The XACML Standard
Addressing Architectural and Security Aspects

Óscar Mortágua Pereira, Vedran Semenski, Diogo Domingues Regateiro and Rui L. Aguiar
Instituto de Telecomunicações, DETI – University of Aveiro, Aveiro, Portugal

Keywords: XACML, ABAC, Access Control, Information Security, Software Architecture, IoT.

Abstract: The OASIS XACML (eXtensible Access Control Markup Language) standard defines a language for the definition of 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. When implementing real scenarios, developers can come across with the missing of several issues not addressed by the standard. For example, the architecture proposed defines the workflow but does not define the way components should be distributed over different machines. Additionally, the standard does not include any information about how securing communications between components. This paper proposes a solution to deal with the aforementioned gaps. A proof of concept is also presented in an IoT use case in the context of the European project: SMARTIE – secure and smarter cities data management.

1 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 very often used or integrated in more and more businesses (Keleta et al. 2005). These systems run on different technologies and different platforms. They can utilize many different workflows, methodologies, storage systems, etc.. 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. (Addie & Colman 2010) (Qing & Adams 2006). These issues are solved by developing and/or integrating security components and implementing security mechanisms.

Custom security components developed for solving security issues require significant effort to develop. They 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. Organisations can have many departments, use many services, databases, etc. Depending on how much the structure, architecture or date model changes or expands, issues can occur if the developed component is not flexible enough to deal with those changes.

The OASIS (Organization for the Advancement of Structured Information Standards) XACML (eXtensible Access Control Markup Language) 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 access control methodologies can also use XACML (Xu et al. 2011)(Stepien et al. 2011)(Ferrini & Bertino 2009). 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 in applications. Its main use is managing access to resources, which can be anything that the user defines (data, actions, services, etc.). 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

Pereira, Ó., Semenski, V., Regateiro, D. and Aguiar, R.
The XACML Standard - Addressing Architectural and Security Aspects.
DOI: 10.5220/0006224901890197
In *Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security (IoTBDS 2017)*, pages 189-197
ISBN: 978-989-758-245-5

great potential, flexibility and many advancements along with a uniformed solution, some aspects are not addressed. The issues that this paper addresses are ones that come from an implementation perspective, and not the XACML standard itself. Put more precisely, it will describe issues that were encountered while developing a security component based on the ABAC and the OASIS XACML standard, propose solutions for these issues and present a proof of concept. This paper deals mainly with internal and external communication, connection and architecture issues.

A security component with the proposed architecture was developed and tested in an IoT Smart City (European Project: SMARTIE – secure and smarter cities data management) (FP7 2016) use-case scenario. The security component uses a PDP evaluation engine and other basic XACML functionalities from an open source project (AT&T XACML 3.0 implementation).

The increasing need for integrating security components in systems is a reason to modify the existing architecture from a implementation perspective (Keleta et al. 2005) (Addie & Colman 2010) (Brown et al. 2012). It is because of this that the security component was viewed as a "black box" component that should be easy to integrate into other systems, easy to use and manage. This should therefore result in a more secure system and requires significant changes to the existing proposed solution for the XACML architecture (Brown et al. 2012).

This paper is organized in six additional main chapters. Chapter 2 presents the background technologies and terminologies that are related to this work. Chapter 3 presents the related work. Chapter 4 presents the issues that were found in the current proposal in the standard XACML (Brown et al. 2012). Chapter 5 describes the proposed solutions for the issues identified in Chapter 3. Chapter 6 presents a proof of concept and test results. Chapter 7 contains an overview of the work that was done and a final conclusion.

2 BACKGROUND

Before elaborating on the issues that were identified, a brief description of concepts relevant for this work needs to be given.

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 (Priebe et al. 2006)(Samarati & Di Vimercati 2001).

Before continuing, some key terms need to be explained as they will be used throughout this work: **Subject** - entity that is trying to access a certain resource. Example: person, process, device, etc. **Resource/Object** - anything that access control is being enforced upon. Example: database data, access to an application, service, access to sensors, etc. **Request** - the subject's request for a resource. It can be formatted in some way (document, file, string) and represent an actual "physical" request (database query, call to a service) or it can also be the actual "physical" request. **Policy** - set of rules that an access control based security system needs to enforce.

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 (Priebe et al. 2007)(Priebe et al. 2006). Attributes are typically divided into three categories: **subject** - subject/user attributes (examples: age, postal code, IP address, etc.); **object** - resource attributes (examples: type, value, age, etc.); **environment** (examples: day of the week, hour of the day, etc.).

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.

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) (OASIS 1993). It defines a way to evaluate requests for resources

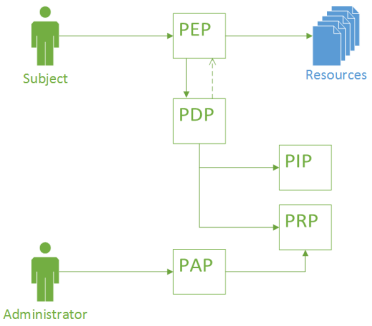

Figure 1: Reference XACML architecture.

according to rules defined in policies. Put simply it is a thought out and standardized solution for implementing access control in software applications (Lin et al. 2013)(Liu et al. 2011). It provides a common ground regarding terminology and workflow between multiple vendors building implementations of access control using XACML and interoperability between the implementations (Fisler et al. 2005)(Lorch et al. 2003). It is primarily intended for ABAC but can also be used for RBAC and others. 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 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. A**ccess requester** - entity that is requesting a resource. O**bligations service** - service that executes any obligations after the evaluation is complete. R**esource** - entity containing one or more

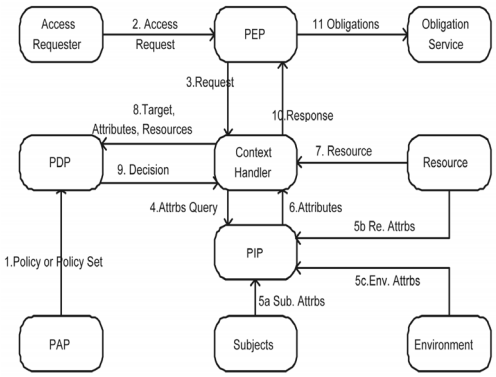

Figure 2: Data workflow proposed by XACML standard.

resources and resource attributes that the access requester is trying to access. S**ubjects** - entity containing subject attributes. Typically the subject attributes are attributes of the access requester. E**nvironment** - 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.

3 RELATED WORK

The architecture proposed in the OASIS XACML has been a basis for many modifications as many implementations have different requirements. As the architecture is somewhat openly defined and leaves many aspect unaddressed, some issues have already been addressed. Many implementations and proposals presented in (Kehlenbeck et al. 2010)(Sardinha et al. 2007)(Brown et al. 2012) demonstrate that there are many possibilities and areas of implementation with XACML but also that the architecture and data flow are often modified to fit specific needs.

The work done by Y. Keleta in (Keleta et al. 2005) has addressed some security issues with the data flow proposed in the standard. The connections between components were recognized as one of the aspects where security mechanisms were not defined. This of course leaves the connection open to various attacks if a malicious party gains access to that connection. The solution that was proposed was based on having a central entity that would distribute a token over SSL to other components and generate a security key for encrypting the data. Although this

work also uses SSL/TLS, other aspects like the central entity, tokens and security keys are not needed, as explained in Chapter 5.

In (Xu & Duminda 2009) concurrency issues between the evaluation and the administration parts were identified. A lock manager is proposed that would give permission to access policies by locking them with write-locks or read-locks.

In (Díaz-López et al. 2015) a proposal for managing XACML systems in a distributed environments and connection between central entities and subsidiaries in a distributed system is presented. It proposes a solution that incorporates SSL connections and message encryption similar to work done in (Keleta et al. 2005).

On the other hand, other related work focuses more on expanding the standard (Ferrini & Bertino 2009)(Ardagna et al. 2009)(Demchenko et al. 2009)(Kabbani et al. 2014), providing it even with more functionality and flexibility. They do this by integrating it with other methodologies and other systems. In (Kabbani et al. 2014) a Situation-Oriented Authorization Architecture is presented that combines a situation management architecture and the OASIS XACML architecture for the purpose of Specification and Enforcement of Dynamic Authorization Policies. These works demonstrate that the development of the XACML standard is not finalized and is likely to continue evolving.

4 IDENTIFIED ISSUES

While developing a security component based on the OASIS XACML standard, a number of issues were identified. These issues were related to the architecture proposed in the standard and security of connections between components and external services.

Removal of PRP By comparing the reference XACML architecture to the one proposed in the OASIS XACML standard v3.0 (OASIS 2013) it can be seen that in addition to new components, the PRP has been merged with the PAP. Put differently, the functionality of the PRP has been added to the PAP and it is now 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 an 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 questions come up, such as 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.

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 means 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. By removing that connection, the role of the Context Handler (from an implementation perspective) becomes a trivial "middle man" in between the PDP's communication with the PIP and the PEP. The role that the Context Handler can still assume is the initialisation/manager role, handling all other aspects that the other components are not 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 an 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 do not 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 hereafter indicated: **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 finds 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 (OASIS 2013). Without enforcing some security measures this leaves the system vulnerable to attacks and may jeopardize the confidentiality of the access requests and the authorization decisions. It is important to put appropriate safeguards in place to protect the system from such attacks. Examples of such attacks include (Keleta et al. 2005): 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 (Keleta et al. 2005). The standard does not 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 a 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, control who gets access to the resources, monitor the traffic, gain insight into what is happening and collect 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. Connections need to be limited only to other components that need to access the policies (PRP, PAP).

5 PROPOSED SOLUTION

After identifying the issues not addressed in the OASIS XACML architecture, a new architecture was made. Tests of a security component implementation were done and are presented in Chapter 6. This Chapter will present the proposed architecture as well as solutions for connection issues. 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

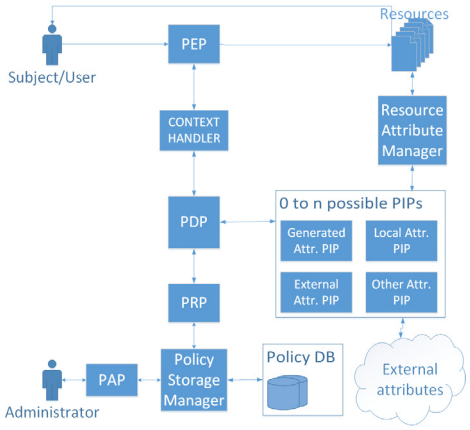

Figure 3: New Proposed architecture.

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 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. 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

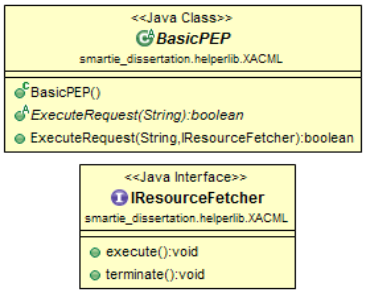

Figure 4: Class diagram of the PEP and additional interface.

the evaluation result is also available.

Solving the distribution and securing connections

The components that should be grouped together, i.e. be deployed together, are: PDP, Context Handler, PRP and PIPs, as shown in Figure 5. 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 is not 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 than 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 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 (Díaz-López et al. 2015). For this to work with the REST service an additional component would be needed. It would have functionality for handling

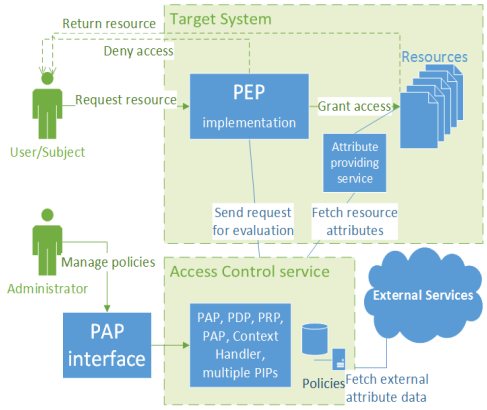

Figure 5: Distribution of components in a use-case.

multiple instances and delegating the workload efficiently. 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 (Keleta et al. 2005) and explained more in Chapter 3. The proposed 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 the components are grouped together this is unnecessary, not to mention that encrypting these tokens can add unwanted overhead.

As mentioned, 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) (Díaz-López et al. 2015) (Keleta et al. 2005). Using this method provides the authentication to both parties

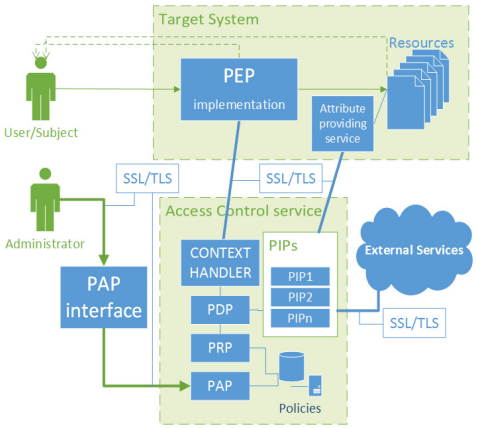

Figure 6: Architecture with marked SSL/TLS connections.

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 shows the architecture, 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 will not go into a detailed analysis of those options or of 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.

6 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 in the context of the European project: SMARTIE – Secure and smarter cities data management.

As stated in (Nam & Pardo 2011), 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 (Shelton et al. 2015).

SMARTIE (Smart City) is a European project with the goal of solving security, privacy and trust issues in IoT, with a Smart City implementation. SMARTIE is still in the development stages and was used as a use case scenario for testing an implementation of the architecture proposed in Chapter 5. The security component which was tested was built using an AT&T XACML implementation (XACML 2013) open source project. The PDP engine was used for evaluating requests and policies and custom PIPs were implemented from basic PIP interfaces to communicate with the PDP. All other components (PRP, PAP, Context Handler, Policy Storage Manager, etc.) were developed and organizes in an architecture shown in Figure 3. The schema of the test scenario is equal to the one shown in Figure 3 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 and the PIPs fetch additional attributes both from internal and external sources. The requests that were sent vary in the complexity as some require all of the PIPs while others do not require any. Additionally, half of the requests result in a positive (*P-Permit*) result and half in a negative (*D-Deny*). The response time and the average were 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. The purpose of this test is to verify that the developed solution gives results as predicted and that the evaluation process is working as intended.
These tests in Table 1showed 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 shows, the solution performed as predicted using requests and policies from the target system.

Table 1: Test results (R-Result, P-Permit, D-Deny).

#	R	(ms)	#	R	(ms)	#	R	(ms)	#	R	(ms)
1	P	55	11	P	118	21	D	50	31	D	59
2	P	58	12	P	75	22	D	46	32	D	60
3	P	72	13	P	83	23	D	50	33	D	48
4	P	99	14	P	132	24	D	75	34	D	58
5	P	80	15	P	121	25	D	49	35	D	56
6	P	79	16	P	73	26	D	48	36	D	43
7	P	86	17	P	57	27	D	57	37	D	48
8	P	102	18	P	58	28	D	51	38	D	47
9	P	127	19	P	72	29	D	39	39	D	65
10	P	85	20	P	59	30	D	47	40	D	47
					Avg:						68.4

7 CONCLUSION

The ABAC model together with the XACML standard has great potential and offers great benefits. 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. 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.

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.

This work has shown that the architecture proposed in the standard (OASIS 2013) does not cover all aspects that need to be considered when deploying such a solution, and implementations require some extensions to keep it secure. 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 and solves distribution and connection issues that are sure to arise when deploying such a system.

ACKNOWLEDGEMENTS

This work is funded by National Funds through FCT Fundação para a Ciência e a Tecnologia under the project UID/EEA/50008/2013.

REFERENCES

Addie, R.G. & Colman, A., 2010. Five Criteria for Web-Services Security Architecture. In 4th International Conference on Network and System Security (NSS),. pp. 521–526.

Ardagna, C.A. et al., 2009. An XACML-based privacy-centered access control system. In Proceedings of the first ACM workshop on Information security governance - WISG '09. p. 49.

Brown, K.P. et al., 2012. Fine-grained filtering of data providing Web Services with XACML. In Proceedings of the Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE. pp. 438–443.

Demchenko, Y., Cristea, M. & De Laat, C., 2009. XACML policy profile for multidomain network resource provisioning and supporting authorisation infrastructure. In Proceedings - 2009 IEEE International Symposium on Policies for Distributed Systems and Networks, POLICY 2009. pp. 98–101.

Díaz-López, D. et al., 2015. Managing XACML systems in distributed environments through Meta-Policies. Computers and Security, 48, pp.92–115.

Ferrini, R. & Bertino, E., 2009. Supporting RBAC with XACML+OWL. In Proceedings of the 14th ACM symposium on Access control models and technologies SE - SACMAT '09. pp. 145–154. Available at: citeulike-article-id:9252058%5Cnhttp://dx.doi.org/10.1145/1542207.1542231.

Fisler, K. et al., 2005. Verification and Change-Impact Analysis of Access-Control Policies. Proceedings of the 27th International Conference on Software Engineering, pp.196–205.

FP7, 2016. SMARTIE - secure and smarter cities data management. Available at: http://www.smartie-project.eu/ [Accessed October 25, 2016].

Kabbani, B. et al., 2014. Specification and enforcement of dynamic authorization policies oriented by situations. In 2014 6th International Conference on New Technologies, Mobility and Security - Proceedings of NTMS 2014 Conference and Workshops.

Kehlenbeck, M., Sandner, T. & Breitner, M.H., 2010. Managing internal control in changing organizations through business process intelligence - A service oriented architecture for the XACML based monitoring of supporting systems. In Proceedings of the Annual Hawaii International Conference on System Sciences.

Keleta, Y., Eloff, J.H. & Venter, H., 2005. Proposing a Secure XACML architecture ensuring privacy and trust, Available at: http://icsa.cs.up.ac.za/issa/2005/Proceedings/Research/093_Article.pdf.

Lin, D. et al., 2013. A similarity measure for comparing XACML policies. IEEE Transactions on Knowledge and Data Engineering, 25(9), pp.1946–1959.

Liu, A.X. et al., 2011. Designing fast and scalable XACML policy evaluation engines. IEEE Transactions on Computers, 60(12), pp.1802–1817.

Lorch, M. et al., 2003. First experiences using XACML for access control in distributed systems. In Proceedings of the 2003 ACM workshop on XML security. pp. 25–37.

Nam, T. & Pardo, T. a., 2011. Conceptualizing smart city with dimensions of technology, people, and institutions. Proceedings of the 12th Annual International Digital Government Research Conference on Digital Government Innovation in Challenging Times - dg.o '11, p.282. Available at: http://dl.acm.org/citation.cfm?id=2037556.2037602%5Cnhttp://dl.acm.org/citation.cfm?id=2072069.2072100%5Cnhttp://dl.acm.org/citation.cfm?doid=2037556.2037602.

OASIS, 2013. eXtensible Access Control Markup Language (XACML) Version 3.0. Available at: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf [Accessed October 25, 2016].

OASIS, 1993. OASIS. Available at: https://www.oasis-open.org/org [Accessed October 23, 2016].

Priebe, T. et al., 2007. Supporting attribute-based access control in authorization and authentication infrastructures with ontologies. Journal of Software, 2(1), pp.27–38.

Priebe, T., Dobmeier, W. & Kamprath, N., 2006. Supporting attribute-based access control with ontologies. In Proceedings - First International Conference on Availability, Reliability and Security, ARES 2006. pp. 465–472.

Qing, X. & Adams, C., 2006. XACML-Based Policy-Driven Access Control for Mobile Environments. In Canadian Conference on Electrical and Computer Engineering. pp. 643–646.

Samarati, P. & Di Vimercati, S.D.C., 2001. Access Control: Policies, Models, and Mechanisms. Foundations of Security Analysis and Design, 2171, pp.137–196. Available at: http://www.springerlink.com/index/80wrewj7j1a716wb.pdf.

Sardinha, A., Rao, J. & Sadeh, N., 2007. Enforcing context-sensitive policies in collaborative business environments. In Proceedings - International Conference on Data Engineering. pp. 705–714.

Shelton, T., Zook, M. & Wiig, A., 2015. The "actually existing smart city." Cambridge Journal of Regions, Economy and Society, 8, pp.13–25. Available at: http://cjres.oxfordjournals.org/lookup/doi/10.1093/cjres/rsu026.

Stepien, B., Matwin, S. & Felty, A.P., 2011. Advantages of a non-technical {XACML} notation in role-based models. In Ninth Annual Conference on Privacy, Security and Trust. pp. 193--200.

XACML, 2013. AT&T XACML 3.0 Implementation. Available at: https://github.com/att/XACML [Accessed October 23, 2016].

Xu, M. & Duminda, W., 2009. A role-based XACML administration and delegation profile and its enforcement architecture. In ACM workshop on Secure web services. pp. 53–60.

Xu, M., Wijesekera, D. & Zhang, X., 2011. Runtime administration of an RBAC profile for XACML. IEEE Transactions on Services Computing, 4(4), pp.286–299.