Improving Internet of Things Device Certification with Policy-based Management

Ricardo Neisse*, Gianmarco Baldini*, Gary Steri*, Abbas Ahmad†§, Elizabeta Fourneret‡, and Bruno Legeard‡§
*European Commission - Joint Research Centre, Ispra, Italy
{ricardo.neisse, gianmarco.baldini, gary.steri}@ec.europa.eu
†Easy Global Market, Sophia Antipolis, France
abbas.ahmad@eglobalmark.com
‡Smartesting Solutions & Services, Besançon, France
{elizabeta.fourneret, bruno.legeard}@smartesting.com
§DISC, Institut FEMTO ST/University of Bourgogne Franche-Comté, Besançon France

***Abstract*—The fast growing rate of the IoT systems with strong pressure to put devices on the market as soon as possible makes these systems vulnerable targets for cyber criminals, as recently seen in the Mirai botnet Distributed Denial-of-Service (DDoS) attack. A way to mitigate these threats is to enforce a comprehensive security certification process of IoT devices based on common standards. In this paper, we present an approach to improve certification of IoT devices using a combination of model-based testing and policy-based management in order to detect post certification vulnerabilities and act on them by introducing runtime policy enforcement capabilities. More precisely, we address these attacks using policy enforcement in order to correct vulnerable IoT device behavior and protect users even if security and privacy were not properly addressed by the device manufactures. We describe the details of our approach and, focusing on authorization vulnerabilities, we present a case study for the oneM2M standard showing how our solution can be applied in practice.**

***Keywords*-security, certification, model-based testing, policy-based management, Internet of Things**

I. Introduction

The deployment of pervasive connected devices with embedded sensors and actuators is driving the development of the Internet of Things (IoT). Even if IoT holds the promise of supporting new business models, increase the efficiency of many business applications, and enrich the life of the citizen with new services, the risk to undermine the privacy of the individuals is even greater than in the current Internet context. For example, mobile devices with sensors capability carried by citizens will generate an increasing amount of data about the environment and the citizens themselves. As a consequence, security and privacy issues of IoT have recently received considerable attention from the industry and research communities.

The current state of affairs is even more threatening since, due to the market pressure to make IoT devices available to customers as soon as possible, device manufacturers do not put high priority on implementation of solutions to support security and privacy requirements in their products. In this context, the ARMOUR project proposes a security certification process for large scale IoT deployments. Tested and certified IoT systems are needed to address security, privacy, and safety issues considering the intrinsic uncertainty due to the interactions between devices, platforms, cloud services, and humans. Recently, also the International Computer Security Association (ICSA) has presented a security testing framework for IoT [1]. However, a major issue for IoT systems is that security certification often fails to deal satisfactorily with systems that are patched frequently, as operating systems in mobile devices now are [2]. In addition, due to considerable market pressure, security and privacy issues are not considered as a main goal in the design and deployment, and usually they are not properly addressed.

A clear evidence of this fact is the Mirai IoT Botnet [3], which infected tens of millions of vulnerable IoT devices that were used in a DDoS attack to major websites including Amazon, Spotify, and Twitter. Soon after the attack the source code of the Mirai Botnet was relasead as open source[1], disclosing the scanner code that identified vulnerable devices to be infected with the malware. The main target devices were IP cameras, DVR recorders, routers, and printers, while the main vulnerability exploited was related to the authentication of users, for example, due to factory default usernames and passwords, which allowed the attackers to deploy dormant malware code (a.k.a. zombie malware).

Apart from this specific example, IoT applications could also be composed by IoT products, which are not security certified. These products could become the vulnerability of the overall IoT application even if it is mostly built on security certified products. Furthermore, security IoT certification may not include the testing of zero-day vulnerabilities and threats, which were not know at the time of security certification. A complementary (rather than alternative) approach to support the IoT lifecycle of products and to mitigate the issues of static security certification is to introduce post-market (and post certification) monitoring of IoT devices. In this approach, a monitoring system is set up to collect data (management data or traffic data), which can be used to identify security threats. This approach is not a new concept; actually, fault management or misbehaviour detection system in ICT based infrastructures (e.g., energy, telecommunication) had fulfilled a similar role

[1] https://github.com/jgamblin/Mirai-Source-Code

978-1-5090-5873-0/17/$31.00

for many dozens of years. Recent analysis of security and privacy aspects in IoT have highlighted the possibility to use monitoring solutions and capabilities [4], to enhance the overall security of IoT deployment. The challenging aspects (as reported by [4] and others) are the scalability and heterogeneity of IoT deployments, which can reach thousands of devices with different technologies or data format. From a semantically point of view, it is also difficult to compare set of data from different IoT devices. Still, in some context like the automotive and the industry sectors where the operational requirements are usually coherent and similar across devices, the deployment of such monitoring systems could be more effective.

The drafting of this paper was motivated by the need to mitigate the risk of future attacks similar to the Mirai IoT Botnet case [3] and to propose a post certification monitoring approach. The approach described in this paper integrates the security certification of IoT devices together with a post-certification monitoring approach based on a policy-based management framework. In a nutshell, our approach combines the certification process defined in the ARMOUR project [5] with a policy-based security management framework proposed in the iCore project [6] that is able to enforce policies and correct vulnerable system behavior in case the IoT system fails a subset of the test cases. In our integrated approach the security functional behavior is tested using test cases derived automatically using a Model-Based Testing (MBT) approach and executed using Testing and Test Control Notation version 3 (TTCN3). With this combination, our approach is able to cope with the current situation where a large number of vulnerable devices are already deployed and do not conform with an adequate certification level.

This paper is organized as follows. Section II describes our integrated approach for a certification process using policy-based management. Section III shows a practical application of our approach considering a scenario inspired in the Mirai Botnet vulnerabilities using the oneM2M standard. Section IV presents the conclusions and future work.

II. Integrated Approach

In this paper we propose to integrate the ARMOUR security testing and certification process with a policy-based approach for specification and enforcement of security policies developed in the context of the iCore project. An overview of the integrated process in described in Figure 1.

The top part of Figure 1 above the dotted line describes the ARMOUR certification process, which is composed of the following main phases:

1) **Risk Analysis and Labelling**: a risk analysis is performed for a specific domain or set of applications where the IoT device must operate with a specific Level of Assurance (LoA) associated to a label;

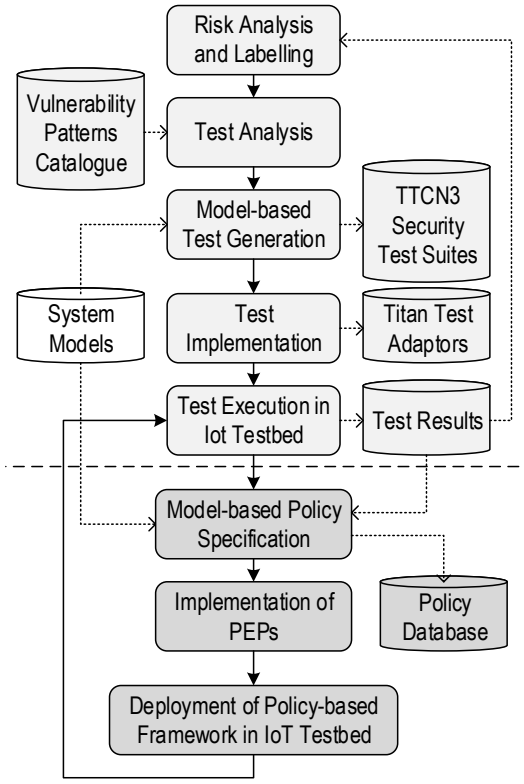

Figure 1. Integrated approach for improved security certification using policy-based management

2) **Test Analysis**: the potential vulnerabilities are identified for a specific context (i.e., single domain or set of applications) from a *vulnerability patterns catalogue*, which must be addressed and evaluated in the subsequent testing phases. The identified vulnerability patterns are used as input to the test generation;
3) **Model-based Test Generation**: TTCN3 security test suites are generated for the IoT system and domain using a model-based approach. Tests consider the security functional testing, vulnerability testing, and behavioral fuzzing. The models formalize the system's behavior, structure, and the *security/vulnerability test patterns* that define the test procedure;
4) **Test Implementation**: the test suites are executed in a specific operating environment using test adaptors, which are needed to support the execution of the tests because each IoT device or platform will have its own specific interfaces;
5) **Test Execution in IoT Testbed**: the tests are executed on a local or external large-scale IoT testbed and the results are collected.

The bottom part of Figure 1 below the dotted line describes the iCore policy-based management process, which is composed of the following main phases:

1) **Model-based Policy Specification**: using the test results as input policies are specified to correct the security functional behavior or identified vulnerabilities. The specification of policies is done using the system models as input in order to identify the system behavior elements (e.g., activities and attributes);
2) **Implementation of PEPs**: Policy Enforcement Point (PEP) components are specified for the respective technology considering the system specific interfaces and instrumentation requirements of the specified policies. One example could be a custom security plugin for an MQTT broker like the one proposed by some of the co-authors in [7];
3) **Deployment of Policy-based Framework in IoT Testbed**: policies and framework consisting of runtime policy evaluation and enforcement components are deployed in the IoT Testbed.

After the deployment of the framework and policies in the IoT testbed the generated tests can be re-executed and the incorrect security functional behavior and vulnerabilities should now be fixed. In the following subsections we concentrate in the details of the two main elements of this process, the model-based approaches for test generation and policy specification.

A. Model-based Test Generation

The generation of tests in our integrated approach is based on a MBT approach, which has shown its benefits and usefulness for systematic compliance testing of systems that undergo specific standards that define the functional and security requirements of the system [8]. This approach is based on two main modules:

1) the MBT CertifyIt technology that generates tests based on the UML/OCL models and Test Purpose (TP) formalism [9];
2) the TTCN3 test cases, generated from the MBT model, that are executed using TITAN;

The TP language used to guide the test case generation is used to express functional security requirements and security test patterns. The TP language itself is based on regular expressions and allows the test engineer to conceive its scenarios in terms of states to be reached and operations to be called. The language relies on combining keywords to produce expressions that are both powerful and easy to be read by a test engineer. The syntax of the language makes it possible to design test purposes as a sequence of quantifiers or blocks, each block being composed of a set of operations (possibly iterated at least once, or many times) and aiming at reaching a given target (a specific state, the activation of a given operation, etc.).

The generated test cases from the MBT model use the TTCN3 language, which has been widely used for many years to test large communication systems [10]. They contain modules that make possible the test execution as they define the type, port and component declarations and definitions, templates, functions, test cases and control part for executing the test cases. The control part can be seen as the main function in other programming languages as C or Java. [11].

With the combination of MBT and TTCN tests can be automated to allow a faster and more uniform testing execution. Tests and security requirements can be formally defined, and they can be used to support harmonization of the tests for security certification. It also allows to keep an overall traceability of the executed tests against the specification.

The structure of the system is modeled by Unified Modelling Language (UML) class diagrams, while the system's behavior is expressed in Object Constraint Language (OCL), using the CertifyIt tool [12]. Functional tests are obtained by applying a structural coverage of the OCL code describing the operations of the IoT system under test. This approach in the context of security testing is complemented by dynamic TP test selection criteria that make it possible to generate additional tests that would not be produced by a structural test selection criterion, for instance, misuse of the system (Model-Based Security Functional Testing) and vulnerability tests, trying to bypass existing security mechanisms (Model-Based Vulnerability Testing).

B. Model-based Policy Specification and Enforcement

The specification of the security policies is done using the Model-based Security Toolkit (SecKit) [6] already sucessfully applied in the IoT domain [6]. The SecKit is an integrated holistic approach for security engineering that defines a collection of metamodels for system design, including security aspects, and runtime components that instantiate these models to manage the system security using a policy-based approach. The system design model considers the structure, behavior, data types, identities, context, rules, trust relationships, threat scenarios for risk analysis, and enforceable countermeasures defined using policy rules to address the identified threats.

The runtime components dealing with policy rule evaluation and enforcement are respectively the Policy Decision Point (PDP), and technology specific Policy Enforcement Points (PEPs). In order to provide comprehensive security guarantees PEPs can be deployed at multiple levels of abstraction, for example, at the network layer, at the middlware layer, at the operating system layer, etc.

Policies are specified in the SecKit policy language using security mechanisms following an Event-Condition-Action (ECA) format: whenever the Event is observed and the Condition evaluates to true, the Action is executed. Events observe activities defined in the system models including actual and tentative modalities, which allows both detective and preventive enforcement mechanisms to be specified. A preventive mechanism is defined considering a tentative

event, for example, just before an activity is executed in the system the mechanism may specify the activity should be allowed, denied, modified, or delayed A detective mechanism is defined considering an actual event allowing the specification of reactive enforcement behavior, for example, after an user logins from an unusual location or time of the day a mechanism may specify that an SMS should be sent to the user's mobile phone. In summary, the action part of mechanisms may specify execution of activities and enforcement actions to allow, deny, modify, or delay the execution of an activity. The Condition part of a mechanism supports the specification of complex conditions including event matching operators for activities, trust patterns, role-based, identity-based matching, propositional, temporal, and cardinality operators. We omit in this paper the details about all the operators and features and we refer the reader to a previous publication for details [6].

An important feature of the SecKit that is particular useful in the context of the approach of this paper is the modular support for configurable mechanism templates using variables. Mechanism templates allow the definition of mechanism instantiation and disposal logic specifying when a mechanism should be activated and removed from the runtime system. Furthermore, patterns of enforcement that are needed in multiple situations or that should be matched for different system resources can be re-used and instantiated multiple times with different parameters. In the case study described in the following Section we illustrate the usefulness of a configurable mechanism in the context of the running example described.

III. Case Study

In this section we describe a case study showing the practical application of our integrated framework in a scenario designed using the oneM2M IoT standard. oneM2M is the leading global standardization body for Machine to Machine (M2M) and the IoT. The goal of oneM2M is to develop technical specifications to address the need for a common M2M service layer that can be readily embedded within various hardware and software. The purpose of these specifications is to provide interoperability that can be relied upon to connect a myriad of devices in the field with M2M application servers worldwide. The oneM2M standard proposes to use Access Control Policy (ACP) to regulate operations in resources of application entities, for example, to create, update, or delete a resource.

In the oneM2M specification the authorization function is responsible for controlling access to services and data to authenticated entities according to the provisioned security policies and assigned roles [13]. Security policies are defined as sets of conditions that regulate whether entities are allowed access to a protected resource or not. With respect to the security policy language and enforcement mechanism the standard allows multiple choices including Access Control List (ACL) and Role Based Access Control (RBAC). In our approach we adopt the SecKit security policy language introduced in the previous section.

In our case study we model the IoT system and security functionalities according to the specifications proposed by oneM2M. Figure 2 illustrates a simplified view of the class diagram representing the oneM2M model related to the security policy operations. In this model the `AE` (Application Entity) hosted by the `IUT` (IoT Platform Under Test) interacts with the `IUT` using request primitives such as SendCreateRequest, SendRetrieveRequest, etc. The request primitive could be concerned with different resource types, for example, with the security policy (`ACP`), Common Services Entity base type (`CSEBase`), or subscriptions of an `AE`, which are all defined in the `TARGET` enumeration.

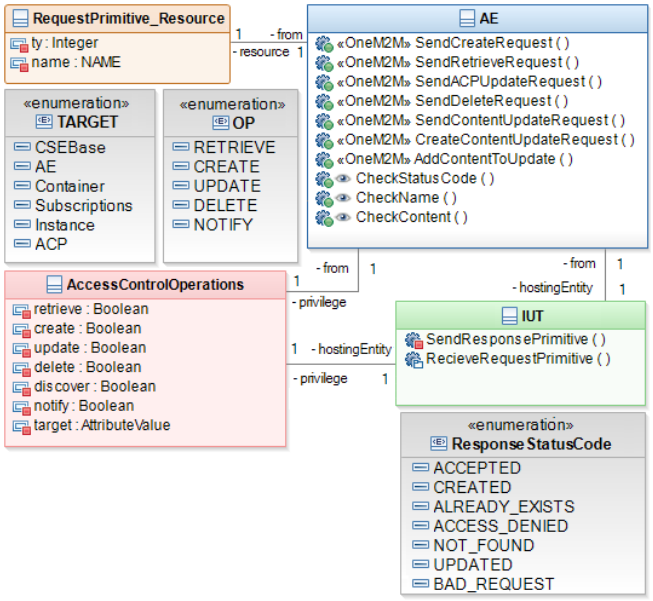

Figure 2. Simplified view of oneM2M standard model

More specifically, the `AE` in oneM2M detains a set of privileges to its resources considering the `AccessControlOperations` class. The different resources operations are supported, represented by the enumeration entity `OP`. The purpose of the security policy is to support the specification of the rules controlling the resource operations performed by the `AE` in the `IUT`. The possible responses by the `IUT` to a request primitive after the security policy is evaluated are represented by the `ResponseStatusCode` enumeration. For example, if an `AE` sends a create request for a `Container` to an `IUT` and the security policy does not allow the operation the expected response is `ACCESS_DENIED`.

The generation of security policy tests from the system model are driven by an embedded Test Purpose (TP) definition, which is illustrated in Figure 3. The TP formalizes the testing objectives for all the defined require-

ments. The particular TP illustrated in the figure is named `TP/oneM2M/CSE/DMR/CRE/BV/004` according to the naming conventions for oneM2M that includes the description of its nature. The naming convention specifies it is a Common Service Entity (CSE), Data Management Repository (DMR), CREate (CRE), and Behavior Value (BV).

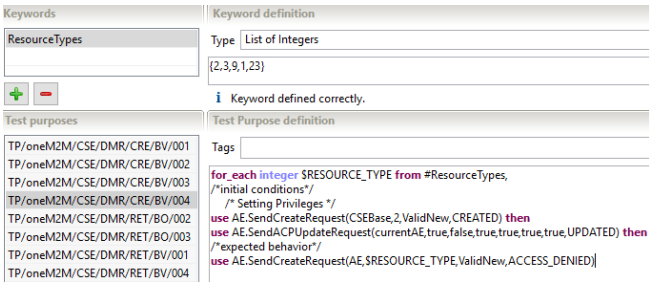

Figure 3. Test Purpose definition

This security policy TP defines the testing scenario where the creation rights are removed after a resource is created, which corresponds to the requirements described in Section 10.1.1.1 of the oneM2M functional architecture specification [14]. This TP defines a common test scenario for each Resource Type represented as a list of integers. The resource types specified in the list ResourceTypes correspond respectively to the following oneM2M resources: Application Entity (`AE`, 2), container (3), group (9), AcessControlPolicy (1), and subscription (23). More concretely, the TP guides the test generation engine to produce test cases specific for testing the security policy previously described. To test the correct implementation of the policy, for each resource type, it creates the resource, updates the policy by setting the privileges for the resource (without create privilege, second attribute set to false in the `AE.SendACPUpdateRequest` call) and then tries to make a `CREATE` request, which should be denied as the resource does not have the required privileges (expected return code `ACCESS_DENIED` in the `AE.SendCreateRequest`). When executing the test any other return code indicates that the tested `IUT` contains a security vulnerability with respect to the tested objective.

In our approach the TP is used to drive the test generation as described in Subsection II-A. In this example, five test cases are generated one for each resource type specified in the TP. Figure 4 illustrates a detailed test description composed of several steps in a readable form used for reporting. The test steps gather the actions to be taken for executing the test case and specify the concrete resource type to be created. Each step is essentially a concrete instance of the test scenario already introduced. More precisely, each step defines a *when-then* sequence specifying the operation to be executed (e.g., CREATE request) and the values of the attributes. In Step 1, for example, `To` is an attribute indicating where the request is going or which component is targeted, which in this case is set to `CSEBase` (actually the address of the CSEBase) and `Resource-Type` indicates the nature of the resource where 2 means `AE`.

Steps	Actions

Step 1 *(AE)*	*SendCreateRequest*
	when {
	The IUT receives a CREATE request from AE containing
	To set to CSEBase and
	Resource-Type set to 2.
	}
	then {
	The IUT should send a Response message containing
	Response Status Code set to CREATED.
	}

1.1	*Check that the message received corresponds to the expected one.*

Step 2 *(AE)*	*SendACPUpdateRequest*
	when {
	The IUT receives a UPDATE request from AE containing
	To set to ACP and
	Resource-Type set to 1.
	}
	then {
	The IUT should send a Response message containing
	Response Status Code set to UPDATED.
	}

2.1	*Check that the message received corresponds to the expected one.*

Step 3 *(AE)*	*SendCreateRequest*
	when {
	The IUT receives a CREATE request from AE containing
	To set to AE and
	Resource-Type set to 3.
	}
	then {
	The IUT should send a Response message containing
	Response Status Code set to ACCESS_DENIED.
	}

3.1	*Check that the message received corresponds to the expected one.*

Figure 4. Test steps

When a device fails the tests (i.e., does not conform with a certain level of security), the next step in our approach is the revision of the deployed security policies or the specification and deployment of new policies in order to correct the vulnerable behavior. Figure 5 shows the security policy specified using a SecKit mechanism template to make sure the `IUT` conforms to the TP defined in Figure 3. This policy is triggered when the `SendCreateRequest` tentative activity or operation is observed with parameters `To1` and `Resource-Type1`, and eventually in the past the same activity has already happened with parameters `To2` and `Resource-Type2`. The enforcement action specified is to deny the tentative create request in the event part of the mechanism, which correspond to the `ACCESS_DENIED` response code. The mechanism configuration shown in the Figure 5 specifies the parameters to ensure the conformance with the specific concrete test steps specified in 4, where the mechanism template should be instantiated with the parameters `ACP`, `1`, `CSEBase`, and `2`.

IV. Conclusion and Future Work

In this paper we propose an integrated model-based approach for improving the certification label of IoT system consisting of devices and platforms using a policy-based

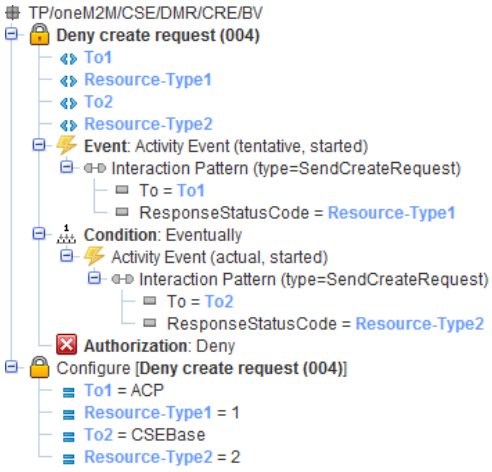

Figure 5. Security policy

approach. Our approach is able to generate tests according to a specific test purpose definition and, in case the device or platform does not conform, security policies can be configured to correct the vulnerability. Therefore, in addition to simply implementing the security policies or approach is able to certify that the combination of the system and the defined security policies conforms to the security requirements specified in the test purpose

We show the details about our approach and also how it was applied to a case study based on the oneM2M standard. Our case study demonstrates that our approach is easily applicable to oneM2M platforms, using relatively little effort. Part of our case study was demonstrated during the second oneM2M interoperability event[2] (May 2016 at Seoul, South Korea). This event gathered more than 20 oneM2M platform providers where we successfully tested their conformance to the standard. As future work we plan to work further in the evaluation and validation of our integrated approach on a larger scale testbed with a higher diversity of IoT devices and platforms considering other standards and proprietary approaches.

Acknowledgement

This work was done in the context of the ARMOUR project Grant id 688237 of the Horizon 2020 Call ICT-12-2015 Integrating experiments and facilities in FIRE+.

References

[1] I. Labs, "Internet of Things (IoT) Security Testing Framework," https://www.icsalabs.com/sites/default/files/body_images/ICSALABS_IoT_reqts_framework_v2.0_161026.pdf, online; accessed 07 February 2017.

[2] R. Anderson and S. Fuloria, "Certification and evaluation: A security economics perspective," in *2009 IEEE Conf. on Emerging Technologies Factory Automation*, Sep 2009.

[3] KrebsOnSecurity, "Who makes the iot things under attack?" https://krebsonsecurity.com/2016/10/who-makes-the-iot-things-under-attack/, online; accessed 10 January 2017.

[4] Z. Yan, P. Zhang, and A. V. Vasilakos, "A survey on trust management for internet of things," *Journal of Network and Computer Applications*, vol. 42, pp. 120 – 134, 2014. [Online]. Available: //www.sciencedirect.com/science/article/pii/S1084804514000575

[5] G. Baldini, A. Skarmeta, R. Neisse, B. Legeard, E. Fourneret, and F. L. Gall, "Security certification and labelling in internet of things," *IEEE World Forum on IoT (WF-IoT)*, 2016.

[6] R. Neisse, G. Steri, I. N. Fovino, and G. Baldini, "Seckit: A model-based security toolkit for the internet of things," *Computers & Security*, vol. 54, pp. 60 – 76, 2015, secure Information Reuse and Integration & Availability, Reliability and Security 2014.

[7] R. Neisse, G. Steri, and G. Baldini, "Enforcement of security policy rules for the internet of things," *2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, vol. 00, no. undefined, pp. 165–172, 2014.

[8] G. Bernabeu, E. Jaffuel, B. Legeard, and F. Peureux, "MBT for global platform compliance testing: Experience report and lessons learned," in *25th IEEE International Symposium on Software Reliability Engineering Workshops, ISSRE Workshops, Naples, Italy, November 3-6, 2014*, 2014, pp. 66–70.

[9] J. Botella, F. Bouquet, J. Capuron, F. Lebeau, B. Legeard, and F. Schadle, "Model-based testing of cryptographic components - lessons learned from experience," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, Luxembourg, Luxembourg, March 18-22, 2013*, 2013, pp. 192–201.

[10] C. Willcock, T. Dei, S. Tobies, S. Keil, F. Engler, and S. Schulz, *An Introduction to TTCN-3*, 2nd ed. Wiley Publishing, 2011.

[11] J. Grabowski, D. Hogrefe, G. Réthy, I. Schieferdecker, A. Wiles, and C. Willcock, "An introduction to the testing and test control notation (ttcn-3)," *Computer Networks*, vol. 42, no. 3, pp. 375–403, 2003.

[12] F. Bouquet, C. Grandpierre, B. Legeard, F. Peureux, N. Vacelet, and M. Utting, "A subset of precise UML for model-based testing," in *3rd int. Workshop on Advances in Model Based Testing*, 2007, pp. 95–104.

[13] oneM2M Partners Type 1, "Security solutions - ts-0003-v2.4.1," Technical Specification, 2016. [Online]. Available: http://www.onem2m.org/images/files/deliverables/Release2/TS-0003_Security_Solutions-v2_4_1.pdf

[14] ——, "Functional architecture - ts-0001-v2.10.0," Technical Specification, 2016. [Online]. Available: http://www.onem2m.org/images/files/deliverables/Release2/TS-0001-%20Functional_Architecture-V2_10_0.pdf

[2] http://www.onem2m.org/news-events/news/104-onem2m-hosts-second-interop-event