Dealing with Stable Environmental Conditions in XACML Systems

Romain Laborde, Thierry Desprats
IRIT UMR 5505, Université Paul Sabatier
118 route de Narbonne, 31062 Toulouse cedex 9, France
{laborde, desprats}@irit.fr

Abstract

XACML (eXtensible Access Control Markup Language) is an XML-based language for access control that has been standardized in OASIS. In this language, any entities involved in access control (i.e. users, resources, actions and environment) are specified by a set of attributes. This specification also includes the description of an architecture that explains how the policy decision point (PDP) retrieves the needed attributes values when it evaluates the policy to take its authorization decision.

In this paper, we show this approach for getting the attributes values is a bottleneck to the performance of the authorization decision-making process for attributes whose process for retrieving the value is long and the changing of its value doesn’t impact the policy frequently. Thus, we propose an improvement of the XACML architecture in order to accelerate the decision-making process when PDP has to treat such kind of attributes.

1. Introduction

Today, resources and services are complex and heterogeneous. Users are changeable and members of multiple organizations. To fit these requirements, access control models and access control solutions have followed the evolution of the systems environment to protect. Traditional Identity Based Access Control (IBAC) models [1], which are closed and inflexible, have been replaced by new models like Role-based Access Control Models (RBAC) [2] and more recently Attribute Based Access Control models (ABAC) [3].

Unlike IBAC and RBAC, the ABAC model can define permissions based on just about any security relevant characteristics of requestors, actions, resources, and environment, known as attributes.

This approach makes ABAC scalable and flexible and thus more suitable for distributed, open systems that are identity-less like the Internet where subjects are identified by their characteristics, such as those substantiated by certificates.

XACML (eXtensible Access Control Markup Language) is an XML-based language for access control that has been standardized in OASIS [4]. XACML policies are ABAC-Based. XACML specification also includes the description of an architecture that explains attributes are retrieved using a query/response approach during the decision-making process.

This synchronous method is a bottleneck to the performance of the authorization decision-making process for attributes whose process for retrieving the value is long and the changing of its value doesn’t impact the policy evaluation frequently. We name stable conditions the policy conditions whose results change rarely.

Thus, the XACML architecture should be improved in order to accelerate the decision-making process when it evaluates these attributes. We build our approach on 1) removing the stable conditions from the evaluated policy and 2) adding the ability to adopt an event driven behaviour to XACML. The idea is to benefit from asynchronous notification mechanisms to inform the decision-making process the value of a stable condition attribute has changed in such a way the value of the associated stable condition has also changed.

The rest of the paper is organized as follow. Section 2 introduces the XACML policy language and its architecture. Section 3 presents a scenario that points out the performance issue. Section 4 provides a generic definition of stable conditions. Section 5 and 6 present an enhancement of the XACML architecture to deal with stable conditions and its implementation for the scenario. Finally, section 7 describes the related works and section 8 concludes the paper.
2. Presentation of XACML 2.0

XACML (eXtensible Access Control Markup Language) is an XML-based language for access control that has been standardized in OASIS. XACML describes both an attribute-Based access control policy language and a request/response language. XACML policy language is used to describe general access control requirements in term of constraints on attributes. Specifically, attributes could be any characteristics of the subject, resource, action, or environment in which the access request is made. Considering attributes allows the language to be very flexible. Moreover, XACML language is extensible. It has standard extension points for defining new functions, data types, combining logic, etc.

In addition, XACML provides a management architecture that describes the different entities and their roles related to the decision-making-process. This architecture is described by the data-flow model. A simplified version of this model is depicted in Figure 1.

![Figure 1. Simplified XACML data flow model](image)

The model operates by the following steps.
1. Policy Administration Points (PAP) write policies and policy sets and make them available to the Policy Decision Point (PDP). These policies or policy sets represent the complete policy for a specified target.
2. The access requestor sends a request for access to the Policy Enforcement Point (PEP).
3. The PEP sends the request for access to the context handler in its native request format, optionally including attributes of the subjects, resource, action and environment.
4. The context handler constructs a standard XACML request context and sends it to the PDP.
5. The PDP requests any additional subject, resource, action and environment attributes from the context handler.
6. The context handler requests the attributes from an Attribute Information Point (PIP).
7. The PIP obtains the requested attributes.
8. The PIP returns the requested attributes to the context handler.
9. The context handler sends the requested attributes. The PDP evaluates the policy.
10. The PDP returns the standard XACML response context (including the authorization decision) to the context handler.
11. The context handler translates the response context to the native response format of the PEP. The context handler returns the response to the PEP that enforces the authorization decision.

The way the PDP gets the required attributes values when it evaluates the policy can constitute a bottleneck to the performance of the decision-making-process. We introduce an example to point out the issues.

3. Example of a scenario

In this example, we consider a FTP server protected by an XACML-Based access control solution as illustrated in Figure 2.

![Figure 2. Architecture of the example](image)

The FTP server holds two directories: private and public. The private directory contains confidential files about the organization whereas files that are accessible to everybody are stored in the public directory. The policy states that users with role “corporate” can access to any files in the private directory. Files in the public directory can be accessed by anybody unless the network is too busy, which means the bandwidth use rate on the edge router is greater than 60%. This last constraint protects the network from the congestion in order to ensure people from the organization to be able to access the any files in the private directory at any time. The XACML specification is defined in Figure 3.
The edge router hosts an SNMP agent that provides the bandwidth rate. In order to simplify our example, we consider that there is a management object that carries this value. Actually, this is not true in most cases and the standard Management Information Bases (MIB) imply the SNMP manager to send several requests to calculate this value [5].

Considering the XACML dataflow model and mainly steps 5 to 10, every time a user accesses any file in the public directory, the context handler needs to ask the PIP the value of the bandwidth use rate. The PIP, which also acts the role of SNMP manager, gets this value by sending a message SNMP GET-REQUEST. When the SNMP agent receives the request, it looks in its MIBs and returns the value through a message SNMP GET-RESPONSE. Then, the PIP can reply to the context handler that can provide the attribute to the PDP. Now the PDP can evaluate the policy.

The main issue is this process consumes both time and network bandwidth. In addition, the network is busy being an exceptional event emphasizes the uselessness of systematically evaluating the bandwidth use rate.

The next section analyzes the characteristics of such kind of attributes.

4. Stable conditions

As the example above shows, it is not necessary to evaluate if the bandwidth use rate is greater than 60% because this state is exceptional. We name stable conditions this kind of constraints.

A stable condition is an expression that always returns the same result during a given period considered to be long.

In our example, the condition “the bandwidth use rate is less than 60%” is mostly always true. In the same way, any event generated by an intrusion detection system should be a stable condition. Another example, which is more common, could be “the current time should be between 7:00 and 19:00”. This example is less demonstrative because the requesting process is faster. This comes from the current time value is given by a process running on the same machine as the XACML access control solution. But, like the bandwidth example, this expression returns always true from 7:00 until 19:00 and always false between 19:00 and 7:00.

How to identify a stable condition? It is difficult to determine if a condition is stable or not. Indeed, stability does not only depend on the attributes. In our previous example the value of the bandwidth rate changes all the time. Here, the stability property is due to the variance of the attribute, the operator “less than” and the value of the constant “60%”. Thus, stability can be identified only by experience of administrators or analysis of the past evaluations of the condition. We call stable condition attributes, the attributes that are parameters of stable conditions.

5. Proposed architecture

The evaluation of stable conditions being useless, we propose to enhance the XACML framework in such a way stable conditions are considered only when it’s necessary. In addition, we want enhanced XACML systems to be still compliant with the original XACML specification.
Our idea is:
1) remove stable conditions from the policy evaluated by the PDP,
2) and modify this policy according the changes of stable conditions values.

Let $R$ be a policy rule that includes a stable condition $C$. Let $R$-True (resp. $R$-False) be rule $R$ without $C$ when $C$ returns true (resp. false). For example, rule PublicAccess in Figure 3 will be divided into rules PublicAccess-True (Figure 4a) when the network is operational and PublicAccess-False (Figure 4b) when it is congested.

In order to make the PDP switches $R$-True to $R$-False (and vice-versa), we add XACML the ability to adopt an event driven behavior. The idea is to benefit from asynchronous notification mechanisms to deal with management of stable conditions.

In reaction to this event, the PIP will reflect the stable condition changing within the XACML framework by alerting the PAP, which is the entity responsible of the policy administration. The PAP should be able to dynamically modify of the rule(s) ($R$-True into $R$-False or vice-versa) concerned by this changing. The PAP stores both $R$-True and $R$-False. It can also read a configuration file that indicates what rule(s) should be changed for each notification message.

This new architecture allows the PDP to use either $R$-True or either $R$-False during policy evaluation without neither having to consider $C$ nor having – a fortiori - to retrieve the values of the stable condition attributes in $C$ as it should have achieved in the case of $R$ evaluation. The decision to use either $R$-True or $R$-False is dynamically driven by the significant changes of the stable condition attribute values.

In Figure 6 we show the impacts of our proposition on the XACML standard framework are limited. Classical entities behaviors and interactions between them are not modified. The behavior of both PAP and PIP has been enhanced without disabling their standard activities. The OASIS standard roles of management entities are conserved. The PIP is still acting as an information collector, and the PAP as the policy administration point.

Initially, the PAP sets the PDP with a policy that includes $R$-True or $R$-False depending on what the value is when the status of the system is operational. The PIP has subscribed to a trusted Environment Attribute Provider (EAP) its interest to be notified when $C$ is changing. The PIP is listening for relevant
notification messages issued from the EAP. When such a notification is received, the PIP builds a specific XACML-based message mentioning the C condition and its current value. This message is then sent to the PAP. This latter evaluates the value of the condition. If the value is false, it changes the current policy by using the R-False rule. When it is true, R-True is loaded. Note that during this process, no synchronous interaction is achieved between the XACML framework and the EAP concerned by the stable condition attributes. The evaluation process concerning the non stable condition attributes is achieved in the standard way.

6. Example of implementation

We present an example of implementation (Figure 7) we are developing as a proof of concept. It is limited to the scenario described in section 3. We use NET-SNMP [6] to handle asynchronous SNMP trap messages and the Sun’s XACML implementation [7].

![Figure 7. Example of implementation](image)

The SNMP agent, which runs on the edge router, sends SNMPv3 traps to the daemon snmptrapd [6] on the FTP server. In respect with our definition of a trusted EAP, we have chosen to use SNMPv3 traps because this version provides authentication, integrity, confidentiality and anti-replay mechanisms [8]. The SNMP agent sends alternatively traps that indicate the bandwidth use rate is greater than 60% (OID: .iso.org.dod.internet.experimental.siera.bw.greater60) and traps that indicate the bandwidth use rate is less than 60% (OID: .iso.org.dod.internet.experimental.siera.bw.less60). When traps are received by snmptrapd, they are validated (integrity and confidentiality) and forwarded to the notification aware PIP. Then, the PIP translates OID of the stable condition changing to its associated standardized notification message. The PIP informs the PAP of the changing that is identified by the URN. Finally, the PAP modifies the public access rule. When the received notification is <NotificationMessage StableConditionID=urn:irit:siera:notification:name:bwlessthan60 Value="false"/>, the new rule denies the access to the public directory (Figure 4b). When the notification is <NotificationMessage StableConditionID=urn:irit:siera:notification:name:bwlessthan60 Value="true"/>, the access is granted (Figure 4a).

7. Related works

Privileges Management Infrastructure (PMI) solutions (e.g. PERMIS [9] or Akenti [10]) propose two ways for retrieving attributes: the push and pull models. Their architecture is slightly different from XACML as there is no context handler and no PIP explicitly mentioned. Thus, the PEP talks directly to the PDP. In the push model, the PEP provides the PDP all the attributes. For example, the PEP sends an attribute certificates with the Distinguished Name (DN) of the user and its roles. In the pull model, the PEP provides some of attributes. The PDP completes the list of the required attributes. In this case, the PEP only sends the DN of the user and the PDP gets the attributes certificate in a LDAP repository based on the user DN. Environment attributes are limited to the time or the date.

Important research works have been carried out to deal with specific requirements in context-sensitive environments like mobile and pervasive computing, ambient intelligent systems or smart spaces. They focus on proposing “context-aware” architectures and/or frameworks. It means the context leads the behaviour of the system.

Inspired from the PMI push model, Al Muthadi et al. [11] have proposed a “Context Provider” that can get contextual information from either sensors or others data sources. The “Inference Engine”, which is the core of their context-aware security systems, can either query a Context Provider for context values or ask it to be notified when a condition changes. However, they do not explain when/why request/query or notification messages should be used. No mechanism improves the decision-making-process.

The same approach is adopted in [12] where a “Dynamic Context Service” is a trusted entity used for acquiring context information either directly or via a

---

5 OID stands for Object IDentifier
third party. A policy governs the frequency at which the context is acquired and updated and specifies some thresholds for notifying the core system. However, when a condition change, all the policy is re-evaluated that is a bottleneck for high speed environments.

Covington et al. have proposed in their initial works a PMI pull-like model named Generalized RBAC [13]. This model adds environment roles that can be activated or not. They use caching techniques and add some mechanisms to estimate the freshness of their values. The approach improves the performance of the decision-making-process because deactivated environment roles are not considered during the evaluation process. But, caching techniques cannot be used in our context, as stable condition changing notifications should be considered immediately. More recently, Convington and Sastry have proposed the Contextual ABAC (CABAC) model [14] that is the update of GRBAC principles to the ABAC model. In the same way, environment attributes could be activated/deactivated. They just propose a conceptual access control model that doesn’t deal with implementation issues. Despite, they point out the fact that some of environment attributes need to be evaluated every time while others are subject to non changing conditions during a entire session, no clear ratiocination has been developed on this subject.

8. Conclusion

We have shown in this article that all the attributes should not be retrieved in the same manner. We have defined what stable conditions are and why they should not be treated by the XACML systems like usual conditions. We have provided an architecture to improve the performance of the decision-making-process when it deals with stable conditions. This architecture is compliant with existing XACML specification. Finally, we have presented an implementation that uses the Sun’s XACML implementation and net-snmp.

However, our solution has two drawbacks: the administrator should know which of the conditions are stable and (s)he needs to configure the EAP, the PIP and PAP. As a consequence, it is more difficult to specify a policy now.

Our future work will focus on automating this process. The administrator will specify a security policy without taking into account the existence of stable conditions. The decision-making-process will monitor the results of each rule condition to provide information for automatically detecting stable conditions. When a stable condition is found, an optimization process will configure the EAP, the PIP and the PAP to use asynchronous notification mechanisms. This works aim to endow an XACML framework with a self-optimizing behaviour, which is one of the Autonomic computing properties defined in [15].

9. References