A Model to Enable Application-scoped Access Control as a Service for IoT Using OAuth 2.0

Federico Fernández, Álvaro Alonso, Lourdes Marco, Joaquín Salvachúa

Abstract—**Access Control is crucial for security management, but in the context of the Internet of Things it cannot be implemented the same way as traditional systems do. Indeed, devices that make the Internet of Things impose some constraints that encourage the design of new access control mechanisms, which should provide flexibility of configuration, as well as support several authorization scopes at the same time, yet being computationally light, dynamic and scalable in order to be ready for the forthcoming Cloud Computing paradigm. In this paper we propose an authorization model that is based on the OAuth 2.0 protocol. From the point of view of the identity provider, this model allows managing roles and permissions for an application-scoped authorization, to enable more flexible scenarios in which multiple tenants take part. With regard to devices, the OAuth 2.0 makes authorization extremely light, because all the required information is provided with a token. Considering all this, authorization management is completely delegated to an external system, so that an as-a-service access control mechanism is provided. The proposed model complies with the security, flexibility and performance requirements that are needed in the Internet of Things paradigm.**

I. Introduction

Security management is an essential issue in all digital contexts, but in the Internet of Things (IoT) it poses even a more critical challenge. Sensors, actuators and, in general, devices that make the IoT serve somehow as connecting points between the real, physical world and the digital, ever-connected Internet; thus, the IoT will not settle unless management and control over the access of those devices to the rest of the Internet is ensured right from the design phase.

In most IoT use cases, such as in applications of Smart Cities, the devices need to access resources in a service, either to publish information (sensors) or to read it and trigger some reaction (actuators). Security of these interactions should be guaranteed, but not at all costs. Instead, the system providing access control over those resources should also take into account the constraints that IoT devices impose, which add to the access control requirements of their traditional, non-IoT counterparts. Among them are their low computing capabilities and their remote, usually difficult-to-access location.

Multi-tenancy should also be featured, so that access control does not only depend on the type of device. This way, authorization to the given sensor or actuator would be granted or not depending on the application it is willing to access. Besides, flexibility requirements should also be taken into consideration, because different applications or even resources within an application may have different security requirements.

The work that is related to this matter is analyzed in the following Section. We then in Section III describe in detail our solution. Finally, some conclusions and future work are discussed in Section IV.

II. Background

Traditionally, security administration of large systems has been simplified by a Role-based Access Control approach (RBAC), which scales better than other previous models like Identity Based Access Control (IBAC) [1]. By treating roles and identity as characteristics of a principal, Attribute Based Access Control (ABAC) fully encompasses the functionality of both IBAC and RBAC, and can define permissions based on just about any security relevant characteristics [2]. A more flexible policy enforcement can be achieved constructing Flexible Data Access Control services [3].

With regard to IoT some approaches have used ABAC models, combined in some cases with the second version of the OAuth protocol (OAuth2) or the Constrained Application Protocol (CoAP), to protect the access to data [4] [5].

OAuth2 is an open authorization protocol with a simple and secure way to authenticate users and allow access to their protected data [6], giving the service with an access token. Authors of [7] propose an approach targeting HTTP/CoAP services to provide an authorization framework, which can be integrated by invoking an external OAuth-based Authorization Service (OAS). However, in the scenario we describe the IoT device acts as *Resource Owner* (using the OAuth2 terminology), therefore the use case is slightly different.

III. As-a-service Access Control with OAuth 2.0

In this Section we first identify the requirements that the access control system must meet. Second, a description of the proposed architecture is provided, followed by a sample interaction flow of a typical use case.

A. Requirements to be fulfilled

According to the problem described in Section I, the designed access control architecture should feature the following four characteristics:

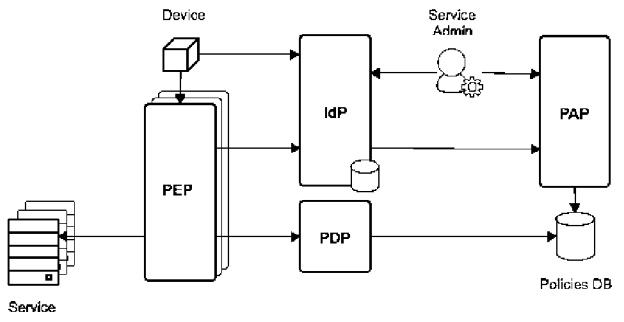

Fig. 1. As-a-service Access Control Architecture

1) Application-scoped: The system should make it possible for different authorization policies to be enforced depending on the service that is to be accessed by the given IoT device.

2) Client-independent: Access control should be provided the same way regardless the client that tries to access resources in the service. It should work both for highly constrained devices (like IoT devices) and heavy clients, such as web browsers or other back-end services. This way, interoperability between IoT and the rest of the Internet is guaranteed with a single access control architecture.

3) Flexible: The service administrator should be able to define authorization policies to their service in a fine-grained way. Using a policy description language like XACML [8] is recommended. Besides, service administrators should be allowed to manage roles of both single devices and groups of devices, so that they can create a multi-tenant authorization scheme.

4) Delegated: Detaching authorization operations from the rest of entities would provide computational lightness for clients (which is critical for IoT devices), as well as scalability and ease of remote configuration for the service administrator, because the whole authorization functionality is centrally provided.

B. Proposed Architecture

Figure 1 shows the architecture scheme of our model. We consider the following entities: an IoT device (we will just write *Device*), an application (i.e. a *Service* whose resources are willing to be accessed by the *Device*) and one of its administrators (the *Service Admin*), as well as an Identity Provider (*IdP*) and the Policy Administration, Decision and Enforcement Points (*PAP*, *PDP* and *PEP*). The last three are the components that make the widely known access control architecture [9]. Besides, we have included a *Policies DB* where policies are stored by the *PAP* and checked by the *PDP*.

The *IdP* is a key component of the architecture. It is in charge of managing the credentials (usually a username and a password) of both the *Service Admin*, the *Device* and optionally any other user of the *Service*. Groups of devices and users may also be created in this component, so that complex authorization scenarios can be configured.

Access control policies are defined by means of permissions and roles, which the *Service Admin* manages in the *PAP*. Permissions are usually composed by an action (e.g. an HTTP verb) and a resource of the *Service*. However, more complex policies may also be defined by means of a policy-description language like XACML. Roles are in turn sets of permissions, which may be assigned to the *Device* or to a group of devices in the *IdP*. Granting a role to a *group* of devices, rather than to a single device, has the benefit of linking a given role to the belonging to a certain group. For instance: devices from a neighborhood A in a Smart City application must be the only ones able to write data in the *Service* about neighborhood A; therefore, when one of those devices moves to a neighborhood B, removing it from the A-group is enough to deny its access to resources of their past neighborhood. Besides, roles are assigned to devices in the scope of the *Service*, so a given *Device* may have different roles depending on the application.

It is important to highlight that, although policies and their relationships with roles are defined in the *PAP*, the mapping between users, devices and roles is made in the *IdP*. The *Service* must also be registered here. As a result of the registration, the *Service Admin* obtains the associated OAuth2 credentials, which must be stored in the *Device*.

When performing a request to the *Service*, the *Device* will make use of the Implicit OAuth2 Grant [10] to obtain an access token. During an authorization check, the *PEP* is the entity that uses this token to retrieve the roles assigned to the *Device*. Thereby, a mapping can be made between OAuth2 consumers and applications. The *PDP* fetchs the policies associated to those roles from the *Policies DB*, and decide whether or not access should be granted based on them.

Note that one *PEP* is set for each *Service*, thus a transparent, as-a-service authorization is provided to the latter. Besides, the delegation of the authorization operations meets the computational restrictions that the *Device* imposes, because it should only have to implement an OAuth2 client. In fact, any entity implementing an OAuth2 client library is eligible for working as a client in this architecture, thus interoperability with other non-IoT systems is guaranteed.

C. Sample Authorization Flow

We now present a typical IoT use case to better illustrate the interactions between the modules we have presented in our model. Let us consider a scenario in which a *Device* publishes the data it collects in a back-end *Service*. Our objective is to add an access control mechanism that allows restricting access to that application depending on 1) the type of IoT device, 2) the resource that it is trying to access and 3) the action that it is trying to perform. A *Service Admin* is assumed to be registered in the *IdP*, with the needed rights to register new applications and manage policies in the *PAP*.

The fist step consists on the registration and configuration of the *Service* and the *Device*. To do this, the *Service Admin* has to perform the following operations:

1) Register the *Service* in the *IdP*, to obtain both the OAuth2 credentials for its application (i.e. the *client_id* and *client_secret*) and those associated to its corresponding *PEP* (i.e. the *pep_username* and *pep_password*).

2) Register the *Device* in the scope of the *Service*, to obtain the *Device* credentials (i.e. the *device_username* and *device_password*).
3) Create the desired roles and policies in the *PAP* and define the relationships between them. The *PAP* will then store those entities and their relationships in the *Policies DB*.
4) Grant the desired roles to the *Device* (in the scope of the *Service*) using the *IdP*. To display the available roles, the *IdP* must send a request to the *PAP*. As it has been explained above, roles can be assigned to devices or to groups of devices, so the *Service Admin* may optionally manage groups of devices and assign roles to them (again in the scope of the *Service*).

Once the environment is configured, the process to send a publish request from the *Device* to the *Service* consists on the following interactions:

1) The *Device* sends a request to the *IdP* in order to create an OAuth2 token. The *Device* and the application credentials that were obtained during the registration are needed here.
2) The *Device* sends the publish request to the *Service*, which includes in the authentication header the OAuth2 token created in the previous step.
3) The *PEP* module intercepts the request and extracts the token from the header. Then, it sends a validation request to the *IdP* to check whether the token corresponds to a registered device in the platform. The *PEP* is allowed to perform this validation in the *IdP* because it authenticated previously, using the *PEP* credentials obtained by the *Service Admin* at registration time. If the validation is rejected, the *PEP* sends an *Unauthorized* response to the *Device*. In case the validation succeeds, the *IdP* adds the public information of the *Device* to the response, which includes the roles that it has in the scope of the application in which the token was created.
4) The *PEP* then sends an authorization request to the *PDP*, which includes the retrieved roles, the action that the *Device* is trying to perform and the resource that it is trying to access. Based on the policies that are stored in the *Policies DB*, the *PDP* takes the decision of either allowing or denying the access to the resource, and returns the verdict to the *PEP*.
5) If the *PDP* has denied the access, the *PEP* will finally send an *Unauthorized* response to the *Device*. Otherwise, it will forward the request of the *Device* to the *Service*, including the public information which was previously obtained from the *IdP*. This information can be useful for the *Service* when performing the needed actions.

IV. Conclusions

We have presented an architectural model that enables access control in IoT contexts. The use of the OAuth2 protocol makes this model interoperable with other RESTful services in the rest of the Internet, while also making the whole authentication mechanism extremely light for devices, since an OAuth2 token provides all the required information.

Our approach reduces effort for IoT application developers to add an as-a-service authorization layer to their applications, because they only need to register their service in the *IdP* and deploy a *PEP* component, which is a generic component that can be reused for all services and usually may also be provided as part of the access control service. Besides, the fact that the access control service is completely external to services facilitates the configuration of authorization policies, since they are all managed from a central, single component: the *PAP*.

As a future research line, this model should be implemented and validated. In cases when storing the OAuth2 credentials in the IoT device constitues an issue (e.g. access to the device could be compromised), an approach like using a different OAuth2 grant could be considered. Moreover, the whole process be made even lighter by supporting the CoAP protocol in the interactions in which the IoT devices take part.

Acknowledgment

The authors would like to thank the FIWARE project and its partners. Part of the ideas we propose in this paper have been developed in the scope of this European Project.

References

[1] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, “Role-based access control models,” *Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[2] E. Yuan and J. Tong, “Attributed based access control (abac) for web services,” in *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*. IEEE, 2005.

[3] Y. Zhu, D. Huang, C.-J. Hu, and X. Wang, “From rbac to abac: constructing flexible data access control for cloud storage services,” *IEEE Transactions on Services Computing*, vol. 8, no. 4, pp. 601–616, 2015.

[4] M. Hemdi and R. Deters, “Using rest based protocol to enable abac within iot systems,” in *Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2016 IEEE 7th Annual*. IEEE, 2016, pp. 1–7.

[5] S. Kinikar and S. Terdal, “Implementation of open authentication protocol for iot based application,” in *2016 International Conference on Inventive Computation Technologies (ICICT)*, vol. 1, Aug 2016, pp. 1–4.

[6] S. Emerson, Y.-K. Choi, D.-Y. Hwang, K.-S. Kim, and K.-H. Kim, “An oauth based authentication mechanism for iot networks,” in *Information and Communication Technology Convergence (ICTC), 2015 International Conference on*. IEEE, 2015, pp. 1072–1074.

[7] S. Cirani, M. Picone, P. Gonizzi, L. Veltri, and G. Ferrari, “Iot-oas: An oauth-based authorization service architecture for secure services in iot scenarios,” *IEEE sensors journal*, vol. 15, no. 2, pp. 1224–1234, 2015.

[8] “eXtensible Access Control Markup Language (XACML) Version 3.0.” January 2013. [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html (accessed: February, 2017)

[9] F. Turkmen and B. Crispo, “Performance evaluation of xacml pdp implementations,” in *Proceedings of the 2008 ACM workshop on Secure web services*. ACM, 2008, pp. 37–44.

[10] E. D. Hardt, “The OAuth 2.0 Authorization Framework,” Internet Requests for Comments, RFC Editor, RFC 6749, October 2012. [Online]. Available: http://tools.ietf.org/html/rfc6749 (accessed: February, 2017)