Advanced IoT cybersecurity for human beings

Ane Sanz[(1)], Jasone Astorga[(1)], Maider Huarte[(1)], Eduardo Jacob[(1)], Mikel Uriarte[(2)]
asanz060@ikasle.ehu.eus, jasone.astorga@ehu.eus, maider.huarte@ehu.eus, eduardo.jacob@ehu.eus, muriarte@nextel.es
[(1)]Department of Communications Engineering. University of the Basque Country UPV/EHU, 48013 Bilbao, Spain
[(2)]Nextel S. A., 48170 Zamudio, Spain

***Abstract*—Security, and access control in particular, is one of the main drawbacks to the broad adoption of IoT solutions, as traditional solutions cannot be directly used in IoT environments due to the significant resource constraints of the targeted devices. Among security solutions specifically tailored to IoT, the Hidra access control mechanism stands out as an efficient approach to bring to the IoT world the flexibility and expressiveness achieved in traditional computing environments. This is fulfilled by the definition of complex security policies and their codification using a specific policy language. Unfortunately, the resulting system becomes complex and difficult to use by non-security experts, hindering its broad adoption. Therefore, this paper presents some enhancements to make powerful security mechanisms usable by human beings whatever their expertise: a user-friendly mechanism to create and manage security policies, and an encoder/decoder module to convert security policies from human-friendly to machine-friendly format and vice versa.**

I. Introduction

The Internet of Things (IoT) is a system of interconnected devices which have the ability to receive, collect and send data in an ubiquitious way. For this aim a large number of Constrained Device Sensors (CDSs) are deployed.

Although traditional IoT scenarios comprise sensors that gather data and transmit it to a centralized well-known server, the next generation of IoT envisions more powerful scenarios, involving smart CDSs which behave as small servers. In such scenarios, the CDSs have the ability to receive and process client queries directly with a secure end-to-end (E2E) communication establishment. Therefore, in this kind of applications it is necessary to guarantee security properties such as confidentiality, availability, integrity, authenticity and authorization of the access requests received by the CDSs.

However, in the last years there has been a large increase in cybercrime, generating in 2018 a loss of USD 1.5 trillion according to S21sec [1]. Although companies have enforced and developed their security mechanisms to prevent from attacks, criminals have also consolidated and improved their attack techniques, and this increases the vulnerability of the systems. In this context, IoT remains as a specially vulnerable scenario, since traditional security mechanisms designed for non constrained devices are not directly applicable to CDSs where processing capacity and memory are highly limited.

In order to deal with these attacks and prevent them, different alternatives have been implemented, mainly based on encrypting communications at the link layer. However, access control remains insufficiently resolved, with static approaches that do not take into consideration the dynamic and flexible nature of the targeted applications, and alternatives that lack expressiveness.

As it will be explained in more detail below, Hidra is an access control mechanism that implements very powerful mechanisms in order to enable the expressiveness of policies. However, this may go against usability when the policies to be defined increase their complexity. Therefore, to ensure usability, some new mechanisms are needed, namely, a user-friendly software to create and represent policies, and an encoder/decoder to convert the policies from human-friendly to machine-friendly format. These mechanisms will be explained and presented in this paper.

The rest of the paper is structured as follows. Section II analyses related works, while Section III presents building technologies such as Hidra and the conveyed policy language. The proposed graphic interface and encoder/decoder are described in Section IV. Section V presents the validation scenario and Section VI summarises the main conclusions of the paper.

II. State of the art

Currently, one of the main issues that is hindering the broad adoption of IoT architectures is the lack of suitable security solutions, specially in the field of strong, expressive and usable access control solutions. Traditional security mechanisms are not directly applicable to IoT environments due to the heavy limitations of the devices involved. Although there have been efforts to adapt traditional policy-driven solutions to IoT, such as the Authorization Framework for the IoT [2] and the Usage based access control (Ucon) adapted to IoT [3], these solutions rely on a centralized approach and do not consider local context conditions in the CDSs.

Among the architectures based on a distributed approach, CapBAC for IoT [4] defines a capability-based access control solution based on XML and developed in Java. Therefore, it is not suitable for very constrained devices. An alternative approach is DCapBAC [5], which proposes a capability-based fully distributed architecture. However, the flexibility and dynamism of this solution is limited since permissions are included in the capabilities, which are obtained before the first access attempt and cannot be refreshed.

Another solution is the delegated CoAP authentication and authorization framework (DCAF) [6], which is based on pre-shared keys and the establishment of a DTLS channel. In this case, authorization policies are defined as local conditions and CBOR serialization is used to compress the policies. However, this solution implies the overhead of a DTLS channel establishment and CBOR [7] serialization, which being a general purpose mechanism, does not achieved the needed compression levels. Similarly, OSCAR [8] proposes an object security architecture for IoT which allows E2E access control based on PKC schemes. However, OSCAR suffers also from the overhead of the DTLS channel establishment.

In order to tackle the resource scarcity in the targeted devices, Ladon [9] protocol was proposed as a very lightweight

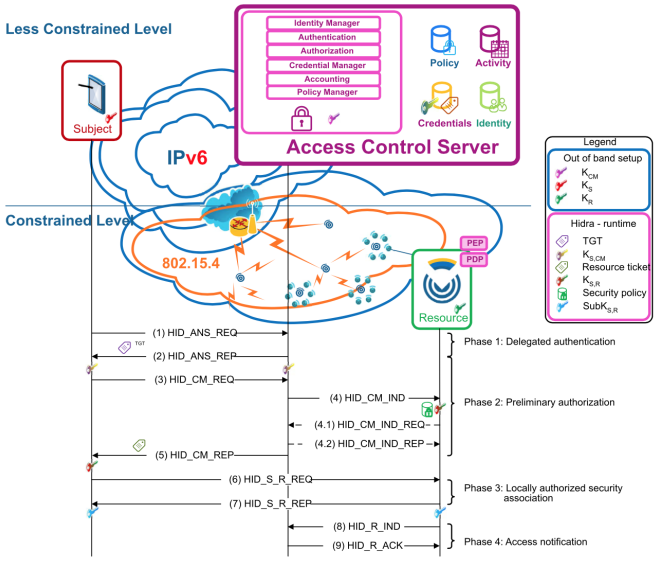

Fig. 1. Hidra protocol

access control solution for CDSs. However, the security policies used by Ladon are rather static and therefore, the Hidra protocol was designed as a mechanism to define very expressive policies, which implies a rather complex policy language. For this reason, and in order to facilitate the adoption of powerful and expressive access control mechanisms for IoT, some mechanisms are needed to hide the complexity of the security mechanisms from final users and make security for IoT usable for non-expert human beings.

III. Building Technologies: The Hidra Access Control Mechanism

In this paper, some mechanisms to enhance the usability of access control models for IoT are presented. Although these mechanisms can be extended to a wider scope, currently they have been specifically designed and implemented to fit the Hidra access control model. For this reason, in this section Hidra is briefly summarized. Readers wanting a more detailed explanation about the operation of the Hidra access control model, please refer to [10].

Hidra is a security protocol that guarantees both the authentication and authorization of a remote subject wanting to access a service in a CDS. Besides, in order to provide expressiveness to the access control procedure, it combines a very expressive policy language with a distributed architecture which makes the authorization in two steps.

As shown in Figure 1, the centralized Access Control Server (ACS) performs authentication and preliminary authorization (phases 1 and 2) of remote requesting subjects. This preliminary authorization allows discarding most unauthorized service requests before they even reach the CDS. However, in order to enable a rich and context-based access control decision, a second authorization phase is performed locally in the CDSs (phase 3).The access control enforcement in the CDSs is mandatory to enable access control policies aware of local or context parameters such as the battery level in the CDS, environmental temperature, etc. The access control policy to be applied by the CDS is created by the

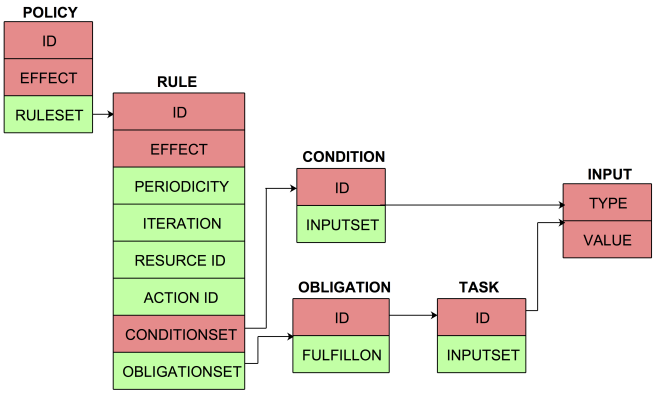

Fig. 2. Policy language

centralized authorization server and conveyed to the CDS by means of a specific message (message 4, HID_CM_IND), enabling this way dynamism and avoiding the necessity of storing large access control policies permanently in the CDS. Finally, in phase 4 (Access Notification), the CDS sends to the central architecture the details about the access attempts, both successful or not, enabling centralized logging and accounting suitable to detect anomalies, for example. The results presented in [10] show that this access control mechanism is adequated for the targeted sensors in terms of processing time, storage needs and energy consumption.

As explained in the previous paragraph, the CDS performs a local-context based authorization according to an access control policy previously received. In order to guarantee the proper operation and the expressiveness of this process, it is necessary to define a policy language, which is going to be described in the next subsection.

A. Policy language

The defined policy language consists of different constructs, which can be mandatory or optional, and that go nested in other constructs so that the whole policy has the appropriate meaning. In order to understand the policy correctly, it is very important to maintain the order of the constructs. The constructs that have been defined are *policy*, *ruleset*, *conditionset*, *obligationset*, *task* and *inputset*. In Figure 2, the relation between the different constructs is shown, which will be described in more detail below.

The first construct of the language is the policy construct, which is mandatory and contains an *id* and a *granting effect*. Then, it is optional to add to this construct some *rule* constructs. Each rule construct will have the following mandatory information: *id*, *effect* and at least one *condition* construct. Moreover, it can also have more optional information to detail the application of the rule: *periodicity*, *iteration*, *resource*, *action*, and some *obligation* constructs.

The condition construct defines the conditions that should be evaluated by the CDS, and it has a mandatory *expression* or function identified with an *id* as well as an optional *inputset* construct for the attributes.

The obligation construct, which is optional and extendible inside the rule, has a ”fulfill on” variable and a *task* construct nested.

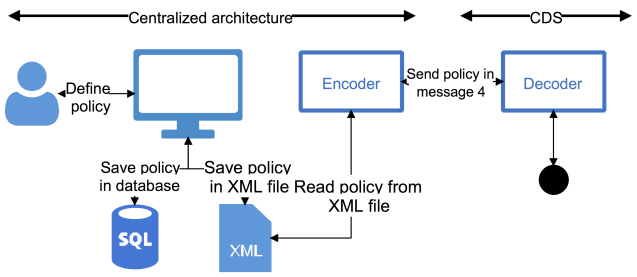

Fig. 3. Block diagram of the usability enhancements for access control

The task construct, which is mandatory in the obligationset, enables the definition of a function to be executed as an obligation. It has a task function identified with an id and can have an optional inputset construct for the attributes.

The last construct is the inputset construct, which can be used in both conditionset and task constructs. Each input needs to have two variables to define the type and the value of the attribute.

In order to implement the Hidra protocol with all the functionalities it offers and make it usable in real industrial environments, some parts are still missing: the possibility to add and remove policies easily, an encoder module to encode the policy and send it from the ACS to the CDS, and a decoder to decode the policy in the CDS and make a decision depending on the information it contains.

IV. Usability enhancements for access control mechanisms

In this section, the mechanisms proposed to enhance the usability of the access control system are explained. As shown in Figure 3, the first step is a graphical interface that enables the creation and saving of new policies to authorized users. When the policy is saved, it is read from the CM (Credential Manager)and it has to be encoded according to the codification that will be described below. After encoding the policy, it is included in message 4 of Hidra and sent to the CDS. When the CDS receives the message, it processes it and in order to make a decision, it has to decode back the policy.

In the next subsections the enhancements made to Hidra to improve its usability will be described, namely the graphical user interface to easily define security policies, the encoder and the decoder.

A. Graphical User Interface

As mentioned in the previous section, in order to make the Hidra protocol usable, it is important to have a graphical interface so that network administrators with proper authorization can create and add new policies to the database to send them to the CDS when necessary.

First of all, it is important to have an access control for the service of this interface, as it can be used only by people with proper authorization. Furthermore, as described in the previous sections, the graphical interface consists of creating new policies by adding data to them. The structure of the policy can be different in terms of length and contents, because most of the parts are optional and are nested in other structures. Therefore, a dynamic web interface has been programmed, where the user has the option to decide whether she wants to add some structs or information to the policy or not. During the creation process, it is possible for the user to visualize the contents that have been already added to the new policy. This way, she will be able to see all the nested structures in different tables and understand the final policy she is creating.

In Figure 4, an example of the policy creation is shown. In this example, the user has created a policy with 2 rules, and each of them has different information. Red columns are mandatory in the corresponding struct, while green columns represent optional information. Finally, empty columns mean that the user has decided not to include any information in them.

When the user has entered all the information in order to create the policy, this has to be saved correctly. On the one hand, the policy is saved in an XML file, which will be used to read the information and encode it. On the other hand, all the policy's information is saved in a database in order to keep the information of all the policies that have been created. Apart from saving the different data structures the policy contains, it is also important to save the information about who created which policy and when. For this aim, a MySQL database has been chosen.

B. Encoder and decoder

As mentioned in the previous sections, the CDS performs a local context-based authorization based on a policy received from the ACS. This policy, which has been created using the graphical interface and saved in both a database and an XML file, needs to be codified following the policy language structure explained before.

Due to the constraints of the sensors in terms of capacity and energy consumption, it is important to have a very compressed representation of the policy in order to reduce the impact on storage, transmission and processing. There are different ways of representing the information of a policy, such as JSON [11] and CBOR. However, being these two alternatives generic, do not offer a good level of compression. For this reason, a different policy codification is proposed in Hidra: Authorization Policy Binary Representation (APBR). This codification offers the best compression factor in comparison with the other alternatives. For example, in a policy with one rule containing both a condition and an obligation, JSON representation needs 236 bytes, CBOR 123 bytes and APBR just 9 bytes. Therefore, the compression and optimization achieved with APBR meets the needs of the targeted devices.

The APBR codification creates a binary representation of the policy information following the order of the different structs of the policy language. Each field is codified to its binary representation, with different amount of bits in each case. Furthermore, as there are some structs that are optional, some flags are introduced to the bitstream in order to depict that the optional part that follows exists. If, on the contrary, the flag is not activated, it means that there is not optional construct in that part.

Therefore, an encoder module that follows the APBR codification has been created. This module reads the policy from the XML file, and creates a binary representation,

Policy

Effect
PERMIT

Created rules

Effect	Periodicity	Iteration	Resource	Action	Obligations	Conditions
PERMIT	4				FulfillOn: DENY; Task: Function: activate; Inputs: Type: SYSTEM_REFERENCE, Value: onMaintenance	Function: lowBattery; Inputs:
DENY				DELETE	FulfillOn: PERMIT; Task: Function: ++; Inputs:	Function: /; Inputs: Type: STRING, Value: admin Function: contains; Inputs: Type: BOOLEAN, Value: true Type: REQUEST_REFERENCE, Value: roles

Fig. 4. Example of a policy in the graphical interface

which will be sent to the CDS in the message 4 of Hidra. When creating the APBR representation of the policy, it is crucial to follow the construct order described in the policy language. This binary representation will be received and decodified by the CDS later on, and in order to interpret the information correctly, it is necessary for both the codification and decodification processes to follow the same order.

After the policy has been codified and sent to the CDS, and in order to make an authorization decision, the CDS has to decode back the bitstream and interpret the information. For this purpose, a decoder module has also been created.

V. Validation scenario

The system described above has been validated in an Industry 4.0 scenario, specifically, in a centre that focuses on research. This centre is the Aeronautics Advanced Manufacturing Centre (AAMC) [12], created by the University of the Basque Country with the financial support of companies of the aeronautics sector and the local and autonomous government to develop advanced manufacturing technologies and promote transference to the industrial sector. In order to become a research reference at international level, the AAMC has a manufacturing plant with cutting-edge equipment, where several sensors are deployed in order to gather information about the manufacturing processes taking place and the operation of the machine tools.

As different companies, even competing ones, share resources and machine tools in the AAMC, guaranteeing access control to the information gathered by sensors is crucial. Moreover, most of the people working there are industrial engineers with very basic knowledge on networking and security. Therefore, providing a human friendly user interface to the definition of the access control policies is essential in order to promote the usage of such IoT security systems.

VI. Conclusions

IoT applications are expected to be expanded and converted into a very important part of the society. Furthermore, in order to make the Industry 4.0 grow and work as expected, proper IoT security mechanisms are essential, due to the high level of sensitivity of the data used by these applications. However, not only powerful and expressive security mechanisms are needed, but also some user friendly interfaces in order to make them usable by non-expert people.

Therefore, the main contribution of this paper is an intuitive user interface to define and save different security policies that are locally executed by the CDS. Moreover, as extra modules, an encoder and a decoder, have been created in order to compress the policy information as much as possible when it is transmitted to the CDS, thus saving CDS's resources. As a validation scenario the AAMC, a research focused center, has been used.

Acknowledgements

This work was supported by the Department of Economic Development and Competitiveness of the Basque Government through the CogNoms4.0 KK-2018/00049 research project.

References

[1] S21sec, "Cyber predictions for 2019." https://www.s21sec.com/en/cyber-predictions-2019/, 2019.
[2] L. Seitz, G. Selander, and C. Gehrmann, "Authorization framework for the internet-of-things," *WoWMoM IEEE*, pp. 1–6, 06 2013.
[3] G. Zhang and W. Gong, "The research of access control based on ucon in the internet of things," *JSW*, vol. 6, pp. 724–731, 04 2011.
[4] S. Gusmeroli, S. Piccione, and D. Rotondi, "Iot access control issues: a capability based approach," 07 2012.
[5] J. Hernández-Ramos, A. J. Jara, L. Marín, and A. Skarmeta, "Dcapbac: Embedding authorization logic into smart things through ecc optimizations," *International Journal of Computer Mathematics*, vol. 93, 05 2014.
[6] V. Beltran and A. Skarmeta, "An overview on delegated authorization for coap: Authentication and authorization for constrained environments (ace)," pp. 706–710, 12 2016.
[7] C. Bormann and P. Hoffman, "Concise binary object representation (cbor)," RFC 7049, RFC Editor, October 2013.
[8] M. Vučinić, B. Tourancheau, F. Rousseau, A. Duda, L. Damon, and R. Guizzetti, "Oscar: Object security architecture for the internet of things," pp. 1–10, June 2014.
[9] J. Astorga, E. Jacob, M. Huarte, and M. Higuero, "Ladon: End-to-end authorisation support for resource-deprived environments," *IET Information Security*, vol. 6, pp. 93–101, 06 2012.
[10] M. Uriarte, J. Astorga, E. Jacob, M. Huarte, and M. Carnerero, "Expressive policy-based access control for resource-constrained devices," *IEEE Access*, vol. 6, pp. 15–46, 2018.
[11] T. Bray, "The javascript object notation (json) data interchange format," STD 90, RFC Editor, December 2017.
[12] "Aeronautics advanced manufacturing centre." https://www.ehu.eus/en/web/CFAA/home.