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

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.

# Background

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.



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

Figure 4 shows the architecture proposed in the OASIS XACML standard. The components are as follows:

* **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 are split into three types: subject, environment and resource attributes;
* **PAP** (Policy Administration Point) - the policy administration point contains the functionality required for managing policies. Typically this means adding and removing policies;
* **Context** **Handler** - this entity controls the workflow of the system. It communicates with the PEP, PDP, PIP and resource;
* **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.

One other component that should be mentioned as it was part of the architecture in version 1.0 of the standard and will be relevant for the work described later in this dissertation is the PRP.

* **PRP** (Policy Retrieval Point) - component used for retrieving of policies. In version 1.0 it is used to provide the PDP with policies unlike in version 3.0 as shown in Figure 4 which uses the PAP for that purpose

An issue with removing the PRP adn integrating it's functionality in the PAP is that the context handler has access has access to PAP's functionality, meaning it can potentially modify, remove or change policies.

The removal of a PRP from the new architecture and having that functionality integrated in to the PAP could bring issues. Because of this, the final solution has an architecture that is different and will be addressed in section 4.

One significant beneficial characteristic that is a result of this architecture and workflow is that the initial request that originates from the access requester doesn't need to contain all of the attributes needed for evaluation. It therefore doesn't need to be aware of possible services and other entities that need to be contacted by the PIP for getting additional attributes. This also means that there is no need to rely on the entity requesting access to include all required attributes correctly. From an implementation perspective this should result in a system that is easier to integrate. Also, this means that it is more secure as the PIP is an internal component and can be trusted more than an entity requesting access.

# Issues

While developing this solution, a number of problems emerged. The architecture was therefore modified. The main issue was the updating of the resource attributes, storing them on a separate system, the security issues regarding that connection and the synchronisation of data between the systems. Another issue is the connection between the context handler and the PRP. As a request is received the Context Handler has to get the policies and deliver them to the PDP. As it is just a "middle man" in this case, it is better for it to be connected directly to the PDP. Another issue is connection between the context handler and the PIP. The Context Handler could in this case could work in three ways:

1. Go through a list of policies, see if there are any attributes that are missing and fetch them from the PIP and deliver the modified request to the PDP for evaluation;
2. Fill the request "blindly" with every attribute that it can get with the attributes already contained in the request and deliver the modified request to the PDP for evaluation;
3. Wait for the PDP to ask for the attributes while it is evaluating, fetch the attributes from the PIP and deliver them to the PDP.

The 1st possibility is somewhat inefficient. The Context Handler has to have functionality for searching and parsing policies and requests, and while doing that it will go through all the policies and all fill the request with everything that it would need for all of the policies. The request could potentially become too large and it would require significant time to parse trough everything and fetch, thus degrading performance. Also, this functionality is already built into the PDP as is does that while evaluating the requests.

The 2nd possibility is requires a simpler functionality as it doesn't require it to parse trough all of the requests but it fills every request with all possible attributes all the time. This will later be used by the PDP and it will slow down the evaluation process. Additionally, if there is a PIP that has to fetch attributes from another outside source through a service and the process of fetching the attributes takes significant time, that additional time will be added to the evaluation time. This will happen for every request evaluation therefore diminishing performance.

The 3rd way is equal to the one defined in the standard. This method is efficient as the PDP will ask for attributes only when it needs to and the Context Handler will deliver them to it. The problem in this scenario is that the Context Handler is still the "middle man" and just forwards requests and responses without having any additional functionality.

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].

ISSUES

SECURITY

SCALABILITY

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. ...

# 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.

# 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]. |