Proposing a secure, implementation-oriented XACML architecture

Óscar Mortágua Pereira[1], Vedran Semenski[2], Rui L. Aguiar[3]
Instituto de Telecomunicações,
DETI, University of Aveiro,
Aveiro, Portugal
{omp[1], vedran.semenski[2], ruilaa[3]}@ua.pt

***Abstract* - The OASIS XACML (eXtensible Access Control Markup Language) standard [1] defines a language for defining access control requests and policies. It is intended to be used with ABAC (Attribute Based Access Control). Along with the language, the standard defines an architecture, workflow and evaluation mechanism. While developing a security component utilizing the standard, a number issues occurred which weren't addressed in the standard. The architecture defines the workflow but doesn't define the way components are distributed over different machines and doesn't deal with securing communication between components therefore leaving some security issues open. This paper will propose a modified architecture, dealing with the mentioned issues and present a proof of concept in an IoT (Internet of Things), Smart City use-case scenario.**

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

I. Introduction

There is an increasing number of information systems, applications and services that are interconnected and dependant on each other. They use a variety of data, cover many domains and are used or integrated in more and more businesses [2]. These systems run on different technologies and different platforms. They can utilize many different workflows, methodologies, storage systems and others. Using many different services over different platforms is often a requirement. Security in these systems is often an issue and dealing with different platforms presents a significant challenge. Other challenges include lower maintenance, ease of integration, and performance. These security issues and requirements can be associated in many areas including: Web applications, IoT(Internet of Things) applications, mobile applications, business information systems as well as services, etc.

Custom security components developed for solving security issues require significant effort to develop, are not unified and cannot be used in other systems and have significant problems in the long terms. Depending on how complex the business layer of an application is, the security component can become complex and less flexible. Depending on how much the structure, architecture or date model changes or expands, issues can occur if the developed component isn't flexible enough to deal with those changes and growth. Organisations can have many departments, use many services, databases...

The OASIS XACML is a platform independent standard that defines a language for writing policies and requests along with an architecture, workflow and methodology of evaluation requests against policies. It is based around ABAC (Attribute Based Access Control) but RBAC (Role Based Access Control) and other types of access control can also use XACML (eXtensible Access Control Markup Language). Because it is standardised and it is made around the ABAC methodology, it offers great potential, flexibility and a standardised way of dealing with security issues. It is one of the best options for dealing with security issues in applications. Its main use is managing access to resources, which can be anything that the user defines (data, actions, services...). It is not meant to deal with connection or communication issues in networks (like for instance security protocols). It is more suited for application and business layer security issues.

While ABAC together with XACML offers great potential, flexibility and many advancements along with a uniformed solution, some aspects are not addressed. The issues that this paper will address are ones that come from an implementation perspective. Put more precisely, it will describe issues that were encountered while developing a security component based on the ABAC and the OASIS (Organization for the Advancement of Structured Information Standards) XACML standard, propose solutions for these issues and present a proof of concept. The issues that will be dealt with are internal and external communication and connection issues and distribution issues.

The increasing need for integrating security components in systems is a reason to modify the existing architecture from a implementation perspective. It is because of this, that the security component is going to be viewed as a "black box" component that can be easily integrated into other systems, easy to use, manage. This should therefore result in a more secure system and requires significant changes to the existing proposed solution for the XACML architecture[1].

This paper is organized in five main sections. Section II will present the background technologies and related. Section III will present the issues that were found in the current proposal in the standard [1]. Section IV will propose solutions for every issue that was mentioned in section III. Section V will present a proof of concept and test results. Section VI will

give an overview of the work that was presented and give a conclusion.

II. Background

Access Control

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

ABAC (Attribute Based Access Control) is a type of access control that evaluates requests against policies according to attribute values [3]. Attributes are typically divided into three categories:

- **subject** - subject/user attributes (examples: age, postal code, IP address...);
- **object** - resource attributes (examples: type, value, age...);
- **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

XACML (eXtensible Access Control Markup Language) is a declarative access control policy language implemented in XML and created by OASIS (Organization for the Advancement of Structured Information Standards). 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 [4][5]. It provides a common ground regarding terminology and workflow between multiple vendors building implementations of access control using XACML and interoperability between the implementations [6][7]. It is primarily intended for ABAC but can also be used for RBAC and others.

OASIS is a non-profit consortium that produces open standards in the areas of security, IoT, cloud computing, energy, content technologies, emergency management, and other areas. It has more than 5,000 participants representing over 600 organizations and individual members in more than 65 countries. The goal of OASIS is to provide standards that offer efficient and open solutions for common problems in the areas mentioned before, drive development and building of standardized open source solutions that can easily be used by many, therefore accelerating innovation and development in general [8].

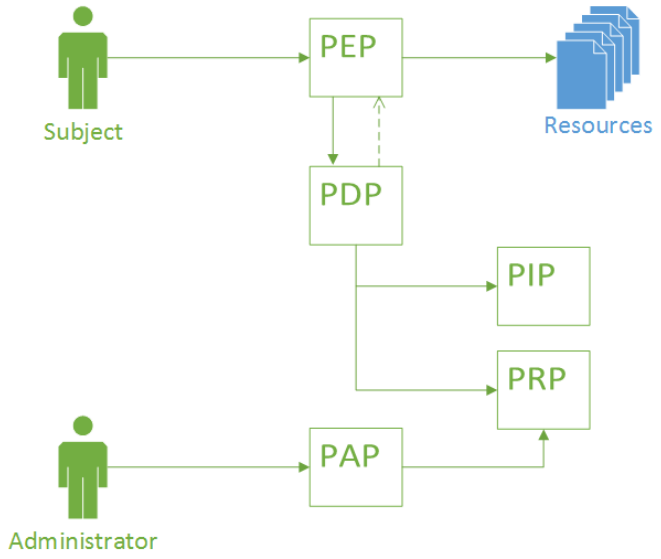

Figure 1. Reference XACML architecture

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 component contains the functionality required for managing policies. Typically this means adding, removing and modifying policies.

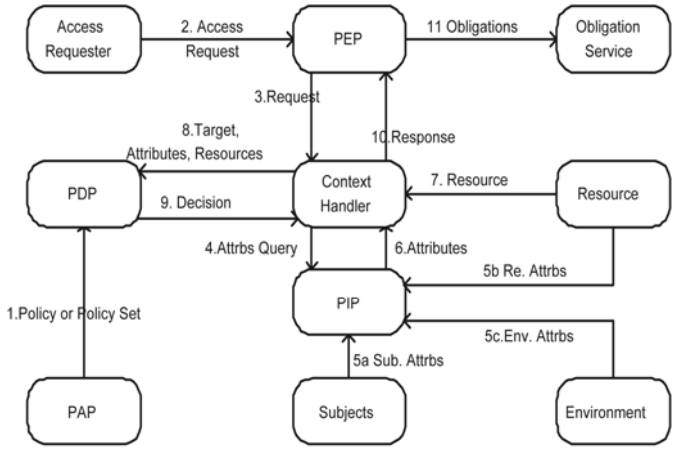

Figure 2. Architecture proposed by the OASIS XACML standard

Figure 2 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, it 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 [9], 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 [10]. The context that will be regarded to in this work is as an IoT application scenario. We will therefore 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 [11] 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 €.

SMARTIE is still in the development stages and was used as a use case scenario for testing an implementation of the architecture proposed in section IV. The test result and test scenarios are presented in section V.

III. Issues

Comparing the reference XACML architecture to the one proposed in the OASIS XACML standard v3.0 [1] 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.

Removal of PRP

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.

Differences between the defined architecture and an implementation

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.

Reviewing the functionality of the PEP, it can be defined as a simple component that needs to 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 (form 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 organized 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.

Communication

Communication between components and the distribution of components on several machines is not defined in the standard[1]. 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. There are several attacks that could be possible. Examples include[2]:

unauthorized disclosure, message replay, message insertion, message deletion and modification. Considering a simple scenario in a XACML-based security component or system, the PEP sends an XACML request to the PDP[2]. The standard doesn't define any mechanism which would ensure that messages were not changed during communication or that the sender and receiver are indeed the ones they represent to be. Without any that connection is not safe from attacks. For example, if an malicious party manages to gain access to the communication channel between the PDP and the PEP, that party would be able to intercept requests and results. This means that it could monitor, modify or even fake requests and responses. Effectively this means that it could potentially gain control over all decisions made and therefore control who gets access to resource. This means that it could monitor the traffic and gain insight into what is happening and collecting information that is potentially confidential. This unauthorized disclosure of information causes a compromise to the privacy of the users and the system itself.

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 unauthorised malicious third parties. In addition the storage mechanism for policies has to be protected against any unwanted connections and basically connections need to be limited only to other components that need to access the policies (PRP).

Distribution

Distribution of components of the XACML architecture isn't addressed. This is an issue because for example, large scale IoT applications require scalability and ability to operate in a vastly distributed system. The standard, does not address performance, scalability and distribution issues. A proposal that would define a way of scaling a security component (or service) would be useful. Distributing the components over many machines could seem to be the correct way of accommodating these requirements but these can create additional issues. Not only connection issues between components but also performance issues that would therefore hurt scalability. Components need to be grouped in a logical manner and a plan for scaling needs to be defined. A proposed solution will be described in section IV.

IV. PROPOSED SOLUTION

Proposes architecture

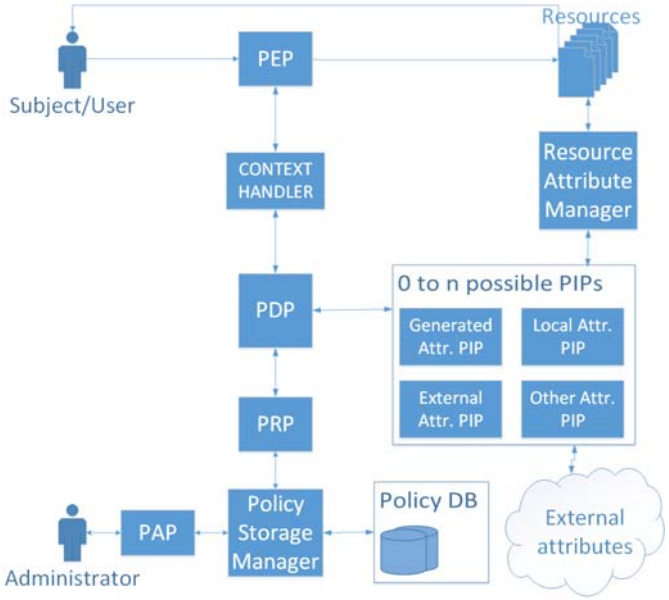

Figure 3. New proposed architecture

The proposed architecture can be seen in Figure 3. 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.

The PEP

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 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*. This is, of course the safer and more straightforward way because it removes any decision making from the implementation because the decisions are made automatically in the PEP.

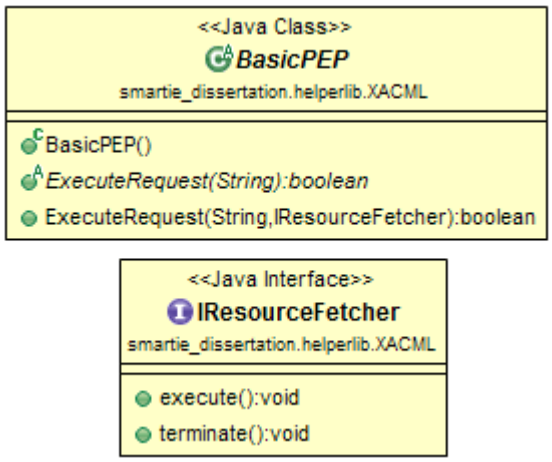

Figure 4. Class diagram of the PEP and additional interface

In Figure 4 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.

Solving the distribution and securing connections

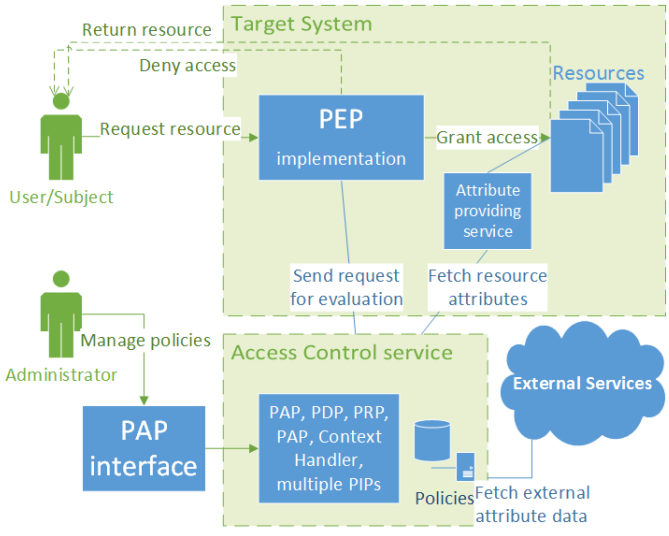

Figure 5. Distribution of components in a use-case

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, connection points to outside components should also be added to this group. These would include components like web interfaces for the PAP, REST service components and any other component over which the communication with the access control service is done. 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.

Some of the issues with connections were identified in the work of [2]. Their solution was to have a centralized entity that would connect to every component over TLS and distribute a token and encrypt messages. This would ensure that the message is unmodified and that the request comes from a authorised and verified source. Because of the grouping of components this is somewhat unnecessary and because of the encrypting it can present unwanted overhead.

The issues in the internal communication between the PDP, Context Handler, PRP and PIPs are no longer an issue if those components are grouped together. The remaining connections that present an issue are the connection between the PEP and the Context Handler and between the PIPs and external sources (including the resource when fetching resource attributes). The problems with these connections are regarding message integrity and validity of both sides. As these communications are most likely be over some kind of internet

connection (for example, over a REST service) the technology to secure them already exist and are proven to work well. A simple and effective way of securing these connections and solving these issues is over a HTTPS connection (SSL/TLS). Using this method provides the authentication to both parties involved in the communication and protects the privacy and integrity of the data being exchanged between them. This would be sufficient to solve these issues because the Server and Clients could trust they are communicating with one another and that the messages are not being tampered with.

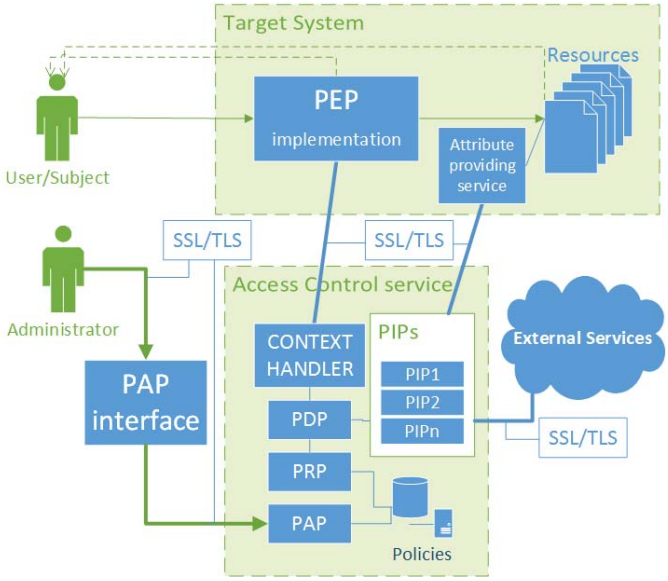

Figure 6. Architecture with marked SSL/TLS connections

Figure 6 shows the architect, distribution of components and has the SSL/TLS connections marked where they are required to be for a secure system.

Other options like OAuth 2 and OpenID Connect can be used on-top of TLS and provide additional benefits when considering connection with other systems but this work won't go into a detailed analysis of those options nor TLS as those technologies are already familiar and known solution for these types of problems. The additional benefits include delegation of the evaluation process and utilizing the tokens used by OAuth and Open ID Connect when connecting to other systems and , for example, fetching attribute data.

V. Proof of Concept

The use case scenario that the test was simulating was using the security component as an external service and communicating with it over a REST service. The use case is an IoT application called SMARTIE described in section II. The schema of the test scenario is equal to the one shown in Figure 5. but without connection in between the Access Control service and the resource (for fetching resource attributes) .This means that the PEP is integrated in the target solution and it communicates to the access control service over a REST service.

The requests that were sent vary in the complexity as some require all of the PIPS while others don't require any. Also, half of the requests result in a positive (*Permit*) result and half in a negative (*Deny*). The response time and an average was calculated. It also has to be noted that the test does not incorporate any type of caching so the repetition of the requests did not result in inaccurate results.

#	Result	Response time (ms)	#	Result	Response time (ms)
1	Permit	55	21	Deny	50
2	Permit	58	22	Deny	46
3	Permit	72	23	Deny	50
4	Permit	99	24	Deny	75
5	Permit	80	25	Deny	49
6	Permit	79	26	Deny	48
7	Permit	86	27	Deny	57
8	Permit	102	28	Deny	51
9	Permit	127	29	Deny	39
10	Permit	85	30	Deny	47
11	Permit	118	31	Deny	59
12	Permit	75	32	Deny	60
13	Permit	83	33	Deny	48
14	Permit	132	34	Deny	58
15	Permit	121	35	Deny	56
16	Permit	73	36	Deny	43
17	Permit	57	37	Deny	48
18	Permit	58	38	Deny	47
19	Permit	72	39	Deny	65
20	Permit	59	40	Deny	47
				Avrg.:	**68,35**

Table 1. Performance test results

These tests in Table 1 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.

VI. 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, but also for many others. 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. This, of course is not a trivial task and it requires precision.

A significant benefit of having this kind of system for enforcing security is that the initial requests made by the target system do not require to have many attributes, therefore they do not need to fetch all the information needed for evaluation They can rely on the access control service to fetch all additional attributes when and if needed in an efficient manner.

This work has shown that the architecture proposed in the standard [1] requires some modifications when implementing the standard. This is often the case as not all issues can be predicted in the planning stages. The architecture proposed in this work is an integration oriented proposal aimed to make XACML easier to use by other systems. Although the architecture is not a significant departure from the one defined in the standard it offers benefits as it defines the implementation scenario, solves distribution and connection issues.

VII. 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].

[2] Y. Keleta, "Proposing a Secure XACML architecture ensuring privacy and trust," *http://www.researchgate.net/profile/Hs_Venter/publication/228849158_Proposing_a_Secure_XACML_architecture_ensuring_privacy_and_trust/links/00463521dd0113e496000000.pdf*, 2005.

[3] W. D. N. K. Torsten Priebe, "Supporting Attribute-based Access Control with Ontologies," *Proceedings of the First International Conference on Availability, Reliability and Security (ARES'06)*, pp. 0-7695-2567-9, April 2006.

[4] P. R. R. F. E. B. F. I. a. J. L. Dan Lin, "A Similarity Measure for Comparing XACML Policies," *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 25, NO. 9,*, pp. 1946 - 1959, September 2013.

[5] F. C. J. H. a. T. X. Alex X. Liu, "Designing Fast and Scalable XACML Policy Evaluation Engines," *IEEE Transactions on Computers (Volume:60 , Issue: 12)*, pp. 1802-1817, December 2011.

[6] S. K. L. A. M. M. C. T. Kathi Fisler, "Verification and Change-Impact Analysis," *ICSE '05 Proceedings of the 27th international conference on Software engineering*, pp. 196-205, May 2005,.

[7] S. P. R. L. D. K. S. S. Markus Lorch, "First Experiences Using XACML for Access Control in Distributed Systems," *Proceeding XMLSEC '03 Proceedings of the 2003 ACM workshop on XML security*, pp. 25-37, 2003.

[8] "OASIS official wabsite," OASIS, 2015. [Online]. Available: https://www.oasis-open.org/org. [Accessed 20 March 2015].

[9] T. N. &. T. A. Pardo, "Conceptualizing Smart City with Dimensions of Technology, People, and Institutions," *The Proceedings of the 12th Annual International Conference on Digital Government Research*, pp. 282-291, June 2011.

[10] M. A. Taylor Sheltona, "The 'actually existing smart city'," *Cambridge Journal of Regions, Economy and Society, Volume 8, Issue 1*, p. 13–25, 2015.

[11] "SMARTIE Homepage," IHP GmbH, 22 July 2014. [Online]. Available: http://www.smartie-project.eu/project.html. [Accessed 20 February 2015.].