An implementation XACML implementation architecture for IoT use-case scenario

..., Vedran Semenski, ...

Instituto de Telecomunicações, DETI,

University of Aveiro, Aveiro, Portugal

vedran.semenski@ua.pt

*Abstract* - The OASIS XACML statndard defines a language for defining access control requests and policies. It is intended to be used Along with It Along with the language, the standard defines an architecture and workflow. The architecture defines some aspects but doesn't define the distribution of these components over different machines and doesn't deal with securing communication between components. This paper will propose and modified architecture and present a proof of concept in an IoT, Smart City use-case scenario.

The solution utilizes a modified version of the architecture from the XACML standard [4] described in subsection 2.1.2, and two proposed general methods for implementation/integration with other systems which is described in subsection 4.3. Because of the qualities inherited from ABAC and XACML, it offers great flexibility for a number of possible use-cases, although the primary ones in for this work will be IoT applications

The Extensible Access Control Markup Language (XACML) is a platform independent standard based access control policy specification language. It defines rules on how authorization decisions from evaluating applicable access control policies are combined. However, it fails to incorporate built-in trust and privacy-enhancing mechanisms. There are some possible attacks that are identified in the specification that can potentially breach the security of a system using XACML. They are: unauthorized disclosure, message replay, message insertion, message deletion and message modification. In addition to these, there are no mechanisms in place to ensure the confidentiality and integrity of messages in transit between the components of the standard XACML architecture. This paper will briefly investigate the security loop holes in the XACML architecture and proposes an architecture that incorporates built-in trust and privacy features.

IoT (Internet of Things) is an area which offers great promise and although a lot of core problems already have satisfactory solutions, security has remained somewhat unaddressed and remains to be a big issue. Access Control is a way of enforcing security that involves evaluating requests for accessing resources and denies access if it is unauthorised, therefore providing security for vulnerable resources. Access Control is a broad term that consists of several methodologies of which the most significant are: IBAC (Identity Based Access Control), RBAC (Role Based Access Control) and ABAC (Attribute Based Access Control). In this work ABAC will be used as it offers the most flexibility compared to IBAC and RBAC. Also, because of ABAC's adaptive nature, it offers longevity and lower maintenance requirements. OASIS (Organization for the Advancement of Structured Information Standards) developed the XACML (eXtensible Access Control Markup Language) standard for writing/defining requests and policies and the evaluation of the requests over sets of policies for the purpose of enforcing access control over resources. It is defined so the requests and policies are readable by humans but also have a well defined structure allowing for precise evaluation. The standard uses ABAC. This work aims to create a security framework that utilizes ABAC and the XACML standard so that it can be used by other systems and enforce access control over resources that need to be protected by allowing access only to authorised subjects. It will also allow for fine grained defining of rules and requests for more precise evaluation and therefore a greater level of security. The primary use-case scenarios are large IoT applications such as Smart City applications including: smart traffic monitoring, energy and utility consumption, personal healthcare monitoring, etc. These applications deal with large quantities (Big Data) of confidential and/or personal data. A number of NoSQL (Not Only SQL) solutions exist for solving the problem of volume but security is still an issue. This work will use two NoSQL databases. A key-value database (Redis) for the storing of policies and a wide-column database (Cassandra) for storing sensor data and additional attribute data during testing.

*Keywords— ABAC; access control; information security;XACML; software architecture;*

# Introduction

The increasing need for integrating security components in a systems is a reason to modify the existing architecture from a implementation perspective.

If the security component is viewed as a "black box" component with two main entry points. One being a PEP and another being a PAP for managing policies.

This requires significant changes to the existing proposed solution for the XACML architecture.

There is an increasing need for information systems integration in organizations. This is because various business applications are developed in different technologies, and run on different platforms. However, it is important to examine if there are any side effects and security implications that could subsequently emerge after the integration process. For instance, departments within an organization may have different information systems, and each information system may have its own proprietary access control implementation. Thus, the integration of these information systems could have various security implications such as inconsistencies in authorization decisions due to many points of enforcement of access control policies within the same organization. Therefore, in order to avoid such inconsistencies and put an appropriate common access control mechanisms in place, the security infrastructure of each information system should be carefully understood before any integration takes place.

Due to the fact that there are various implementations of proprietary access control mechanisms, addressing the security requirements of an organization makes an integration

process reasonably complicated. That is why there is a need for a common standard-based access control policy specification language that could be deployed in heterogeneous environments such as in Web services. Since Web services are mainly designed for the purpose of integration of different applications and platforms, it is very important to have a standard-based access control specification language that could be used by all the interacting parties that ensures only authorized users have access to a resource. The main idea is to find a convenient access control mechanism which can interoperate easily with any information system.

Taking the need for a standard-based access control policy specification language into consideration, the Organization for Advancement of Structured Information Standard (OASIS) ratified the Extensible Access Control Markup Language (XACML) [1]. XACML is believed to be the best candidate for an access control policy specification language in Web services. Although XACML is a powerful tool for specifying access control policies, and processing access requests and authorization decisions, it lacks certain built-in trust and privacy-enhancing mechanisms. There are some possible attacks identified in the specification that can potentially breach the security of a system using the standard XACML architecture [1]. They are: unauthorized disclosure, message replay, message insertion, message deletion and message modification. In addition to these, there are no mechanisms in place to ensure the confidentiality, integrity and availability of access control policies. This paper gives an overview of the XACML architecture and briefly discusses the interactions between the components of the architecture. It emphasizes on the possible attack scenarios that could potentially threaten an XACML-based system. The background section gives a broad overview of the XACML architecture. Section 3 discusses the possible attacks that can take place on an XACML-based system. In Section 4, we propose an XACML architecture with built-in trust and privacy features. Finally, the conclusion section summarizes the paper by highlighting the main concepts discussed in the paper.

The XACML OASIS standard defines a standardised way of writing policies and requests. They can be defined in both JSON and XML format. It defines the procedure of evaluating requests against one or more policies and although meant to be used for enforcing ABAC, it can also be used for RBAC and others. The standard also defines an architecture along with a workflow for it.

This architecture will be analyzed and issues with tit will be presented. This work will also present a distribution scenario

# Background

Access Control is a general term that can be described as a way of securely granting, limiting or denying access to resources therefore protecting the resources from potentially malicious parties.

Before continuing, some key terms need to be explained as they will be used throughout this work:

* **Subject** - this is a term used for the entity that is trying to access a certain resource. This can be a person, but it can also be a process, machine or any other computer system trying to access a resource;
* **Resource/Object** - these two terms will both be used and they represent anything that access control is being enforced upon. This means that a resource can be data from a database, access to an application, service, access to sensors, actuators, facility (room, building), actions over resources, etc. This wide definition is needed because of the wide variety of use-cases;
* **Request** - this is a term that represents the subject's request for a resource. It can be formatted in some way (example: document, file, string) and represent an actual "physical" request (example: requests for data from a database) or not. It can also be the actual "physical" request;
* **Policy** - this term represents one or more rules that the access control system is enforcing when evaluating of a request. This can also be formatted in a document or can be represented "physically" (example: checking the conditions manually before request execution) in the actual implementation.

Access control is a security technique that enforces security over resources by limiting access to them. The access is given only to authorised subjects which can be people or other systems, depending on the implementation. A typical workflow with access control would consist of: receiving a request for a certain resource, evaluating the request against one or more policies, and allowing or denying the request depending on the evaluation result. The systems enforcing access control must have an architecture to facilitate enforcement of access control, an evaluation methodology and well defined policies (or rules) for evaluating the requests. The significance, complexity and size of these, of course, varies from implementation to implementation and can depend heavily on the business layer of the system that is integrating access control.

ABAC (Attribute Based Access Control) is a type of access control that evaluates requests against policies according to attribute values[18]. Attributes are typically divided into three categories:

1. subject - subject/user attributes (examples: age, postal code, IP address...);

2. object - resource attributes (examples: type, value, age...);

3. environment (examples: day of the week, hour of the day...).

These attributes therefore contain data from the subject trying to access the resource, data from the resource that is being accessed and environmental data which represent current conditions. When a request is being evaluated, the decision is made according to these values and conditions/rules defined in policies. Policies are commonly defined in policy files and contain rules and conditions for evaluation. An example policy would be restricting people under the legal age limit to apply for a driving licence. The evaluation would require getting the subjects age (or date of birth) and the legal age limit which would be an environmental attribute. Only after getting these attributes, comparing them and confirming that the subject is over the age limit, the request can be allowed. Policies can, of course, also work on a deny principle and define conditions/rules that would deny access to certain resources. Keeping close to the previous example, one with a deny principle would be defining a policy that would deny access to driving a car (by blocking the car's accelerator pedal) if it is detected that a person has a high alcohol level. Policies therefore contain rules and conditions that have to be defined and met for a decision to be made after comparing all the attributes needed for evaluation.

XACML (eXtensible Access Control Markup Language) is a declarative access control policy language implemented in XML and created by OASIS. It defines a way to evaluate requests for resources according to rules defined in policies. Put simply it is a thought out and standardized solution for implementing access control in software applications[21][22]. It provides a common ground regarding terminology and workflow between multiple vendors building implementations of access control using XACML and interoperability between the implementations [23][24]. It is primarily intended for ABAC but can also be used for RBAC and others.

**XACML REFERENE ARCHITECTURE**

The XACML reference architecture can be seen in Figure 1 this architecture is built out of basic components.

* **PEP** (Policy Enforcement point) - component that performs access control by performing the decision provided by the response. This may also mean fulfilling obligations that come in the response;
* **PDP** (Policy Decision Point) - this component is responsible for evaluating the request against a policy. It contains all the functionality to make the evaluation and produce a response;
* **PIP** (Policy Information point) - This component is responsible for retrieving attributes. The attributes in ABAC are split into three types: subject, environment and resource attributes;
* **PRP** (Policy Retrieval Point) - component used for retrieving of policies;
* **PAP** (Policy Administration Point) - the policy administration point contains the functionality required for managing policies. Typically this means adding, removing and modifying policies.



Figure 4. Architecture proposed by the OASIS XACML standard [4]

Figure 4 shows the architecture proposed in the OASIS XACML standard. Compared to the reference XACML architecture this proposed architecture contains some additional components.

These components are as follows:

* **Context** **Handler** - this entity controls the workflow of the system. It communicates with the PEP, PDP, PIP and resource. As it controls the workflow it has many responsibilities. Mainly, iti has to forward requests from the PEP to the PDP and return the responses from the PDP to the PEP. Additionally it has to fetch attributes when the PDP requests and fetch resource content;
* **access** **requester** - entity that is requesting a resource;
* **obligations** **service** - service that executes any obligations after the evaluation is complete;
* **resource** - entity containing one or more resources and resource attributes that the access requester is trying to access;
* **subjects** - entity containing subject attributes. Typically the subject attributes are attributes of the access requester.
* **environment** - entity containing one or more environmental attributes.

It can be seen that, compared to the reference XACML architecture, the PRP has been removed and the functionality of the PRP has been merged with the PAP. This can be concluded because the PDP fetches policies over the PAP.

As mentioned in[75], the term smart city is widely used, often outside of the computer science context but rather in a more social and cultural context. Definitions therefore vary and many exist, but the final aim is to make a better use of the public resources, increasing the quality of the services offered to the citizens, while reducing the operational costs of the public administrations[76]. The context that is regarded to in this work is as an IoT application scenario. We will define it as:

*"A city utilizing an infrastructure of sensor networks and services to collect and utilize the generated data for the main purpose of improving efficiency and managing of the city, e.g.: traffic, energy and utilities, healthcare, public safety, education etc."*.

IoT, Big Data and security are therefore areas essential to smart cities. They are one of the applications that offer great promise in the improvement of everyday life and because of the infrastructure, could also help in scientific research.

SMARTIE (Smart City) is a European project with the goal of solving security, privacy and trust issues in IoT, in a Smart City implementation. Partners include companies, universities and cities from Germany, Portugal, Serbia, Spain and the UK. As stated on the official website [78] the project officially started on September 1st 2013., and is scheduled to end on August 31st 2016. It has a total budged of 4,862,363 € with the contribution from EU in the amount of 3,286,144 €.

# Issues

Comparing the reference XACML architecture to the one proposed in the OASIS XACML standard v3.0 it can be seen that in addition to additional components, the PRP has been merged with the PAP. Put differently, the functionality of the PRP has been added to the PAP, therefore the PAP is used for retrieving policies.

An issue with removing the PRP and integrating its functionality in the PAP is that the PDP has access to other functionality of the PAP that is outside the scope of what would be in a PRP. This means it can potentially add, remove or modify policies. This is of course an issue as the PDP should not be allowed to do those actions. Separating the PRP from the PAP will remove any possibility of the PDP to misuse the PAP. Additionally, as the PAP is and entry point for system administrators, separation of the PAP means that that workflow is also completely separated from the normal workflow of evaluating policies. This completely removes the system administrator from the rest of the system.

Looking at the architecture from an implementation perspective other issues come up. Some kind of storage solution is needed for storing policies. Commonly this would either be a database or the policies could be stored in a file storage system. For the purposes of the architecture this doesn't matter but for further reference we'll put a Redis database because it is a simple and fast solution.

Keeping in the implementation perspective and viewing the XACML as a security component that needs to provide functionality for enforcing security in other systems.

Reviewing the functionality of the PEP it can be defined as a simple component that needs act accordingly to the response that comes from the PDP. This meant that it needs to fulfil all obligations and pass the request in case of a positive or terminate the request in case of a negative response.

The connection between the Context Handler and the resource is an issue because all information that the PDP needs for evaluation has to be formed as attributes. The fetching of information therefore should be through the PIP because the PIP is responsible for providing additional attributes. Removing that connection the role of the Context Handler forma an implementation perspective becomes a trivial "middle man" in between the PDP's communication with PIP and PEP. The role that the Context Handler still can assume is the initialisation/manager role, handling all other aspects that the other components aren't responsible for handling. This would mainly mean taking care of the initialisation and possibly handling multiple instances.

By removing the Context Handler from the PDP-PIP connection but still leaving it in between the PEP and PDP allows it to have some management functionality. These would include initialisation and configuration, managing multiple instances for a parallel execution scenario and leave it open for expansion if needed.

Another issue is the division of attributes by type. This is regarding the division of attributes in categories as: environment, subject and resource. This is a good way of dividing them when viewing the problem from a logical and functional standpoint. Looking it from a PIP implementation perspective the difference between attributes are not in the information they represent but the type of source they have to fetch it from. For the perspective of the PIP it is not important if the PIP is fetching resource, subject or environment data if it's all coming from the same source or the way of fetching them is the same. For example: if a person is a registered user on a website and wants to change some data on its user profile e.g. telephone number. The resource that the user is trying to access and change, and the attributes of that resource come from the same source as the subject attributes. The methodology of fetching those attributes is also the same. The differentiation of these is therefore pointless from a implementation or PIP functionality perspective. As another example, the environment attributes can easily come from different sources and have much different methodologies for acquiring those attributes. Simple time based environmental attributes can be generated by the system and looked up at the time of evaluation. They don't need any kind of storage or external connections. On the other hand fetching attributes like: legal age limits, tax rates, currency conversion rates etc., is much different and could evolve external connections and special procedures.

Because of this the differentiation of connections for the PIP by attribute type is pointless and a differentiation by source or methodology of acquiring is much more appropriate. The PIP therefore can be split into many PIPs depending on the way the attributes are acquired and the source. A simple example would be having three PIPs. organizes as:

* **Generated Attributes PIP** - responsible for fetching all attributes that can be generated locally without the need to contact any database or external service;
* **Local Attributes PIP** - responsible for fetching attributes that are located on local databases of can be fetched from other local services
* **External Attributes PIP** - responsible for fetching attributes by contacting external services. These would for example be REST services.

The PIPs also need to know which attributes they can acquire and which attributes, if any, are needed to fetch those attributes. The PIPs can be organized in a group and the PDP can than go through the group asking which attributes they can provide and which are needed. When it comes to a match, it requests the attributes and the evaluation continues. Along with dividing the functionality of the PIP by functionality as opposed to type of attributes, this means that the PIPs are modular as one or several can easily be removed or added to the list.

The resulting architecture can be seen in -----

Communication between components and the distribution of components on several machines is not defined the standard. Without enforcing some security measures this leaves thing open to security issues with Confidentiality of access requests and authorization decisions. It is important to put appropriate safeguards in place to protect decision requests and authorization decisions from several attacks. Some of which could be: unauthorized disclosure, message replay, message insertion, message deletion and modification. For instance, consider a typical data flow scenario in an XACML-based system where the PEP sends an XACML request to the PDP. There are no mechanisms in place that ensure whether or not messages in transit are safe from attacks. For example, if an adversary manages to gain access to the communication channel between the PDP and the PEP, he would then be able to intercept XACML requests and authorization decisions easily which in turn enable him/her to insert, modify, interpret or delete messages. This unauthorized disclosure of information causes a compromise to the privacy of the users in the system. In addition to these, the adversary can effectively observe and record legitimate messages which could potentially equip him/her for other attacks such as message replay. Message replay is an attack in which an adversary can easily forge decision requests and authorization decisions using previously recorded legitimate messages. 3.2 Integrity of messages (request or response) in transit There are no mechanisms put in place to ensure the integrity of messages in transit. For example, when the PDP receives an XACML request, how does it know that the request has not been modified while it was in transit? 3.3 Trust between the PDP and PEP.

The main question that may arise here is: how does the PDP ensure whether the XACML request it received was indeed sent from the PEP? Similarly, how does the PEP know whether the authorization decision was indeed sent from the PDP? Therefore it is very important to establish trust between the PDP and the PEP, because if there is an appropriate trust enforcing mechanism in place, the PDP and PEP do not need to be concerned about the identity of senders of a message. 3.4 Trust between the PDP and PAP Similarly, the correctness of the result of evaluation of the XACML request by the PDP depends mainly on the integrity of the access control policies that are created and supplied by the PAP. In addition to this, access control policies that are stored in a policy server should be protected from any threats such as policy modification, deletion and insertion. The PAP has to check the uniqueness of the policies on a regular base. 3.5 Privacy of users

Disclosure of information such as the requestor’s identity in the decision request has a huge impact to the privacy of the users in the system. Appropriate safeguards should be adequately put into force to prevent the communication channel between the PDP and the PEP from being intercepted by an adversary. In addition to this, if the policy server is not secure enough from any kind of unauthorized access, an adversary can gain access to private information of users (service requestors) and misuse it accordingly. Besides this, the communication channel of the access requests and authorization decisions, including all the communications that may occur between the components in the XACML architecture, should be secure enough to prevent from unauthorized disclosure of the information stored in the messages.

After some work was done, a final architecture was settled upon with slight modification to the one initially proposed, and as a result, has slightly deviated from to the architecture proposed in the XACML standard [4].

. ...

# Proposed solution

graph of the architectue



Figure 14. Architecture of the final solution

The final architecture can be seen in Figure 14. The changes do not change the "outside" view of the system but are more of an internal change and more refined solution. The connections to the PIP and PRP are moved from the Context Handler to the PDP so it can fetch policies and all of the attribute information as it needs, while evaluating policies. The PIP is not a single entity but rather a list of PIPs that all have the same interface, and all fulfil the same purpose of fetching attributes. Because some attributes are located on different locations and need to be fetched using different services they need to implement different means of fetching that information. This allows for easy expansion of the PIP functionality and better configuration options. This architecture therefore deals with the issues identified in the initial one. The Context Handler maintains only an initialisation and configuration role rather that handling the workflow and being the "middle man". This was established as being more efficient and was adopted because of that. The PDP now fetches the policies and additional attributes directly from the PRP and list of PIPs, only when it needs to.

graph of the distribution

PEP (Policy Enforcement Point)

The PEP is the point where (as the name states) access control is enforced. This means that this point needs to be located in the system that wants to enforce access control at the exact place inside the workflow where access control is needed. It therefore needs to be robust enough to ensure correct execution and flexible to be implemented on various types of systems. Because of this and reasons explained in Section 4.3 the PEP can be used in multiple ways. It can be implemented by providing it with only a XACML request and depending on the response given act appropriately. This way the system that is implementing the PEP decides what the resulting action will be after the evaluation is finished. The other way is to along with the request, provide the PEP with an object that implements a defined interface IResourceFetcher.



Figure 16. Class diagram of the PEPs

In Figure 16. the class diagram for the PEPs can be seen. The IResourceFetcher is used to ensure that the object provided has methods available for both the positive and negative results of the requests evaluation. With this, the PEP executes the execute() in case the evaluation result is positive and executes terminate() in case of a negative result. The purpose of this is to remove the decision making part from the system that implements the PEP and have it already built in and working. In the case of specific scenarios, the other method of simply getting the evaluation result is also available. The RESTPEP and LocalPEP should never both be available for use by another system and the intent is to have only one PEP available for implementation/integration but they don't collide in functionality. This is explained more in Section 4.3.

These scenarios were made and the components were grouped to run as a single entity for a reason. The components that should be grouped are: PDP, Context Handler, PRP and PIPs. These components are the essential components needed for evaluating the requests. Separation of these components would not bring any benefits, instead it would bring only connection issues and possibly diminished performance. The PIPs can be connected to external services and fetch attributes from outside the system but should not be separated. Additionally the PAP, PAP web application, *Rest Service* and *PolicyDB* manager should be added to this group and should run on the same machine where the Redis database used for storing policies is running. Although this group isn't an essential part to the evaluation process they are endpoints that revolve around the database containing policies. Keeping these together with the rest of the group means keeping communication between components simple, fast and safe without the need of implementing additional safety measures. The PEP needs to be on the machine that is integrating access control.

This method of grouping these components brings up issues regarding scalability. Normally a distributed system scales much better that a non-distributed system and if the components cannot be separated it is hard to have a distributed system. The solution to this would revolve around the replication capabilities of the Redis database used to store policies. The database can be replicated on multiple machines and multiple instances of the solution can run on all of those machines. This would then scale as needed. For this to work with the REST service an additional component would be needed. It would have functionality for handling multiple instances and delegating the workload efficiently. This of course doesn't have to rely on the Redis database and can be exchanged for another storage solution if a better one is found. Because this can be viewed as a service for evaluating requests against policies, it is therefore a single "black box".

Along with scalability, the parallelisation of the process is an issue that has to be considered. This can be achieved using the same principle as before. Having multiple instances of a PDP and providing each one with a subset of policies and running everything parallel is an easy and straightforward way to deal with the parallelisation issue. Long evaluation times in the case of a large set of policies can therefore be split in a fraction of the time by dividing the work and aggregating the result at the end.

# Proof of concept

description of tests and results

Some tests

Two types of tests were performed on every test scenario. The first was a qualitative test running a variety of requests. The goal of this test was the verification of functionality, or to put more precisely, verifying that the system is behaving as expected and returning the expected responses. The requests vary in the complexity and also on the PIPs that their evaluation requires. Some do not require fetching additional attribute data while others require attribute data from multiple sources so all aspects and components of the developed solution were tested . The second test was a performance type of test and it consisted of repeating requests many times, some with a positive and some with a negative response. The response time was timed and because the responses were repeated, it was possible to extract a reliable result. It also has to be noted that the developed solution does not incorporate any type of caching so the repetition of the requests did not result in inaccurate results. It also has to be noted that performance wasn't a primary issue or concern while developing this solution, therefore improving performance could be possible. The request that were used for testing are named with the expected result added as a suffix. This means that a request with the suffix "*\_permit*" or "*\_all*ow" has the expected result *TRUE* and a request with the "*\_deny*" has the expected result *FALSE*.

The last thing that has to be mentioned is that these tests didn't include testing of the PAP Web Application/interface and PAP component. This is because the PAP isn't part of the request/policy evaluation and the test over those parts were done manually by interacting with the PAP Web Application.

The machine used for running local tests and for running/hosting the solution as a service is a HP proBook 4530s with a 8,00 GB of RAM, Intel(R) Core(TM) i5-2450M CPU @ 2.50GHz 2.50 GHz processor package, running the Windows 7, 64bit operating system.

These tests showed that the developed solution performed as intended from a functional perspective and satisfactory from a performance perspective, meaning that the overhead for the response times is acceptable for integrating in other systems. The tests that were done by making calls from the SMARTIE component were also a "proof of concept" test as the primary targeted system was SMARTIE. As the test show, the solution performed as predicted using requests and policies from the target system.

# Conclusion

The ABAC methodology together with the XACML standard, has great potential and offers great benefits with virtually no downsides, which is not something that happens often. A finalized open source implementation that implements every aspect of the standard along with connectivity options with many types of services, would offer great benefits for many implementations, not only IoT applications as mentioned before. After building and having a secure system, verifying that it works correctly and predictably, the potential failure point is no longer directly a point in the system but the interfaces that system administrator and people implementing the solution have to use. The system's security relies primarily on correctly defined policies, making requests that correctly mirror the true requests and integration that is done correctly.

# References

|  |  |
| --- | --- |
| [1] | “eXtensible Access Control Markup Language (XACML) Version 3.0,” 22 January 2013. [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf. [Accessed 4 November 2014]. |