legislation, privacy is a fundamental right that should be protected in the interaction of the citizen with the digital world. In the evolution of Internet towards new paradigms like Internet of Things (IoT), protection of privacy can be a challenging task because IoT connected objects can generate an enormous amount of data, some of which actually constitute personal data. In addition, it is difficult to control the flow of data when there is no user interface or adequate tools for the user. In this paper we describe an efficient solution to enforcement security policy rules that addresses this challenge, and takes a more general enterprise architecture approach for security and privacy engineering in IoT. This enforcement solution is based on a Model-based Security Toolkit named SecKit, and its integration with the MQ Telemetry Transport (MQTT) protocol layer, which is a widely adopted technology to enable the communication between IoT devices. In this paper, we describe the motivation and design of our enforcement solution, demonstrating its feasibility and the performance results in a case study.**

***Keywords*—Security; Enforcement; Internet of Things**

I. Introduction

The Internet of Things is a new paradigm that according to [1] *links the objects of the real world with the virtual world, thus enabling anytime, anyplace connectivity for anything and not only for anyone. It refers to a world where physical objects and beings, as well as virtual data and environments, all interact with each other in the same space and time*. Fundamental to this definition is the capability of these *things* to efficiently communicate and exchange information in order to cooperatively provide useful services. On the other side of the coin, IoT connected objects like wearable sensors, health monitoring equipment, connected cars and others examples can generate an enormous amount of data, some of which actually constitute personal data. There are significant issues related to providing control to the users on the distribution of their data through IoT connections and middleware. Adequate mechanisms should be put in place to control the flow of data and to enforce policies implementing existing regulations and users' preferences. Such mechanisms should be flexible to support the range of technologies used in IoT infrastructures and the various contexts (e.g., home, office) where user can operate.

The research results described in this paper were developed in the EU funded iCore project [2], which is focused on the definition of a cognitive framework to support the IoT paradigm and adopts the Message Queue Telemetry Transport (MQTT) protocol [3] to enable the communication in the IoT context. MQTT is a lightweight publish-subscribe connectivity protocol aimed to resource constraint devices such as mobile phones and low power embedded sensors. In the IoT context, MQTT is widely used to enable the communication between devices using a publish-subscribe messaging approach. Clients in MQTT exchange messages using a broker by means of publications and subscriptions to a *topic*. Specifically in the iCore project, MQTT has been widely adopted to realize the IoT concept and support the interaction between services and IoT devices and systems, which are respectively represented through abstractions called Virtual Objects (VO) and Composite Virtual Objects (CVO). Additional details on the VO and CVO concepts are provided in [2]. In this paper, we focus on privacy and data protection requirements for IoT connectivity.

Problem. The problem we address in this paper is the lack of security policy enforcement capabilities in existing MQTT implementations to address the privacy and data protection requirements of IoT scenarios. The MQTT standard and existing implementations provide only support for authentication and simple authorization policies considering only the subscription of clients to message topics. These policies can be specified to allow or deny subscription and publication of messages by clients to specific topics. This is not sufficient in IoT scenarios where data anonymization, obfuscation, or dynamic context-based policies are required and should be evaluated dynamically for each message forwarded by the broker.

Solution. Our solution consists of a Model-based Security Toolkit named *SecKit* [4][5] to address security aspects of the IoT systems including privacy and data protection requirements. SecKit supports integrated modeling of the IoT system *design* and *runtime* viewpoints in order to provide integrated specification of security requirements, risk management, and usage control policy specification [6][7]. Our focus in this paper is on the runtime support of SecKit for enforcement of expressive security policy rules at the MQTT broker level.

Contribution. Our contribution is an extended implementation of an open source MQTT broker that has been integrated with SecKit, and is capable of enforcing the expressive security policies required. We describe in detail the motivation of our security policies, the design of the enforcement solution, and also show in a case study the evaluation of our implementation using real IoT devices (i.e., positioning platforms). Our implementation consists of the SecKit general purpose components and a modified version of the Mosquitto open source MQTT

978-1-4799-5041-6/14/$31.00 ©2014 IEEE

broker [8]. Mosquitto has been chosen because in contrast to other open and closed source MQTT implementations it has the most advanced and extensible policy enforcement support, it has a very small and lightweight open source footprint, which is particularly important in IoT devices, with limited processing and memory capabilities. Mosquitto is also the platform choice of many partners in the iCore project.

The paper is organized as follows. Section II describes the MQTT protocol and architecture, including the non-standardized security capabilities of the Mosquitto broker, and motivates the privacy and data protection requirements provided by our solution. Section III shows the SecKit interface for security policy specification at design time. Section IV describes the MQTT policy enforcement extensions for MQTT implemented as part of the SecKit runtime components. Section V presents a case study and implementation with IoT development boards including a performance evaluation of our implementation. Section VI compares our approach with existing enforcement support for MQTT brokers. Finally, Section VII presents the conclusions and future developments.

II. MQTT Security and Requirements

Figure 1 shows a simplified behavior diagram of an IoT system using MQTT. In this behavior model, an IoT System consists of clients and brokers instantiations that interact with the final goal of enabling clients to exchange messages using a publish-subscribe pattern. Clients may publish or subscribe messages to *topics*, which are multi-level structures separated by a forward slash similar to a directory structure. An example of a topic for publishing GPS location information of an IoT device could be $gps/deviceId$.

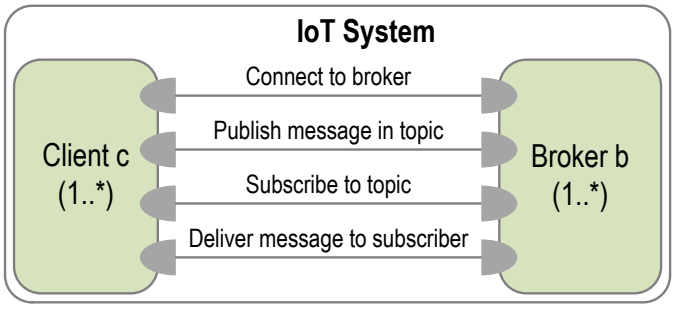

Fig. 1. MQTT behavior model

Messages can be published with a Quality of Service (QoS) parameter indicating that a message should be delivered "at most once", "at least once" and "exactly once". MQTT also supports persistence of messages to be delivered to future clients that subscribe to a topic and *will* messages that are configured to be sent in specified topics when the client connection is closed abruptly. Finally, MQTT also implements keep alive messages, by means of ping request/response that are not shown in Figure 1.

The MQTT V3.1 Protocol Specification [3] does not define any security management function in addition to a plain username/password authentication embedded in the connect packet. The public review draft of MQTT V3.1.1 [9] includes a chapter with guidance only about threats and security mechanisms that should be provided by MQTT implementations. However, each MQTT implementation is free to implement or provide their own non standard version of security functions for authentication, authorization, integrity and privacy. The technical security checklist provided in the public review draft include: mutual authentication of client and servers, integrity/privacy of messages and control packages, non-repudiation of messages, and detecting malicious and abnormal behaviors of clients/servers.

Mosquitto [8] is a widely adopted open source MQTT message broker that implements version 3.1.1 of the protocol, and it is the target of the work proposed in this paper. The non-standardized security function provided by Mosquitto is nearly the same as the support provided by other open/closed source implementations (see Section VI). We performed an analysis of the authentication and usage control configuration options in the Mosquitto MQTT broker implementation from a data protection and privacy perspective.

Mosquitto can be configured to allow connection by anonymous unauthenticated clients identified by a client id string, authentication using a username and password combination, and mutual client and broker authentication using Public-Key (PK) cryptography. In the configuration file it is possible to specify the following authentication options:

- Anonymous: no authentication, identifier (id) is provided arbitrarily by client;
- Username/password: access control list with allowed clients to connect;
- Certificates (SSL/TLS): client Common Name (CN) from the provided certificate is used as username for access control list. The connection with the broker is only encrypted with this authentication option, for the other options connections are in plain.

The Mosquitto broker provides access control functions using static Access Control Lists (ACLs) with the possibility of giving users read and write permissions to topics. Control is limited to allow or deny subscription/publication to a topic and it is possible to use wild-cards referring to all topics in a determined path, including the client id as a variable in the path composition. This wild-card can be used, for example, to allow a client to read and write to topics where the path contains the client id and/or username used by the client for authentication in the connection to the broker (e.g., GPS based location or client id). A default access policy can be specified also for anonymous clients that do not provide an username when connecting to the broker.

If the access control list is not enabled, Mosquitto does not perform access control and all clients are allowed to publish and subscribe to messages in all topics. If the access control list is enabled the broker operates in a white-listing mode, where if no access is explicitly granted by the access control list the client is denied read and write to all topics. Mosquitto provides a plugin mechanism to enable custom specification

of authentication operations, and plugin implementations are provided to integrate Mosquitto with different technologies containing the user credentials and authorization permissions (e.g. MySQL databases) [10].

Mosquitto does not implement support to modify the messages being delivered, for example, to anonymize or to obfuscate part of the content for a particular client. Furthermore, in order to prevent real-time tracking and to protect the privacy of IoT system users, it may also be desirable to delay the delivery of messages.

Access control at the topic level without considering the delivery of individual messages is a major drawback of the Mosquitto authorization model. After the access to a topic subscription is granted, it is not possible to revoke it, or perform the access control considering the context or payload of individual messages. This is a problem in dynamic environments like a smart home where authorization decisions may change due to variations in the context, for example, if a smart home user is at home (s)he may decide to allow the delivery of some messages while these messages should be inhibited, anonymized, or obfuscated when (s)he is away from home.

Finally, Mosquitto does not provide a mechanism for logging of messages or for defining reaction rules. IoT device owners may be willing to be informed about certain types of behaviors even tough they do not want to deny all accesses. For example, a Smart TV may be allowed communication with the Internet in order to update its firmware once per month. In case access happens more than once a month the owner of the device should be informed to initiate further investigation about the misbehavior. This is one example of an abnormal behavior mentioned in the public review draft of MQTT V3.1.1 [9]. An important aspect is also the user consent, that must be provided and checked on a per-request basis considering the purpose of access according to the EU regulations such as the new Data Protection Regulation [11][12].

The lack of MQTT implementations that fulfill these requirements was our main motivation for the work presented in this paper. In addition, MQTT is a good example of the issues related to the deployment of middleware in large scale IoT infrastructures in Europe, where it is of fundamental importance that security and privacy requirements are fulfilled on the basis of EU regulations. The following list summarizes our contributions in contrast to the missing features in current MQTT implementations:

1) modification of messages and identity obfuscation in addition to simply allow or deny;
2) delaying of messages to prevent real-time tracking of devices and users;
3) enforcement when a message is delivered to a client in addition to enforcement when a client subscribes to a topic;
4) support for reactive rules to notify, log, or request user consent;
5) misbehavior checking rules, for Denial-of-Service (DoS) attack detection.

III. Policy Design using SecKit

For the specification of security policies at the MQTT level we propose the Model-based Security Toolkit (SecKit) [4]. The SecKit foundation is a collection of metamodels that provides the basis for security engineering tooling, add-ons, runtime components and extensions to address security, data protection and privacy requirements. In contrast to other approaches, the SecKit takes an integrated perspective to security engineering tooling and precisely specifies the relation between the security concepts and other security-relevant system concepts. The SecKit metamodels describe:

- Data: data types mapped one-to-one to the ECore metamodel of the Eclipse Modeling Framework (EMF) [13]. Data types can be specified using the EMF tooling and imported in SecKit;
- Time: time units, timestamps, time durations, and time intervals;
- Identity: identity types and attributes;
- Role: role types and hierarchy;
- Context: context information, context situations, and Quality-of-Context attributes;
- Structure: entities and interaction point mechanisms;
- Behavior: behavior and activities (actions and interactions);
- Trust: aspect-specific trust relationships;
- Rule: Event-Condition-Action (ECA) rule templates and configurations;
- Risk: assets, vulnerabilities, threats, risk, and countermeasures mapped to rules or trust relationships.

Figure 2 shows the MQTT behavior design in SecKit Graphical User Interface (GUI), which is a design of the model already presented in Figure 1. In the behavior model the behavior and activity types (actions and interactions), and activity instantiations are specified. Activity types instantiate data and identity attributes, which represent the produced results of the respective activities. Behavior types are assigned to entity types in the structural design model.

Figure 3 shows the structural design model of the iCore framework. This model specifies and IoT system, which consists of VO, CVO, and Service containers. The containers instantiate the respective types of abstractions (VOs, CVOs, and Services), and communicate using a Network interaction point with a middleware. In this specific instantiation of the IoT system the middleware is an MQTT broker, which is assigned to the Broker behavior type. The container types are assigned to the MQTT client behavior.

Figure 4 shows the context design model GUI, which specifies *Context Information* and *Context Situations* types. Context information is related to data values about an entity that is acquired at a particular moment in time. Context situations are composed data types that model a specific condition that begins

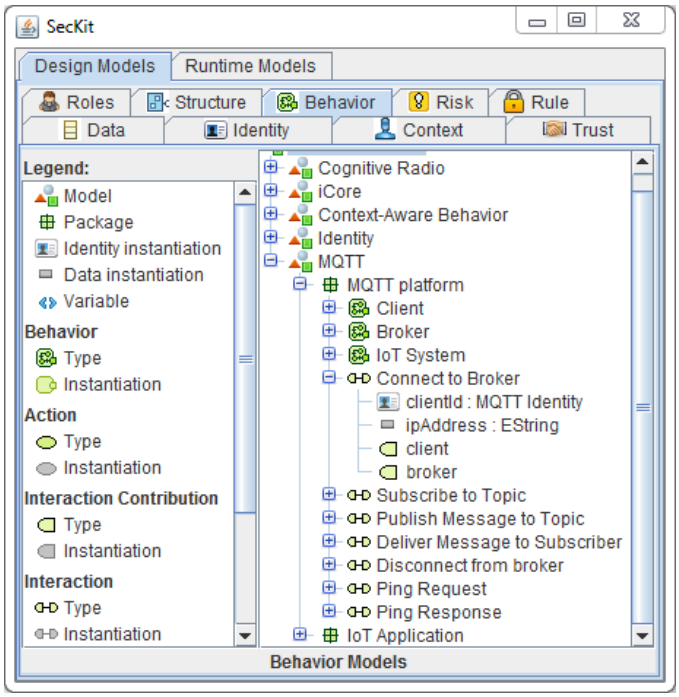

Fig. 2. SecKit GUI behavior design model

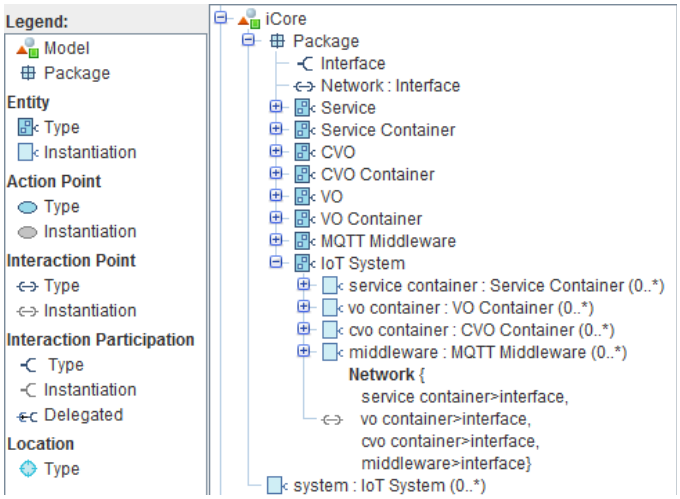

Fig. 3. iCore structural design model

and finishes at a specific moments in time [14]. For example, the GPS location is an example of a *context information type*, while *In 100 meters range* is an example of a *context situation type* where a target entity is not more than 100 meters away from a set of nearby entities with their reciprocal roles. The monitoring of context information and context situations is done in SecKit by a Context Manager component, which interfaces and periodically generates context information and situation events for the Policy Decision Point (PDP).

Authorizations and obligations are specified in SecKit by means of parametrized rule templates that must be explicitly

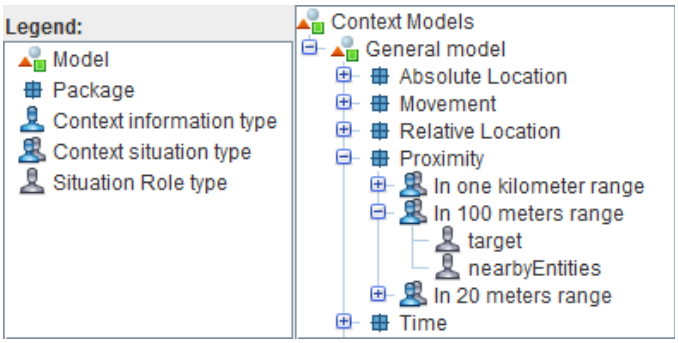

Fig. 4. Context design model

instantiated using template configurations. The rule templates follow an Event-Condition-Action (ECA) structure, whenever the event (E) is observed, and the condition (C) evaluates to true, the action (A) is performed. The event part is in fact an event pattern, which may be an activity event (action or interaction), context information and situation events, and lifecycle events for entities and data instances. Events may represent the start, ongoing, and completion of an activity or context situation.

A rule template declares variables that are assigned by the configuration, with the possibility of recursive nesting and re-use of templates and configurations. A configuration also specifies when the rule should be disposed, using the same ECA rule structure for the management of the rule lifecycle. For example, a template configuration can be specified to instantiate a set of policy rule templates when an MQTT client connects to the broker and to disposed this set of rules when the client disconnects.

For *activities*, two types of events are represented: *tentative* and *actual* events. These two types of events model an activity that is ready to start but has not yet being executed (e.g., message ready to be sent by the broker to a MQTT client), or an activity that has already completed. From an enforcement perspective, these types of events allow the specification of reactive and preventive enforcement since tentative events signal activities that can still be denied, modified or delayed. The specification of message modifications requires the modeling of the data types and identity types used in the MQTT message payload.

The condition part of a rule template consists of event pattern matching, propositional, temporal, and cardinality operators. The temporal and cardinality operators supported are: *always*, *before*; *since*, *within*, *during*, *repSince*, *repMax*, and *repLim*. The formal semantics of these operators is based on past time Linear Temporal Logic (LTL) with discrete time steps, and a detailed description of the operators is already published in [7]. The granularity of the time steps depends on the requirements of the policy, for example, if we want to evaluate that one event happens three hours before another event the time step size could be one hour.

Figure 5 shows a context-based policy rule template, that allows the deliver of messages published in the GPS topic

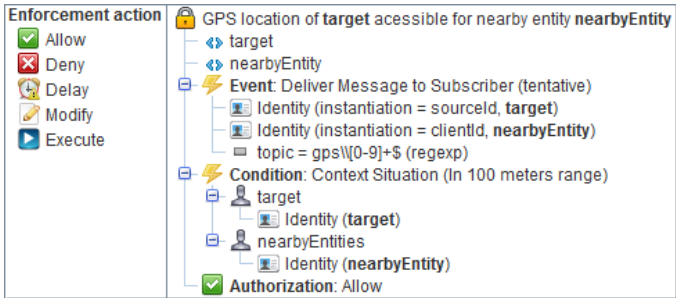

Fig. 5. Policy to allow nearby entities access to their GPS location

by a target for the nearby entities. This policy template illustrates the use of activity events, context situation events, and variables in the specification of complex rules. This template follows the ECA structure, with the event part being the tentative action of delivering a message to the subscriber. When this tentative event is observerd, and the condition of being in 100 meters range is also satisfied, the rule evaluates to true and the activity is allowed. The variables specify that client receiving the message (nearbyEntity) from the source (target) must be the same client that is detected in the 100 meters range.

Figure 6 shows another policy that illustrates a modification example, where the full name attribute of a *Verify Identity* activity is modified to the string 'anonymous'. Indeed the presented policy could also be extended to allow the IoT objects, under the supervision of the MQTT, to forge ad-hoc crafted soft-identities with which get (or provide) the access to services/messages, with the aim of minimizing the disclosure of sensitive information, according to pre-designed scenarios. In the policy of Figure 5, an additional delay could also be specified for the provisioning of GPS location or the obfuscation of GPS coordinates to enable privacy by design. The action part also allows the specification of behavior execution, for example, to log or notify users.

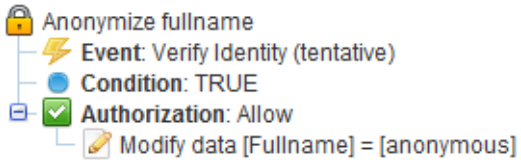

Fig. 6. Policy to anonymize the user identity

In order to be useful for concrete implementation scenarios, the SecKit must be extended with technology specific runtime monitoring components. In the iCore project we provide one extension to support monitoring and enforcement of policies for a MQTT broker. The following section describes the runtime component we have implemented and evaluated.

IV. MQTT Policy Enforcement

The policy enforcement support for MQTT consists of a custom Policy Enforcement Point (PEP) component implemented in the C language according to the SecKit interface specifications. Our PEP is a connector that intercepts the messages exchanged in the broker with a publish-subscribe mechanism, notifies these messages as events in the SecKit PDP implemented in Java, and optionally receives an enforcement action to be executed: allow, deny, modify, and delay. The PEP has been embedded in the Mosquitto broker using an already available extension mechanism called security plugin. This enforcement architecture is described in Figure 7.

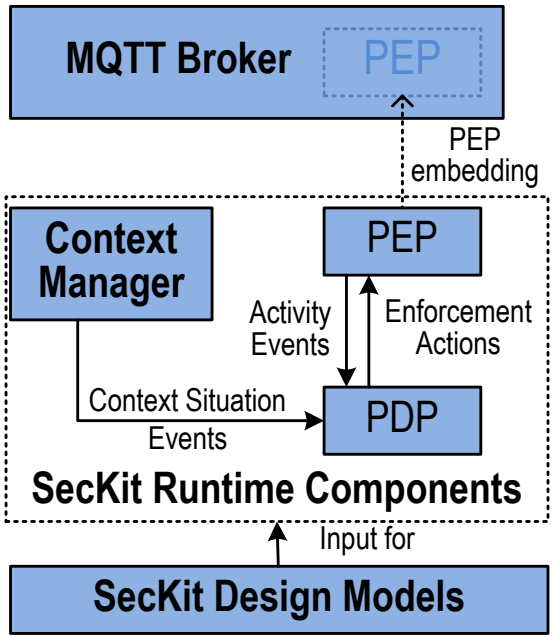

Fig. 7. Enforcement Architecture

A security plugin in Mosquitto can be embedded to implement custom authentication and authorization function in the broker to control authorization to publish (write) or to subscribe (read) to topics. We have extended the internal plugin interface mechanism already implemented to support additional checks when a client connects and when messages published are delivered to subscribers. Our extension supports additional checks of the client IP address and enforcement on the message payload when clients publish messages and when these messages are delivered to each subscriber. This additional check allows the enforcement of policies for publishers and subscribers when messages arrive and when messages leave, and not only when the subscriptions to topics are performed. In this way, we can also monitor flooding of connect messages from specific clients to identify potential Denial of Service (DoS) attacks or harmful client behavior.

By enforcing complex security policy rules inside the broker the overhead in the IoT devices is reduced, which is a positive aspect since the enforcing of these policies could lead to faster battery draining due to the processing overhead. However, this requires a trust model where the broker must be secure and under control of the users with guarantees that the deployed policies are not being circumvented in the broker. We foresee a deployment architecture where the owners of IoT

devices will manage their own brokers with help of the SecKit, and the integration with cloud services and other IoT systems will be done using bridged brokers. Bridged brokers share published messages in topics, which in our case are restricted to a set of messages on which security policies have already been enforced.

One drawback of our approach is the high overhead when one publisher has many interested subscribers, and a policy needs to be checked for every subscriber. This seems a necessary overhead considering privacy and data protection issues, since authorization policies may be different for each subscriber and message received. In summary, using our toolkit, extension policies can be enforced when a client connects, a client publishes messages in a topic, a client subscribes to a topic, the broker sends messages to subscribers, or when ping request/response messages are sent to clients.

V. Case Study

In order to evaluate our PEP implementation for MQTT we set up a case study where two IoT development boards regularly publish their GPS location in a topic. The boards should only be allowed to see each other's location when they are nearby, meaning within a 100 meters range. The SecKit Context Manager component subscribes to the GPS location of all boards. We have specified and executed this policy in our enforcement infrastructure, which is shown in Figure 8.

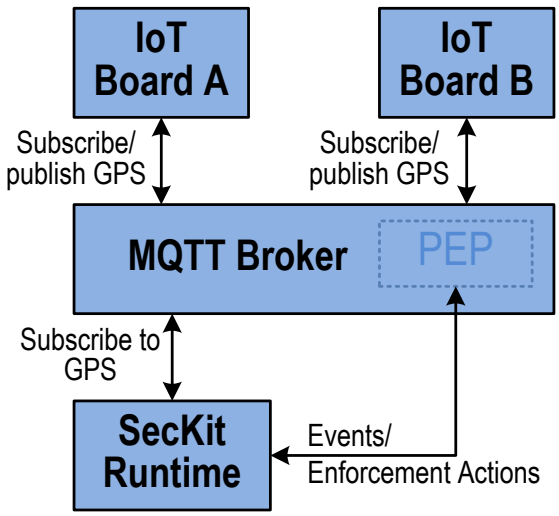

Fig. 8. Scenario Implementation

The two boards used are equipped with a 400 MHz ARM9 processor and 256 MB of DDR2 RAM, running a Linux Debian operating system (kernel 3.13.6). They have an external module which embeds different kind of sensors (e.g., accelerometers, gyroscopes, magnetometers etc.) and a GPS receiver, employed to get the position and to synchronize the clock of the boards. They reach via WiFi a base station to which is connected via Ethernet a laptop (Intel® Core i5 with Windows 7) running the SecKit enforcement components together with the modified MQTT broker and our embedded PEP.

Using the Mosquitto libraries we developed a simple Python client to read GPS positions and send them to the MQTT broker in two different topics, one for each board. When a message is sent to the broker, a timestamp (CPU time with 1 μs accuracy) is registered. The same happens when a message is received. In this way, since the boards are synchronized[1], we can calculate the exact time elapsed since the publication of a message in the broker and its delivery to a subscriber, having the possibility to evaluate the delay introduced by the PEP. In this case the message was delivered only when the policy specified (boards within a 100 meters range) was satisfied. Next subsection presents our performance evaluation results.

A. Performance Evaluation

The architecture described above moves all the load of policy enforcement to the MQTT broker, where the PEP is embedded. The end devices do not execute any extra calculations due to the enforcement. In other words, the performance of the system strictly depends on the overhead of the extra operations performed by the broker to call the PEP and, obviously, on the processing power of the machine that runs the broker.

We first measured the performance of the system with a standard setup of the MQTT broker (i.e., with the PEP disabled): in this reference configuration, published messages are immediately delivered to subscribers and the average time elapsed between sending and receiving a message was 0.515 s, with 1243 messages exchanged (maximum delay of 1 second). After having enabled the PEP, this average time increased to 0.525, that is around 10 ms of extra delay to call the enforcement point. Figure 9 shows the comparison of the averaged delay values and their evolution with and without the PEP. It can be observed a packet delay variation (jitter), which is slightly bigger when the PEP component is deployed. We monitored the processing time of MQTT messages in the broker network interface using a network sniffer and we confirmed that this jitter is due to the network layer packet propagation and is not a jitter internal of the broker message processing delay.

The delay introduced by the policy enforcement can be divided in two main components that apply for all broker activities when policies that reference these activities are defined: connect, publish, subscribe, deliver, and ping messages. The first is the more constant delay introduced by the PEP to stop the activity by the broker, to generate an event in a format understood by the PDP, to signal this event to the PDP, and to process/enforce the response by the PDP. The second part is the variable delay introduced by the PDP to evaluate the deployed policy when an event is received from the PEP, and to decide the response to be sent considering the policy evaluation result. The PDP delay is variable since it depends on the number and

[1]To check the synchronization we also did the test with the same device publishing on and subscribing for the same topic, so that the clock employed to measure sending and receiving time was the same.

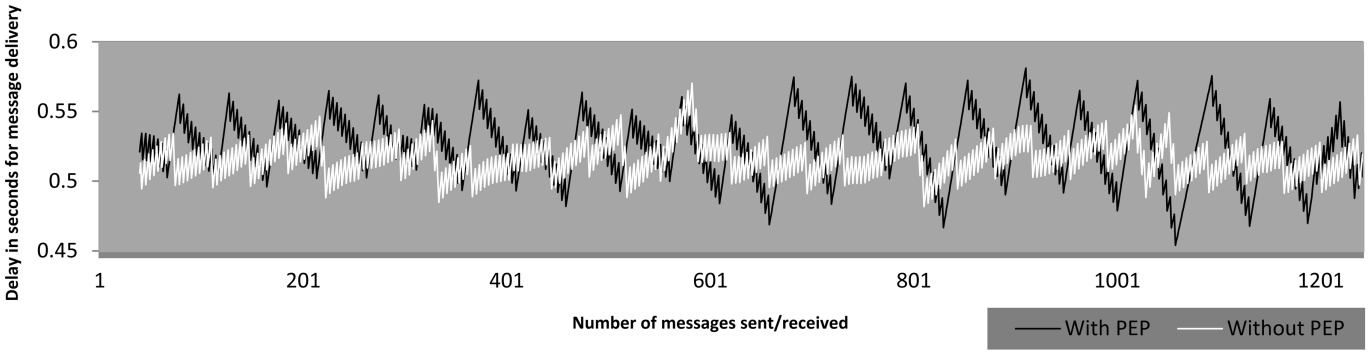

Fig. 9. Average delay with and without PEP

structural complexity of the deployed policies. Our focus on this paper is on the delay introduced by the PEP only, since the PDP performance evaluation is already presented in a previous publication [4].

In this case study we implemented a simple policy rule that does not overload the PDP and simply evaluates to *allow* when the context situation event 'In 100 meters range' is observed for the event corresponding to the activity 'Deliver Message to Subscriber'. The situation events are generated by the context manager and the evaluation of the policy has no overhead at all since the condition is a simple check for this situation event (see the policy in Figure 5). We have not specified in our case study any policy to control the publishing of the GPS location by the boards.

Our performance measurements show that the introduction of the PEP in the broker does not produce a relevant delay in the broker and that 98% of this delay is inherent to the normal functioning of the broker in a distributed environment (network protocols and devices play a very important role). The delay measured by us includes the HTTP calls to the PDP by the PEP, which was running in another machine (a standard Intel® Core i7 with Windows 7) connected via Ethernet to the broker, and the encoding of the events in JSON (JavaScript Object Notation) format. An important detail that contributed to this result is the caching of HTTP connections implemented between PEP and PDP, which avoids reconnection delays.

VI. Related Work

The OASIS MQTT Security Subcommittee has been created to provide guidance on MQTT security and integration with other security standards. In the subcommittee page [15], a set of documents are provided including meeting minutes and discussion descriptions on how MQTT solutions may be made secure, and how this may mesh with existing security standards (e.g., the NIST Cybersecurity Framework). The solution proposed by us on this paper addresses issues related to security policy specification and enforcement without considering some of the issues mentioned in these documents such as identity management architectures, device tamper-proofness, etc.

The MQTT standard does not specify security requirements and features to be supported by implementations, and many of the open source and proprietary implementations available support some set of authentication and authorization features. From a standardization perspective each implementation of MQTT broker is free to add different types of security functions. Many open source and proprietary implementations of MQTT are available, and even though it is widely adopted there is no common view of how security policies should be specified and enforced. A complete list of MQTT broker software including commercial, free, and open source is available at the MQTT.org website [16].

Some of the MQTT implementations available include HiveMQ, Active-MQ, and RabbitMQ. HiveMQ is an MQTT broker implementation focused on enterprise systems, and it is not completely open source or free for more than 25 connected clients. HyveMQ supports lightweight authentication and authorization [17] with group-based authorization and static access control rules to topics, not in a message level but in a per-topic basis. Security can also be implemented in HyveMQ using plugins [18], which are mostly implementations of very specific hard-coded security function such as logging or file authentication.

In Active-MQ [19] MQTT is supported using a mapping to the Java Messaging Service (JMS), with a JAAS mechanism for authentication, and read/write with an access control list which suffers from the same limitations of Mosquitto. In Active-MQ it is possible to implement a custom 'MessageAuthorizationPolicy', which is also a possible implementation choice for a SecKit PEP. Active-MQ was not chosen because it is not a popular choice for IoT developers due to its large general purpose disk/memory footprint and complex JMS configuration options. In fact, some enterprise service bus implementations also support MQTT like RabbitMQ [20], which also include support for many different protocols and mapping function. These brokers can also be extended and some plugins including support for the XACML [21] policy language are provided. From a enforcement perspective, our solution supports more enforcement types than XACML, which only supports the standard allow and deny, and our policy language is also more expressive with temporal and cardinality operators.

Other protocols targeting the IoT world include CoAP

(Constrained Application Protocol), XMPP (Extensible Messaging and Presence Protocol), and RESTful HTTP over TCP [22]. These protocols could also benefit from the solution presented in this paper, the only requirement would be the implementation of technology specific SecKit PEP components. From a policy specification and relevance perspective, Context-based authorizations is a current ongoing topic of discussion in the mailing list of the Authentication and Authorization for Constrained Environments (ACE) working group [23], linked to the IETF standardization working group on Constrained RESTful Environments (CoRE).

VII. Conclusions and Future Work

In this paper we presented the motivation, design, implementation, and evaluation of a solution for the enforcement of security policy rules at the MQTT layer, which can be used to support security and privacy requirements. Our solution is part of the Model-based Security Toolkit (SecKit) and has been implemented as an extension to the open source broker Mosquitto. Our performance results are promising since we are able to enforce complex security policies with a very small additional delay of 10 ms in contrast to the normal operation of the broker.

As future work we are investigating identity management aspects of IoT systems including support for identity generation and identity management policies to improve the privacy of IoT device owners. We also plan to perform a more extensive study of the performance considering different network configurations, battery constraints and introducing mobile nodes for the broker and SecKit components. This evaluation will be done as part of a pilot deployment of a smart home environment with different IoT devices communicating using MQTT.

Acknowledgments. This work was supported by the EU-funded project iCore grant agreement n. 287708. The authors would like to thank Igor Nai Fovino for his valuable comments and suggestions to improve the quality of the paper.

References

[1] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelffl, "Visions and challenges for realising the internet of things," Available at: http://bookshop.europa.eu/en/vision-and-challenges-for-realising-the-internet-of-things-pbKK3110323/, 2010, cluster of European Research Projects on the Internet-of-Things (CERP-IoT).

[2] P. Vlacheas, R. Giaffreda, V. Stavroulaki, D. Kelaidonis, V. Foteinos, G. Poulios, P. Demestichas, A. Somov, A. Biswas, and K. Moessner, "Enabling smart cities through a cognitive management framework for the internet of things," *Communications Magazine, IEEE*, vol. 51, no. 6, pp. –, 2013.

[3] IBM and Eurotech, "Mqtt v3.1 protocol specification," Available at: http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html, 2010.

[4] R. Neisse, I. N. Fovino, G. Baldini, V. Stavroulaki, P. Vlacheas, and R. Giaffreda, "A model-based security toolkit for the internet of things," *The 9th International Conference on Availability, Reliability and Security (ARES)*, 2014, available at: http://ricardo.neisse.name/images/publications/neisse-ares2014.pdf.

[5] R. Neisse and J. Doerr, "Model-based specification and refinement of usage control policies," *11th International Conference on Privacy, Security and Trust (PST)*, 2013.

[6] R. Neisse, A. Pretschner, and V. D. Giacomo, "A trustworthy usage control enforcement framework," *Proceedings 6th International Conference on Availability, Reliability and Security (ARES)*, 2011.

[7] R. Neisse, A. Pretschner, and V. D. Giacomo, "A trustworthy usage control enforcement framework," *International Journal of Mobile Computing and Multimedia Communications*, 2013.

[8] Mosquitto, "An open source mqtt v3.1/v3.1.1 broker," Available at: http://mosquitto.org, 2014.

[9] A. Banks and R. Gupta, "Mqtt version 3.1.1 - committee specification draft 01 / public review draft 01," Available at: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/csprd01/mqtt-v3.1.1-csprd01.html, 2013.

[10] J.-P. Mens, "Authentication plugin for mosquitto with multiple backends (mysql, redis, cdb, sqlite3)," Available at: https://github.com/jpmens/mosquitto-auth-plug, 2014.

[11] "European parliament - directive 95/46/ec on the protection of individuals with regard to the processing of personal data and on the free movement of such data," Available at: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:en:HTML, 1995.

[12] "European parliament - proposal for a regulation of the european parliament and of the council on the protection of individuals with regard to the processing of personal data and on the free movement of such data (general data protection regulation)," Available at: http://http://ec.europa.eu/justice/data-protection/document/review2012/com_2012_11_en.pdf, 2014.

[13] "Eclipse foundation - eclipse modeling framework project (emf)," Available at: https://www.eclipse.org/modeling/emf, 2014.

[14] P. D. Costa, I. T. Mielke, I. Pereira, and J. P. A. Almeida, "A model-driven approach to situations: Situation modeling and rule-based situation detection," in *EDOC*. IEEE, 2012, pp. 154–163.

[15] G. Brown, "Oasis mqtt security subcommittee page," Available at: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt-security, 2014.

[16] MQTT.org, "Mqtt software - brokers," Available at: http://mqtt.org/wiki/doku.php/brokers, 2014.

[17] HiveMQ, "Lightweight authentication and authorization for mqtt with stormpath," Available at: http://www.hivemq.com/lightweight-authentication-authorization-mqtt-stormpath/, 2014.

[18] HiveMQ, "Security plug-ins," Available at: http://www.hivemq.com/plugins/security/, 2014.

[19] Apache, "Activemq - connectivity, protocols, mqtt," Available at: http://activemq.apache.org/mqtt.html, 2014.

[20] RabbitMQ, "Messaging that just works," Available at: http://www.rabbitmq.com, 2014.

[21] E. Rissanen, "extensible access control markup language v3.0," Available at: http://docs.oasis-open.org, 2010.

[22] P. Duffy, "Beyond mqtt: A cisco view on iot protocols," Available at: http://blogs.cisco.com/ioe/beyond-mqtt-a-cisco-view-on-iot-protocols/, 2013.

[23] IEFT, "Constrained restful environments working group (core wg) - authentication and authorization for constrained environments (ace) mailing list," Available at: http://www.ietf.org/mail-archive/web/ace/current/maillist.html, 2014.