future internet

MDPI

Article

IAACaaS: IoT Application-Scoped Access Control as a Service †

Álvaro Alonso *, Federico Fernández, Lourdes Marco and Joaquín Salvachúa

Departamento de Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid, 28040 Madrid, Spain; fefernandez@dit.upm.es (F.F.); lmarco@dit.upm.es (L.M.); jsalvachua@dit.upm.es (J.S.)

* Correspondence: aalonsog@dit.upm.es

† This paper is an extended version of our paper published in Fernández, F.; Alonso, A.; Marco, L.; Salvachúa, J. A model to enable application-scoped access control as a service for IoT using OAuth 2.0. 2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN), 2017; pp. 322–324.

Received: 1 September 2017; Accepted: 13 October 2017; Published: 17 October 2017

Abstract: access control is a key element when guaranteeing the security of online services. However, devices that make the Internet of Things have some special requirements that foster new approaches to access control mechanisms. Their low computing capabilities impose limitations that make traditional paradigms not directly applicable to sensors and actuators. In this paper, we propose a dynamic, scalable, IoT-ready model that is based on the OAuth 2.0 protocol and that allows the complete delegation of authorization, so that an as a service access control mechanism is provided. Multiple tenants are also supported by means of application-scoped authorization policies, whose roles and permissions are fine-grained enough to provide the desired flexibility of configuration. Besides, OAuth 2.0 ensures interoperability with the rest of the Internet, yet preserving the computing constraints of IoT devices, because its tokens provide all the necessary information to perform authorization. The proposed model has been fully implemented in an open-source solution and also deeply validated in the scope of FIWARE, a European project with thousands of users, the goal of which is to provide a framework for developing smart applications and services for the future Internet. We provide the details of the deployed infrastructure and offer the analysis of a sample smart city setup that takes advantage of the model. We conclude that the proposed solution enables a new access control as a service paradigm that satisfies the special requirements of IoT devices in terms of performance, scalability and interoperability.

Keywords: IoT; security; access control; identity management; OAuth 2.0; IAACaaS

1. Introduction

The new paradigm brought by the Internet of Things (IoT) is affecting the way the Internet works at all levels, from IP addressing to information processing, with millions of devices reading and writing great amounts of information from services every minute.

Security management is a key element in every service, especially when it comes to controlling who can access the resources in that service. In IoT services, the key issues for security and privacy can be understood by analyzing the traditional issues we can observe in other traditional services. If we also consider the special characteristics of IoT systems, we can have a good understanding of the threat model we have to address [1–3]:

- Confidentiality is about keeping data private, encrypting the shared data flows to ensure that only authorized entities can access that data. When talking about IoT, as a result of a confidentiality breach, we can expose sensitive data. In domestic applications, this could be related to medical information, keys or passwords. If we think in terms of smart cities services, data are often public, but there are also cases in which citizens' personal information is shared through the Internet.

- Authentication is about verifying that data have been sent by the actual user or device. In IoT terms, the identity theft by an undesired actor could cause the unintentional activation of actuators or access to protected data by malicious users.
- Access refers to only allowing authorized users to access resources and ensuring that those authorized users are not prevented from such access. Even having an authenticated user in a system, we probably want to discriminate which kind of users can access specific sets of information.

As we analyze in the next section, we can find several studies about how to manage authentication and access control in traditional services. However, these traditional mechanisms are not directly applicable to IoT systems, because these pervasive systems are composed of devices with high constraints, be it their low computing capabilities or their limited network connectivity.

In most IoT use cases, the devices that take part need to access resources in a service, either to publish information (sensors) or to read it and trigger some reaction (actuators) [4]. Guaranteeing the security of these interactions is a great challenge that IoT systems are still facing. As we will analyze later, the existing approaches to Access Control (AC) in IoT do not cover all the requirements a scalable, interoperable and efficient system should have met.

In this paper, we propose a model, called IAACaaS, that allows AC for IoT systems and fulfills all the performance, scalability and interoperability requirements. We extend here the work that was presented in [5]. There, we can find an overview of the IoT environments' security requirements and a high level description of a model that enables application-scoped access control in such kinds of systems. The use of the OAuth2 protocol makes that model interoperable with other RESTful services on the Internet, making the authentication mechanism extremely light for devices. Taking this work as a starting point, we give here more details about the proposed architecture and offer a detailed view of an implementation that serves as a validation of our proposal.

Thus, in this work, we provide a solution to handle security threats related to access control. Confidentiality and authentication threats are not directly covered by our solution. However, as the model we propose is based on well-known standards and interfaces, it is totally compatible with other security solutions focused on solving those issues.

The rest of the paper is organized as follows: an analysis of the work that is related to this matter is provided in the following section; we then in Section 3 offer a deep view of the requirements we have identified to be covered by our proposal. After presenting a more conceptual approach to our solution in Section 4, its validation is presented in the scope of the European project FIWARE, that defines a platform for the Future Internet applications development and deployment (Section 5). Finally, some conclusions and future work are discussed in Section 6.

2. Related Work

When it comes to IoT, it is very common to consider security and privacy as critical topics. Roman et al. [6] analyzes and compares the requirements and challenges in IoT environments, both centralized and distributed. On the other hand, Sicari et al. [7] summarizes the existing contributions regarding security in IoT. One of the main conclusions after analyzing these works is the necessity of providing scalable and flexible security models for this kind of system, where the special constraints of the devices make traditional mechanisms not directly applicable.

If we look back, security management of large systems has been traditionally simplified by a Role-Based Access Control approach (RBAC), a policy mechanism defined around roles and privileges. This approach scales better than other previous models like Identity-Based Access Control (IBAC), based on the authenticated identity of an individual [8]. However, when talking about a huge amount of devices, managing roles for individual entities presents a big challenge. As we will analyze later, the possibility of grouping sensors and assigning roles to those that have the same rights is a good solution for this problem.

On the other hand, by treating roles and identity as characteristics of a principal, Attribute-Based Access Control (ABAC), a paradigm whereby access rights are granted to users through the use of policies that combine attributes together, fully encompasses the functionality of both IBAC and RBAC and allows the definition of permissions based on just about any relevant security characteristic [9]. An ABAC model solves the issue of managing a big number of rules by generating dynamic access policies, and the challenge it presents is the huge number of attributes that need to be managed. With respect to this, Zhu et al. [10] assert that a compact policy can reduce the size of private keys. They demonstrate how to provide access policies to facilitate more flexible access control for data access services on clouds.

With regard to IoT, some approaches have implemented a capability-based security model to enable distributed AC IoT environments [11]. A capability can be defined as a self-contained token that references a target object or information along with an associated set of access rights. However, performing the token validation in the device implies an extra computational load that is better to avoid.

Other approaches have used ABAC models, combined in some cases with the second version of the OAuth protocol (OAuth 2.0) or the Constrained Application Protocol (CoAP), to protect access to the data [12,13]. The authors of [14] propose an approach targeting HTTP/CoAP services to provide an authorization framework that can be integrated by invoking an external OAuth-based Authorization Service (OAS).

OAuth 2.0 is an open authorization protocol with a simple and secure way to authenticate users and allow access to their protected data [15]. The protocol [16] defines four roles to explain the actors and the flow to get a protected resource. The resource owner is the entity capable of granting access to the resource. For its part, the resource server is where the protected resources are hosted, and it is capable of accepting and responding to protected resource requests using access tokens (a string that represents the user in terms of authorization). The client is the application making requests, and finally, the authorization server issues access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

Thinking in terms of common IoT environments such us smart cities or homes, we can observe that there are many types of devices that provide and consume data in the deployed services and applications. Moreover, a device can use multiple services at the same time having different access grants for each of them. Thus, using an RBAC/ABAC approach to manage authorization is a good decision. On the other hand, the use of OAuth 2.0 protocol to create access tokens that represent the devices delegates the authentication to a external service and saves load in the devices. We adopt both ideas for designing the solution we propose.

However, all the approaches exposed above consider the device as the object of the operations. The other entity (a client or an external service) is the subject that sends a request to the device in order to perform an action (in the case of actuators) or to get information (in the case of sensors). If we change this paradigm by making the device the subject that checks or updates information in a publish/subscribe service, we can manage authorization and authentication externally.

In the model we propose, the IoT device acts as resource owner (following the OAuth 2.0 terminology), and it uses a token to perform authentication and authorization in an externalized system. Since these processes are delegated and the keys are not shared, what we seek and get is a lighter and safer scenario. After checking the token validity, the request is sent to the publish/subscribe service.

As outlined in [6], centralization is a common problem with AC models. Centralized AC approaches are responsible for authorization, attributes and access policies as a central entity. However, when a distributed scheme is followed, validating the correctness of the authorization tokens is the responsibility of the IoT device. As outlined in the next section, one of the critical requirements of the IoT environments is to remove from the device as many operations as possible. Thereby, delegating authorization to a centralized component is a good design decision. Furthermore, a centralized

solution is always compatible with scalable deployments that adapt the resources availability to the system demand.

3. System Requirements for Access Control in IoT

If IoT devices are to be kept in mind for the design of an AC architecture, the system has to feature special requirements that meet the constraints that these devices impose. However, not only should capability constraints be taken into account, but also the characteristics of the use cases in which IoT devices take part. Examples of these use cases are smart cities and homes [17], healthcare applications [18] or smart vehicles [19].

After analyzing the common requirements of these scenarios [4,20,21] and the related work exposed in the previous section, we have identified a set of characteristics that any architecture intending to provide AC for IoT should feature. These characteristics are of course related to the technical requirements. However, when talking about IoT deployments, the way in which the infrastructure is configured and managed is also crucial to ensure security and privacy. The authentication and authorization management has to be ensured by the IoT access control model, but it has to be just a delegation of high level security decisions made by the entities that deploy and configure the infrastructure. In other words, the access control model has to enable an administration point to easily translate the security requirements of the applications to the internal devices and service policy management.

Taking this into account, the technical requirements to be covered are the following:

3.1. Application-Scoped

Not every use case has the same security requirements. In order for a single AC system to be used in all of them, while at the same time respecting the requirements that each of them has, the designed system should allow the creation of different authorization policies for different applications and their services. When it comes to checking if a given IoT device has the right permissions to perform a certain action against a service, only and exclusively the correspondent policy should be enforced. By fostering this behavior, a device could have different permissions for different applications, thus making it possible for the same device to be reused among different services, being under different security conditions in each of them.

3.2. Client-Independent

One of the key ideas behind IoT is that it should integrate with the rest of the Internet, and this interoperability could not be achieved with an AC architecture that considers IoT devices as the only possible clients. Instead, a desirable characteristic would be that the AC system is also compatible with other types of clients, including heavier clients such as web browsers or even other services. A single AC architecture should feature authorization mechanisms that fit the constraints of IoT devices, yet being able to be used by other clients that want to access the same services.

3.3. Flexible

In order for the AC architecture to be used in all security contexts, the framework for describing authorization policies should fit the level of granularity that the service administrator desires. For this purpose, (a global nonprofit consortium that works on the standards definition for security, IoT and other areas) standardized the eXtensible Access Control Markup Language (XACML) [22], which allows the definition of fine-grained policies. XACML serves as a standard not only for the format of authorization policies and evaluation logic, but also for that of the request/response interactions that take place during an authorization decision. In any case, either this one or any other policy description language should be used, so that a multi-tenant authorization scheme can be created in the exact way that the administrator wishes.

3.4. Delegated

One of the most restrictive constraints of IoT devices is related to their low computing capabilities, usually due to battery limitations. Detaching authorization operations from the rest of entities would provide computational lightness for clients, as well as scalability and ease of remote configuration for the service administrator, because the whole access control functionality would be provided centrally and as a service.

3.5. Configurable

The access control model has to be agnostic of the specific security requirements of each IoT deployment. This means that the architecture has to enable an administration point to easily configure policies and permissions for each of the available devices and depending on the service they are going to access.

4. IoT Application-Scoped Access Control as a Service

After analyzing the requirements for access control in IoT, in this section, we propose an AC model that fulfills the requirements that were identified above. As explained in Section 2, for designing our solution, we take ideas from the existing solutions in the literature. We adopt the use of the OAuth 2.0 protocol to manage RBAC- and ABAC-based authorization policies with regard to sensors. We of course reuse the concept of the policy enforcement point to secure the protected resources in the system. However, we change in our model the way in which the authorization roles of subject and object are assigned to the entities. Making the device the subject that performs publish/subscribe requests to a service allows us to remove the token validation from them, decreasing the computational load. This decision also enables the possibility of having application-scoped authorization, with the same device acting with different roles depending on the service or application.

Figure 1 shows the detailed architecture scheme of our model. We have considered the following entities:

- an IoT sensor or actuator (called *Device* for short);
- a *Service*, i.e., an application whose resources are willing to be accessed by the *Device* and to be secured by the proposed architecture;
- a *Service Admin* in charge of managing the security policies related to the *Service*;
- an Identity Provider (*IdP*), which includes relevant features like credentials' management, *Service* registration and management of groups and roles;
- the Policy Administration, Decision and Enforcement Points (*PAP*, *PDP* and *PEP*), which comprise the widely-known access control architecture [23];
- and a *Policies DB* where policies are stored by the *PAP* and checked by the *PDP*.

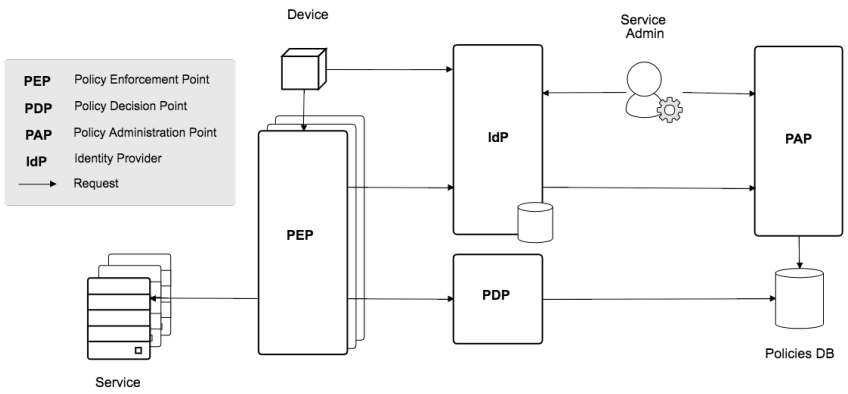

Figure 1. As a service access control architecture.

In the following subsections, the most important components of the model are deeply described. Finally, some sample flows are provided to illustrate the interaction between them.

4.1. Access Control Policies

Following the XACML terminology, policies are composed by a set of rules (as well as other items that are out of the scope of this paper). Rules are in turn made of a target (e.g., the resource), an effect (e.g., allow/deny) and a condition [22]. In our approach, rules are used to implement permissions, in which the target is an action (e.g., an HTTP verb) plus a resource of the *Service*. Of course, more complex policies may be also defined, provided that they are supported by the policy-description language. *Roles* are in turn sets of *Permissions* that serve as a container so that the *Service Admin* can assign more than one permission at once.

Both *Permissions*, *Roles* and their relationships are defined in the *PAP*.

Note that this approach allows both a simpler RBAC scheme and a more complex, more flexible ABAC scheme. It all depends on how the *Permissions* are defined when creating the XACML rules.

4.2. The Identity Provider

The *IdP* is particularly worth noting, because it is in charge of some of the key tasks of the system.

4.2.1. Credentials Management

The *Service Admin* and the *Device* need to be registered in the *IdP*, so that they get a set of credentials (usually a username and a password), which they can use to authenticate and identify themselves against the AC system. Additionally, the *IdP* could also provide identity to other users of the application, following an Identity as a Service (IDaaS) approach [24,25].

4.2.2. Service Registration

This is the basis for providing application-scoped AC. As *Roles* and *Permissions* are described and assigned in the scope of the *Service*, the *Service Admin* needs to register it in the first place. As a result of the registration, the *Service Admin* obtains the associated OAuth 2.0 credentials, which must be stored in the *Device*, and the *PEP* credentials. More details on the latter component are given later.

4.2.3. Groups Management

An interesting feature is that groups of devices (or users) can be created, to allow more complex authorization scenarios. Granting a *Role* to a group of devices, rather than to a single device, has the benefit of linking the given *Role* to the devices belonging to the group. For instance: devices representing the streetlights in a park *A* in a smart city application must be the only ones able to write data in the *Service* about park *A*; therefore, when one of those devices moves to a park *B* (e.g., a subdivision is created), removing it from group *A* is enough to deny its access to resources of its past park.

4.2.4. Roles Assignment

Although policies (in terms of *Permissions*, *Roles* and their relationships) are defined in the *PAP*, it is in the *IdP* where the mapping among them and users, *Devices* and/or groups of them is made. *Roles* are assigned to *Devices* in the scope of the *Service*, so a given *Device* may have different roles depending on the application.

4.3. The IoT Device

Requests from the *Device* may target two components of the architecture: either the *IdP* or the *PEP*. In the case of the former, the *Device* will make use of the Implicit OAuth 2.0 grant [16] to

obtain an access token. As explained before, IoT devices have special requirements with regard to computational capabilities and network connectivity [26]:

- Device heterogeneity: IoT is characterized by a large heterogeneity in terms of devices taking part in the system. The use of standard technologies to perform the needed operations benefits the interoperability and the implementation independently of the device type.
- Energy limitations: Minimizing the energy to be spent for communication/computing purposes is a primary constraint. The less operations we perform in the device, the better the performance and durability of the system.
- Network connectivity: In many cases, entities in IoT are provided with short-range wireless communications capabilities. Using light communication protocols also benefits the system performance and stability.

OAuth 2.0 is a standard protocol that can be supported by a range of technologies and programming languages [27]. This makes it easy to integrate in any device. On the other hand, the use of HTTP or CoAP makes the communication extremely light [12,13]. Therefore, this approach meets the computational restrictions that the *Device* imposes. In fact, any entity implementing an OAuth 2.0 client library is eligible for working as a client in this architecture; thus, interoperability with other non-IoT systems is guaranteed.

The *Device* will include the access token as a header when making a request to the *Service*. Note that, as can be seen in Figure 1, requests to the *Service* are actually intercepted by the *PEP*.

4.4. The Policy Enforcement Point

During an authorization check, the *PEP* is the entity that uses the access token (which the *Device* previously obtained from the *IdP*) to retrieve from the *IdP* the roles assigned to the *Device*. Therefore, a mapping between applications and OAuth 2.0 consumers can be made. To perform that request, the *PEP* will also use its own credentials, which the *Service Admin* obtained from the *IdP* when registering the *Service*, and later set in the *PEP*. Once roles have been retrieved, the authorization check is sent to the *PDP*.

It is important to highlight that, in order for the PEP to intercept the requests to the *Service* and enforce authorization policies, a different *PEP* has to be set for each *Service*. This way, a completely transparent, as a service authorization system is provided, because the only thing that the *Service Admin* must take care of (apart from configuring credentials and policies) is deploying an instance of the *PEP* along with the *Service*. Besides, the AC provider will be most probably providing and supporting the implementation of the *PEP* component, in order to foster this procedure.

4.5. The Policy Decision Point

Once the access token has been used to obtain the roles associated with the authorization context (*Device* + *Service*) by the *PEP*, the second step consist of checking the authorization with *PDP*. *PDP* fetches the policies associated with those roles from the *Policies DB* and decides whether or not access should be granted based on them. This way, authorization is completely delegated and externalized from the *Service*.

4.6. 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 to a back-end *Service*. Our objective is to add an AC mechanism that allows restricting access to that application depending on:

- the type of IoT device,
- the service resource that it is trying to access,

- and the action that it is trying to perform with regard to the service resource.

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*.

4.6.1. Initial Configuration

The first step consists of the registration and configuration of the *Service* and the *Device*. Figure 2 shows a schema of the components interaction. Every operation below is triggered by the *Service Admin*:

1. The *Service* is registered in the *IdP*, to obtain both the OAuth 2.0 credentials for its application (i.e., the `client_id` and `client_secret`) and those associated with its corresponding *PEP* (i.e., the `pep_username` and `pep_password`).
2. The *Device* is registered in the scope of the *Service*, to obtain the *Device* credentials (i.e., the `device_username` and `device_password`).
3. The desired roles and policies are created in the *PAP*, and the relationships between them are equally defined. The *PAP* will then store those entities and their relationships in the *Policies DB*.
4. The desired roles are granted 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 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*).

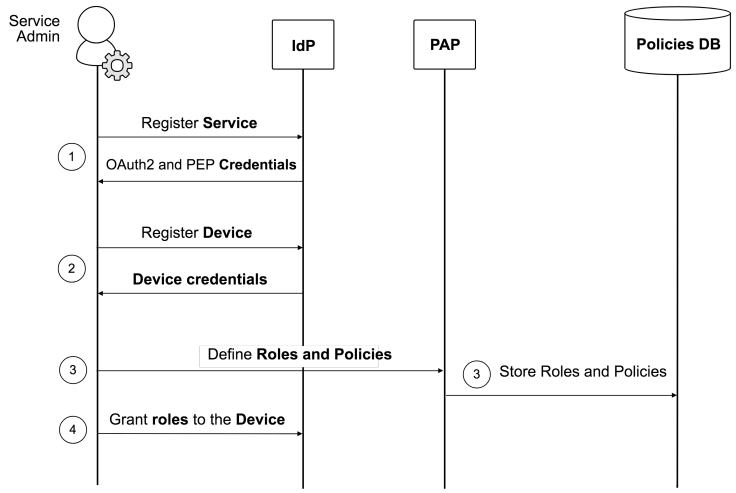

Figure 2. Registration and configuration of the *Service* and the *Device*.

4.6.2. Securing Requests

Once the environment is configured, the process to send a publish request from the *Device* to the *Service* consists of the following interactions and is illustrated in Figure 3:

1. The *Device* sends a request to the *IdP* in order to create an OAuth 2.0 access token. 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 OAuth 2.0 token that was created in the previous step.
3. The *PEP* module intercepts the request and extracts the token from the header. After that, it sends a validation request to the *IdP* in order 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* makes 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.

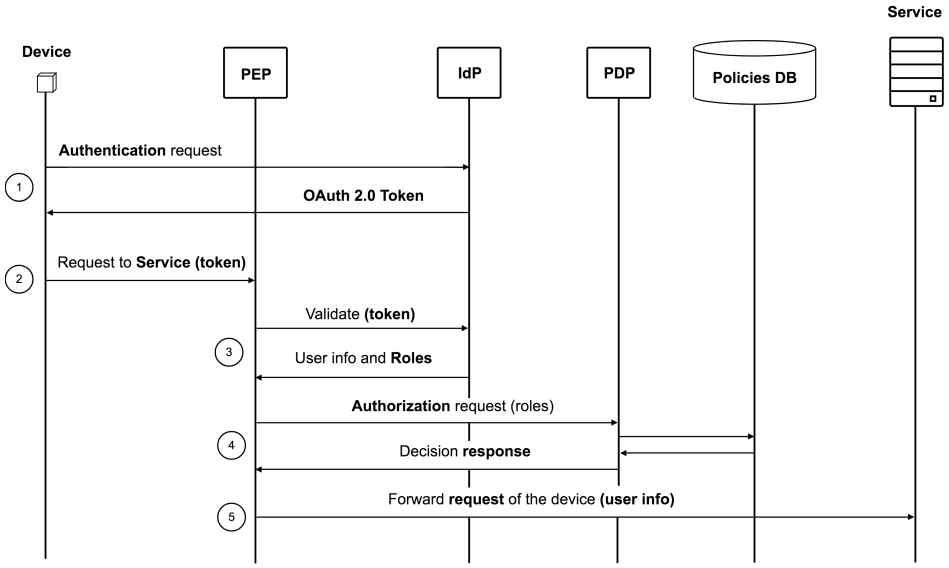

Figure 3. Device request and authorization flow.

5. Implementation and Results

To validate the proposed solution, the authors have implemented and deployed the model in the scope of the FIWARE European project (https://www.fiware.org [Last accessed on 3 October 2017]). The objective of FIWARE is to provide a framework for developing smart applications for the future Internet. To achieve this, it creates a service ecosystem based on key elements called Generic Enablers (GEs), which compose a frame that allows the development of the new future Internet applications. To add security to this framework, the FIWARE Security Team develops a set of GEs with the main objective of providing a single identity service for the rest of the GEs. This way, GE developers provide FIWARE users the possibility of easily logging in using a single account: the one provided by the FIWARE identity service. Moreover, access to the protected resources that those services expose is managed following the FIWARE Security basis, which implements an AC mechanism that follows the architecture presented in this paper.

Each FIWARE GE is defined by an open specification that describes relevant information for users to consume the implementations corresponding to the GE and/or to build compliant services that can work as alternative implementations of GEs developed by FIWARE. Moreover, each GE has a Generic Enabler reference implementation (GEri) that implements the defined open specification and that is used as a reference for the other implementations that could appear in the ecosystem. These implementations are totally open-source and maintained by the community of FIWARE developers.

5.1. Implementation

The different components exposed in Section 4 have been implemented in the form of GEri's. A brief description of their implementation is outlined below. For each component, we include a discussion of the challenges faced to extend or improve the existing frameworks or solutions.

5.1.1. Identity Provider

The Identity Management GEri is named KeyRock (https://catalogue.fiware.org/enablers/identity-management-keyrock [Last accessed on 3 October 2017]). As one of the main services that the FIWARE ecosystem offers to its users is a public, Openstack-based cloud infrastructure, KeyRock has been implemented using Openstack technology as a starting point. This way, interoperability between it and the cloud infrastructure is ensured.

KeyRock has two main components, named after their Openstack's counterparts: Python-based, back-end Keystone and Django-based, front-end Horizon.

Openstack's Keystone (https://docs.openstack.org/developer/keystone [Last accessed on 3 October 2017]) has many of the functionalities that KeyRock requires, but not all of them. This is why a set of extensions had to be developed and included in KeyRock's Keystone. The most important ones for the purpose of this work are one that integrates the OAuthLib library (https://github.com/idan/oauthlib [Last accessed on 3 October 2017]) to support OAuth 2.0 and another one that is in charge of the management of roles and permissions and its application-scoped assignment. The latter one overwrites the vanilla roles engine, which recedes into the background. This component makes use of an SQL database for persistence.

KeyRock's Horizon serves as a friendlier user interface for users and administrators to manage applications, AC policies, etc., as opposed to the REST API offered by KeyRock's Keystone. This component also extends Openstack's front-end service Horizon (https://docs.openstack.org/developer/horizon [Last accessed on 3 October 2017]), to include the needed modules so as to fit the proposed model. These modules are a set of business logic (and their correspondent views) that helps in the tasks that were described in Section 4.2, such as the registration and management of OAuth 2.0 applications, PEP components, IoT devices, user accounts and the assignment of roles and permissions.

The existing OAuth 2.0 libraries offer support for managing access tokens and application registration. However, all the features related to IoT devices and PEP components' management are new for our model implementation. Therefore, we had to include the new entities and to integrate them with the existing data models to offer our application-based access control mechanism. On the other hand, the ability to provide a user-friendly interface to manage this kind of low-level entities was also a challenge to take into consideration.

5.1.2. Policy Administration and Decision Points

This implementation integrates both *PAP* and *PDP* components in the same module providing two APIs:

- Policy decision point API : provides an API for getting authorization decisions computed by a XACML-compliant access control engine;
- Policy administration point API : provides an API for managing XACML policies to be handled by the authorization service PDP.

The full API (RESTful) is described by a document written in the Web Application Description Language format (WADL) and an associated XML schema. The name of the component is AuthZforce (https://catalogue.fiware.org/enablers/authorization-pdp-authzforce [Last accessed on 3 October 2017]) and has been developed by a group of developers belonging to the FIWARE Security Team, not directly by the authors of this work. The implementation provides an API to get authorization decisions based on authorization policies and authorization requests from PEPs. The API follows the REST architecture style and complies with XACML v3.0.

This component is agnostic of the type of clients that manage and consume the policies. Therefore, its implementation does not offer relevant challenges to be considered.

5.1.3. Policy Enforcement Point

The policy enforcement point GEri is named Wilma (https://catalogue.fiware.org/enablers/pep-proxy-wilma [Last accessed on 3 October 2017]) and has been fully developed by the authors of this manuscript, using Node.js. In the context of FIWARE, it is usually referred to as *PEP Proxy*, to emphasize the fact that it intercepts requests to the service it has been deployed on top of (as explained in Section 4.4). Wilma can be configured to check three levels of security:

1. Authentication: Using this level of security, the *PEP* just checks if the client has been correctly authenticated against the *IdP*. Thus, at this level, every user with an active account would be able to access the protected resources. The check is performed by sending a validation request to the *IdP*.
2. Basic authorization: In this case, the *PEP* also checks if the client has the required roles to perform the corresponding action (defined by an HTTP verb) in the corresponding resource (defined by an HTTP path). The check is performed by sending a policy request to the *PDP* module.
3. Advanced authorization: This is the most complex, powerful case, because the authorization check is not only based on the HTTP verb and path, but also on other more advanced, customizable parameters, such as the request body or headers. To perform the check, a custom XACML policy request is sent to the *PDP*.

The main challenge when implementing this component was related to the necessity of having a single PEP component for each registered application in the platform. Thus, the registration and the authentication mechanisms of the component for being able to validate requests was a new feature not present in other existing frameworks.

5.2. Deployment

One of the key points of FIWARE is FIWARE Lab (https://account.lab.fiware.org [Last accessed on 3 October 2017]), a public working instance of FIWARE GEri's available for experimentation. The entry point of this laboratory is an instance of our KeyRock IdP implementation, where developers can create an account and start registering applications, IoT devices, etc., for free. FIWARE Lab also includes an AuthZForce (the PDP and PAP component explained above) PDP and PAP instance, connected to the IdP to manage permissions and AC policies. Furthermore, the implementations of the GEs are available to be deployed in the FIWARE Cloud ecosystem, which is based on Openstack. This includes the Wilma PEP Proxy, which can be instantiated and deployed on top of a service with just a click, by running the available software image.

FIWARE Lab offers cloud infrastructure in 13 regions across Europe (http://infographic.lab.fiware.org [Last accessed on 3 October 2017]), with more than 8500 registered users as of the date this paper was written. Security components are deployed in the Spanish node, owned by one of the partners of the project consortium.

Figure 4 shows in detail the deployment in which the experiments exposed in this paper have been performed. Table 1 describes the hardware used to deploy each component. As explained before, the IdP is based on Openstack and therefore split into two components, Horizon and Keystone, for the front-end and the back-end, respectively. In order to provide a high availability solution, we deployed two instances of Keystone, which is the module that handles a higher number of requests. Both of them are connected to the same MySQL database, hosted in the Keystone 2 instance. An external disk is mounted in this instance, in order to store the data that are periodically backed-up from the database. The High Availability (HA) proxy just redirects the incoming requests to one of the Keystone instances. Finally, the *Policies DB* is hosted in the same instance where the AuthZForce component is running. The figure also shows the public ports exposed by each component.

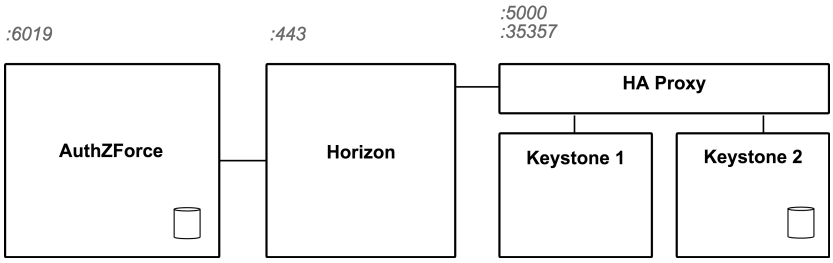

Figure 4. Deployment configuration.

Table 1. Deployment servers characteristics.

Component	Operating System	CPU	Memory	Disk
Horizon	Ubuntu 14.04	8 virtual CPU	8 GB	14 GB
Keystone 1	Ubuntu 14.04	8 virtual CPU	8 GB	14 GB
Keystone 2	Ubuntu 14.04	8 virtual CPU	8 GB	100 GB + 500 GB
AuthZForce	Ubuntu 14.04	4 virtual CPU	4 GB	18 GB
HA Proxy	Ubuntu 14.04	8 virtual CPU	16 GB	14 GB

5.3. Use Case Analysis

To illustrate how the proposed model can be applied to a real use case scenario, we provide a sample smart city setup that we have tested in the environment deployed at FIWARE Lab. Table 2 shows a summary of the services and devices we have registered for this scenario, as well as the roles and permissions we have defined and the corresponding assignments between them.

Table 2. Devices' summary.

Service	Device	Role	Permission
Parks and gardens	$D1_n$ (presence)	R1	P1: `POST /parks/{id}/presence`
	D2 (streetlight)	R2	P2: `GET /parks/{id}/presence` P3: `POST /parks/{id}/luminosity`
	D3 (web panel)	R3	P2: `GET /parks/{id}/presence`
Electricity	D2 (streetlight)	R4	P4: `POST /lights/{id}/status`

On the other hand, Figure 5 shows a diagram of the designed system. We can see how the services administrators (in this case, the administrators of the *Parks and gardens* and of the *Electricity* services) are the Service Admins that manage roles and permissions in the identity provider . They have to register their services in the IdP and then start registering devices and assigning roles to them. Assuming that the services would be provided by the local city government, devices can be shared between services. In the example, streetlights access both services as explained below.

The first service, *Parks and gardens*, is in charge of the monitoring and management of the green areas around the city. One of the features that it offers consists of turning on and off the streetlights of a park depending on the presence of people around. To achieve this, we register a set of presence sensors ($D1_n$) that periodically publish in the *Parks and gardens* service the presence status in different zones around the streetlight. This is possible because we define a *P1* permission that allows `GET` requests to the specific resource in the service (`/parks/{id}/presence`). This permission is assigned to the *R1* role, which is in turn assigned to the sensors. Note that, to facilitate this task and thanks to the multi-tenancy feature of the proposed model, we define a group of devices that contains every $D1_n$ sensor, and we just assign the *R1* role to the group.

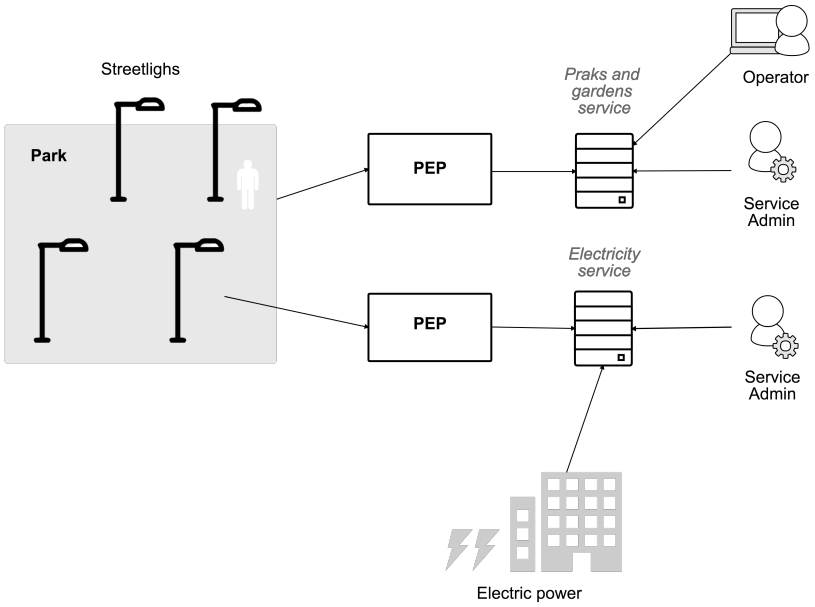

Figure 5. Smart city diagram.

On the other hand, the streetlight periodically retrieves the presence status provided by the sensors around it and acts accordingly by turning on or off the light. To be able to read this information from the service, we define and assign the role *R2* with the permission *P2*, which permits `POST` requests to the presence resource. Streetlights also have a sensor that reports the luminosity to the *Parks and gardens* service; we have created a new permission *P3* that allows this action, and we have assigned it to the role that the device already owns (*R2*).

Finally, to monitor the density of people in the parks, operators can visualize the presence status reported by $D1_n$ sensors on a web page using their PCs/smartphones. This web page can read this information because role *R3* allows it thanks to permission *P2* (we reuse here the one we created for the streetlights). As can be seen, not only IoT devices can benefit from our AC mechanism: every client capable of creating an OAuth 2.0 token is eligible.

A second service, *Electricity*, has been registered to illustrate the application-scoped feature of the proposed model. This service simply receives the status of the streetlights (on/off) to monitor the electrical consumption. A new role and permission are assigned to this device (*D2*) to permit the action. We can see how, depending on the service that is trying to be accessed when creating the token, a different role applies for *D2* (*R2* or *R4*).

5.4. Results

The setup of the scenario explained above illustrates how the proposed model covers the system requirements exposed in Section 3. The use of the single device (*D2* streetlight) for accessing different services with different permissions is a good example of how the model covers the *Application-scoped* requirement. This device has read and write permissions in one service and just write permission in another. Furthermore, the resources the device can access are different in each service.

On the other hand, using two types of clients (a web browser application and a set of devices), we test the *Client-independent* requirement. Thanks to the designed model, any client capable of creating an OAuth 2.0 token can authenticate in the system and thus can send authorized requests to the back-end service.

The configuration of custom permissions by HTTP verbs and paths allows us to decide which kinds of actions (write, read, delete, etc.) are allowed for each specific resource. In this use case, we have just configured this kind of permissions, but as explained before, the use of XACML policies enables the use of advanced authorization decisions based, for instance, on the HTTP headers or the

requests content. This makes the architecture flexible and easily adjustable to the requirements of each use case.

Finally, removing the authorization checks from the device and delegating them to external components (IdP, PEP and PDP) decreases the computational load in the devices. As explained before, this is critical when talking about sensors and actuators with light capabilities. Moreover, as outlined in Section 2, the policy enforcement point protects the access to the object of the authorization architecture. In the deployed use case, sensors act as subjects, and therefore, the performance comparison with devices acting as PEPs does not directly apply here.

6. Conclusions and Future Work

AC is an essential topic when dealing with online services. However, when IoT devices are considered to be interacting with those services, some extra considerations have to be borne in mind, because not every AC approach is suitable for these constrained devices.

In this paper, we have presented an architectural model that enables AC in IoT contexts by using the OAuth 2.0 protocol. On the one hand, OAuth 2.0 ensures that the model fits low-capable devices, because tokens provide all the required information to perform the authentication process, thus making it extremely light. On the other hand, OAuth 2.0 makes this model interoperable with other RESTful services on the rest of the Internet and also with any other client apart from IoT devices.

Besides, our externalized approach enables an as a service authorization service that can be used by application developers as an additional layer to their applications in a seamless way. Moreover, the fact that the AC system is completely external to services facilitates the configuration of authorization policies, since they are all managed from a central, single component.

Furthermore, this design has been implemented and deployed in the scope of FIWARE, which lets us conclude that the proposed model satisfies the special requirements of IoT devices in terms of performance, scalability and interoperability.

As a future research line, some improvements could be considered. In cases when storing the OAuth 2.0 credentials in the IoT device constitutes an issue (e.g., access to the device could be compromised), an approach like using a different OAuth 2.0 grant could be considered. Moreover, the whole process could be made even lighter by supporting the CoAP protocol in the interactions in which the IoT devices take part. On the other hand, the performance of the proposed solution in large IoT deployments should be evaluated. Then, auto-scalable scenarios that adapt the available resources to the system demand can be also explored.

Acknowledgments: The authors would like to thank the FIWARE European Project and its partners. Thanks to FIWARE Lab, we have achieved the deployment and testing of the proposed ideas in real scenarios.

Author Contributions: Álvaro Alonso conceived of and designed the proposed model. Federico Fernández implemented and tested the use case scenario. Lourdes Marco studied the related work, and Joaquín Salvachúa reviewed the contribution.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lin, H.; Bergmann, N.W. IoT Privacy and Security Challenges for Smart Home Environments. *Information* **2016**, *7*, 44.
2. Chan, H.; Perrig, A. Security and privacy in sensor networks. *Computer* **2003**, *36*, 103–105.
3. Medaglia, C.M.; Serbanati, A. An Overview of Privacy and Security Issues in the Internet of Things. In *The Internet of Things: 20th Tyrrhenian Workshop on Digital Communications*; Springer: New York, NY, USA, 2010; pp. 389–395.
4. Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.* **2013**, *29*, 1645–1660.

5. Fernández, F.; Alonso, A.; Marco, L.; Salvachúa, J. A model to enable application-scoped access control as a service for IoT using OAuth 2.0. In Proceedings of the 20th Conference on Innovations in Clouds, Internet and Networks (ICIN), Paris, France, 7–9 March 2017; pp. 322–324.
6. Roman, R.; Zhou, J.; Lopez, J. On the features and challenges of security and privacy in distributed internet of things. *Comput. Netw.* **2013**, *57*, 2266–2279.
7. Sicari, S.; Rizzardi, A.; Grieco, L.; Coen-Porisini, A. Security, privacy and trust in Internet of Things: The road ahead. *Comput. Netw.* **2015**, *76*, 146–164.
8. Sandhu, R.S.; Coyne, E.J.; Feinstein, H.L.; Youman, C.E. Role-based access control models. *Computer* **1996**, *29*, 38–47.
9. Yuan, E.; Tong, J. Attributed based access control (ABAC) for web services. In Proceedings of the 2005 IEEE International Conference on Web Service, Orlando, FL, USA , 11–15 July 2005.
10. Zhu, Y.; Huang, D.; Hu, C.J.; Wang, X. From RBAC to ABAC: Constructing flexible data access control for cloud storage services. *IEEE Trans. Serv. Comput.* **2015**, *8*, 601–616.
11. Hussein, D.; Bertin, E.; Frey, V. A Community-Driven access control Approach in Distributed IoT Environments. *IEEE Commun. Mag.* **2017**, *55*, 146–153.
12. Hemdi, M.; Deters, R. Using REST based protocol to enable ABAC within IoT systems. In Proceedings of the 7th Annual Information Technology ,Electronics and Mobile Communication Conference (IEMCON), Vancouver, BC, Canada, 13–15 October 2016; pp. 1–7.
13. Kinikar, S.; Terdal, S. Implementation of open authentication protocol for IoT based application. In Proceedings of the 2016 International Conference on Inventive Computation Technologies (ICICT), Tamilnadu, India, 26–27 August 2016; Volume 1, pp. 1–4.
14. Cirani, S.; Picone, M.; Gonizzi, P.; Veltri, L.; Ferrari, G. Iot-oas: An oauth-based authorization service architecture for secure services in iot scenarios. *IEEE Sens. J.* **2015**, *15*, 1224–1234.
15. Emerson, S.; Choi, Y.K.; Hwang, D.Y.; Kim, K.S.; Kim, K.H. An oauth based authentication mechanism for iot networks. In Proceedings of the 6th International Conference on Information and Communication Technology Convergence, Jeju Island, Korea, 28–30 October 2015; pp. 1072–1074.
16. Hardt, D. *The OAuth 2.0 Authorization Framework*; RFC 6749; RFC Editor, Dick Hardt, Microsoft: Redmond, WA, USA, 2012.
17. Korzun, D.G.; Balandin, S.I.; Gurtov, A.V. Deployment of Smart Spaces in Internet of Things: Overview of the Design Challenges. In *Internet of Things, Smart Spaces, and Next Generation Networking, Proceedings of the 13th International Conference, NEW2AN 2013 and 6th Conference, ruSMART 2013, St. Petersburg, Russia, 28–30 August 2013*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 48–59.
18. Laplante, P.A.; Laplante, N. The Internet of Things in Healthcare: Potential Applications and Challenges. *IT Prof.* **2016**, *18*, 2–4.
19. Keertikumar, M.; Shubham, M.; Banakar, R.M. Evolution of IoT in smart vehicles: An overview. In Proceedings of the 2015 International Conference on Green Computing and Internet of Things (ICGCIoT), Delhi, India, 8–10 October 2015, pp. 804–809.
20. Singh, D.; Tripathi, G.; Jara, A.J. A survey of Internet-of-Things: Future vision, architecture, challenges and services. In Proceedings of the 2014 IEEE World Forum on Internet of Things (WF-IoT), Seoul, Korea, 6–8 March 2014; pp. 287–292.
21. Lee, I.; Lee, K. The Internet of Things (IoT): Applications, investments, and challenges for enterprises. *Bus. Horiz.* **2015**, *58*, 431–440.
22. *Extensible access control Markup Language (XACML)*, version 3.0; OASIS: Boston, MA, USA, 2013.
23. Turkmen, F.; Crispo, B. Performance evaluation of XACML PDP implementations. In Proceedings of the 2008 ACM Workshop on Secure Web Services, Alexandria, VA, USA , 27–31 October 2008; pp. 37–44.
24. Emig, C.; Brandt, F.; Kreuzer, S.; Abeck, S. Identity as a Service—Towards a Service-Oriented Identity Management Architecture. In *Dependable and Adaptable Networks and Services, Proceedings of the 13th Open European Summer School and IFIP TC6.6 Workshop, EUNICE 2007, Enschede, The Netherlands, 18–20 July 2007*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 1–8.
25. Ducatel, G. Identity as a service: A cloud based common capability. In Proceedings of the 2015 IEEE Conference on Communications and Network Security (CNS), Florence, Italy, 28–30 September 2015; pp. 675–679.

26. Miorandi, D.; Sicari, S.; Pellegrini, F.D.; Chlamtac, I. Internet of things: Vision, applications and research challenges. *Ad Hoc Netw.* **2012**, *10*, 1497–1516.
27. Sciancalepore, S.; Piro, G.; Caldarola, D.; Boggia, G.; Bianchi, G. OAuth-IoT: An access control framework for the Internet of Things based on open standards. In Proceedings of the 2017 IEEE Symposium on Computers and Communications (ISCC), Crete, Greece, 3–6 July 2017; pp. 676–681.