An Advanced Policy Based Authorisation Infrastructure

David Chadwick
University of Kent
School of Computing
Canterbury
+44 1227 823 221
d.w.chadwick@kent.ac.uk

Kaniz Fatema
University of Kent
School of Computing
Canterbury
+44 1227 823 823
kf66@kent.ac.uk

ABSTRACT
We describe a more advanced authorisation infrastructure for identity management systems which in addition to the traditional Policy Enforcement Point (PEP) and Policy Decision Point (PDP) has an application independent policy enforcement point (AIPEP), a credential validation service (CVS) and a master PDP. The AIPEP is responsible for handling sticky policies, calling the master PDP, performing application independent obligations, and validating credentials using the CVS. The master PDP is responsible for calling multiple traditional PDPs that support a variety of policy languages, and resolving conflicts between the various authorisation decisions. Whilst this authorisation infrastructure may seem more complex to implement, it is in fact easier for applications to integrate since nearly all of the complexity is hidden beneath the PEP interface.

Categories and Subject Descriptors
D.4.6. Security and Protection, Access Controls

General Terms
Design, Security

Keywords
Credential Validation Service, Master PDP, Application Independent PEP, Sticky Policy, PDP, PEP, Obligations Service

1. INTRODUCTION
Policy based access control systems are now well established. They rely on an application independent policy decision point (PDP) to make authorization decisions, and an application dependent policy enforcement point (PEP) to enforce these decisions. In a federated system we cannot assume that every service provider (SP) and every user will use the same policy language for specifying their rules Hence the same PDP cannot be used for evaluating all policies. This is because different policy languages support different rule sets and functionality; so it is not possible to construct every type of required policy using just one policy language. Today we have many examples of different policy languages e.g. XACMLv2 [1], XACML v3 [2], PERMIS [3], P3P [4], Keynote [5] etc. and hence many different PDP implementations. If a user provides a sticky consent policy in a different language to that used by the SP's access control policy, the SP will need an authorization infrastructure that is capable of evaluating multiple policies written in multiple languages.

Obligations are actions that must be performed when a certain event occurs. When the event is an authorization decision, then the obligations are actions that must be performed either before, after or along with the enforcement of the authorization decision [8]. Many obligations will be application specific, but some may be application independent, for example, recording the authorization decision in a secure audit trail, or notifying the user that someone has been granted access to his personal data. Other obligations may be introduced to make up for deficiencies in the PDP's policy language, e.g. [7] describes how obligations can be used to specify over-ride policies in the XACML language, whilst [8] says how obligations can be used to support state based decision making in stateless PDPs. Given that some obligations are naturally application independent, whilst others are extensions of the functionality of the PDP, then it would be beneficial to have these obligations enacted by an application independent component of the authorization infrastructure, thereby reducing the burden on the application developer.

PDPs need to obtain their policies from somewhere. The XACMLv2 standard proposes a functional component called the Policy Administration Point (PAP) which is responsible for creating the policies and making them available to the PDP through some back channel prior to the PDP making its decisions. The back channel could be, for example, an API to an integrated database, or a communications link to an external repository. However, using a back channel and previously prepared policies is too static for some use cases. Consider the privacy protection of personal data, where a user's privacy policy is stuck to her personal identifying information (PII) [18]. In this case the policy needs to be passed dynamically along with the decision request to the PDP.

In attribute based access controls (ABAC), the PDP makes it decisions based on the attributes of the subject, requested action, resource object and environment. In the XACML model the attributes are provided by a Policy Information Point (PIP). Whilst a PEP can usually reliably obtain the attributes of the resource object and the user's requested action, and in some cases those of the environment, validating the attributes of the subject (and in some cases those of the environment) requires considerably more effort. This suggests that there are different types of PIP. A subject's attributes are most often transferred as credentials, digitally signed by the authoritative source(s) of the attributes. We thus need a type of PIP that is responsible for validating credentials, extracting the valid attributes from them

DIM 2009, November 13, 2009, Chicago, Ohio, USA.

and discarding the rest. We propose a credential validation service (CVS) for this [9].

Finally, in an identity management system that uses sticky policies, we need a way of securely transferring these policies between system components along with the data they are "stuck" to. Whilst we could leave this to each application to do in its own protocol specific way, we could also do this in an application protocol independent way thereby making it easier for applications to incorporate sticky policies into their systems [19]. We propose solutions for each of the above problems in our advanced authorization infrastructure.

The rest of this paper is structured as follows. Section 2 describes our advanced authorization infrastructure. Section 3 concludes and describes our future plans.

2. AN ADVANCED AUTHORISATION INFRASTRUCTURE

2.1 The Application Independent PEP

We introduce several new components into existing authorization infrastructures. Firstly we introduce an application independent policy enforcement point, the AIPEP. The AIPEP is responsible for coordinating the various actions of the application independent authorization infrastructure. It presents a single interface to the application dependent PEP, in order to make integration easy. To the PEP it appears to be a standard PDP. To the (Master) PDP, it appears to be a normal PEP. It is therefore a proxy for both. The interface that we are currently using for the PEP-AIPEP is the recently published OASIS draft standard [10]. This SAML profile allows an XACML formatted authorization decision request to be combined with the policy that is to be used by the PDP and both passed as a SAML query in a new element called an XACMLAuthzDecisionQuery. The SAML response allows an XACML response context, containing a response and optional obligations, to be returned in a newly defined SAML assertion, the XACMLAuthzDecisionStatementType. Other optional parameters can also be in the assertion, such as an altered request context which contains the set of attributes that were actually used in the authorization decision making. When the AIPEP receives the authorization decision query message (step 1 in figure 1), it first calls the CVS to validate any credentials that are contained in the query message (step 2 in figure 1). We have specified how various types of unvalidated credentials such as SAML attribute assertions, X.509 public key certificates and X.509 attribute certificates can be passed to the AIPEP/PDP proxy in an Open Grid Forum profile [11] of [10].

2.2 The Credential Validation Service

The CVS is described in detail in [9]. It is a specialized PIP that is configured with a credential validation policy which tells it which credentials are valid, in terms of who the trusted attribute authorities (AAs) are and which attributes each are trusted to issue to which groups of users. The protocol we use for communicating between the AIPEP and the CVS is based on WS-TRUST [12] and SAMLv2 [13] and the complete profile is specified as an Open Grid Forum profile [14]. The CVS can work in either pull mode, push mode or pull and push mode.

Pull mode means that the requester does not have any credentials and requires the CVS to pull them itself from its configured trusted AAs, or a subset of them. The reason for adding the subset advisory field, is that in a large federation, such as the UK Access Management Federation, there could be several hundred trusted Identity Providers (IdPs) or AAs configured into the CVS's validation policy, and in pull mode one would not want the CVS to ask each of these if it had any credentials for the subject in question. We have also specified an Open Grid Forum profile for the protocol to pull credentials [15], which is based on SAMLv2. Our CVS implementation however is more sophisticated than this, and can also pull X.509 attribute certificates from an LDAP repository or WebDav server [16], LDAP string attributes from a trusted LDAP server, and follow delegation chains of credentials. Our CVS therefore contains a delegation policy that tells it which delegated credentials it can trust. There currently isn't a standard policy language for a CVS policy, and in [9] we say why XACMLv2 is not sufficient for this (neither is XACMLv3).

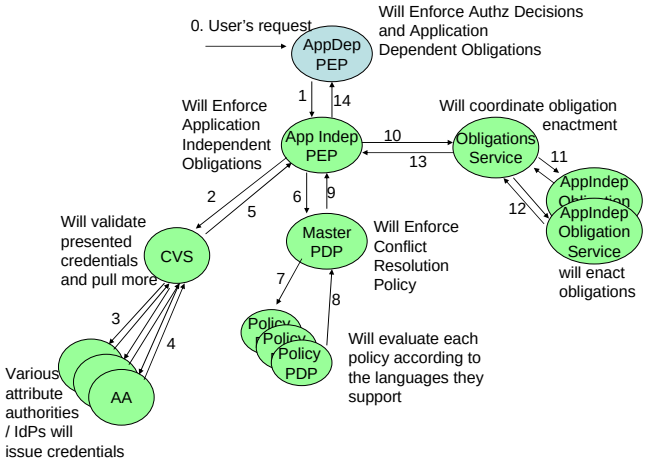

Figure 1. An Advanced Authorisation Infrastructure

Push mode means that the WS-Trust request message contains the full set of credentials which are to be validated by the CVS, and none further need to be pulled. Note that the push/pull dialect field is advisory only, since the CVS is allowed to pull further credentials if it needs to. For example, if the AIPEP sent a delegated credential to be validated and the delegator is not a root of trust in the CVS's policy, then the credentials of the delegator will need to be pulled. Pull and push mode, as its name implies, is requesting the CVS to validate the credentials in the request message and pull any further credentials that it can find for the subject of the authorization decision query. Again an advice field can specify a subset of the AAs/IdPs to pull from. Once the CVS has finished validating the subject's credentials, these are returned to the AIPEP as XACML formatted attributes in step 5 of figure 1, ready to be passed to the Master PDP.

2.3 The Master PDP

In order to evaluate multiple policies in different languages we introduce a new conceptual component called the Master PDP. The Master PDP is responsible for calling the multiple configured policy language specific PDPs (step 7), obtaining their authorization decisions (step 8), and then resolving any

conflicts between these decisions, before returning the overall authorization decision and any resulting obligations to the AIPEP (in step 9). The Master PDP is configured to know each policy language that each subordinate PDP supports, so that when it is passed a sticky policy it knows which PDP to give each component of the sticky policy to. Each of the policy PDPs supports the same interface, which is the SAMLv2 profile of XACMLv2 [10]. This allows the Master PDP to call any number of subordinate PDPs, each configured with its own policy in its own language. The Master PDP can also dynamically pass the policy that is to be used by each PDP at decision time. One of a new breed of PDPs that is currently being built as part of the EC TAS^3 project [6] is a behavioral trust engine which will return an authorization decision about whether the requester is trusted or not to perform the requested action on the specified resource. The policy language for specifying the behavioral trust rules is SWI-Prolog and is still being finalized, but the current design isolates this policy language from the rest of the authorization infrastructure, and the Master PDP will not be affected by any changes in this policy language as it evolves. The behavioral trust PDP will either be configured with its policy via a back channel, or dynamically with the policy that accompanies the authorization decision request message.

2.4 Sticky Policy Contents

We propose a StickyPAD which is a combination of a (set of) **sticky p**olicy(ies) **a**nd the **d**ata to which the policies apply. The stickyPAD is created by the authoritative source of the data (typically the IdP) digitally signing the package. It is the responsibility of the receiving PEP to validate the signature when it receives a StickyPAD message in step 0 of figure 1. The PEP will then parse and unpack valid StickyPAD messages and re-package the various elements in the standard format of [11] ready for passing to the AIPEP.

Each embedded sticky policy in the StickyPAD is flagged with its policy language and its author. Various types of sticky policy may be defined:

- authorization policies – these says who is authorized to perform which actions on the associated data/resource. Each authz policy has an author, so that it can be referred to by the conflict resolution policy.
- conflict resolution policy – says how conflicts between the different authorization policy decisions are to be resolved and which authorisation decision and obligations should be returned to the AIPEP. It must be written by the issuer of the StickyPAD.
- audit policies – say what information should be audited when the associated data/resource is accessed.
- obligations policies – say what actions need to be undertaken by the receiving PEP when it initially receives the StickyPAD.
- privacy policies – contain privacy specific rules such as retention periods and purposes of use. (Note. These may be combined in the authorization policies depending upon the specific authz policy language used, but not all authorization policy languages can support all privacy specific rules).
- authentication policy – says what level of assurance (LoA) is required of requesting subjects who are to be allowed to access the associated data. (Whilst it is possible to represent LoAs in the authorization policy e.g. as described in [17], by keeping this as a separate policy it allows the PEP to short circuit the whole authorization process if the requesting subject has not been authenticated sufficiently.)
- data manipulation policy – provides rules for how the associated data (usually PII) can be transformed, enriched or aggregated with other personal data of the same data subject or of other data subjects.

2.5 Conflict Resolution Policy

The Master PDP is statically configured with a Conflict Resolution Policy which covers access to its static resources. In addition it may be dynamically given a Conflict Resolution Policy taken from a sticky policy attached to dynamic data. The conflict resolution policy states how the decisions and obligations returned from the multiple subordinate PDPs are to be combined together to produce a single result. For example, if the dynamic data is the favorite drink PII attribute of a data subject, then the data subject would be the author of the sticky conflict resolution policy, whereas for a data subject's criminal record the legal system would be the author of the sticky conflict resolution policy. For a static computing resource, the organization (resource owner) would be the author of the static conflict resolution policy. The Master PDP ensures that any resulting authorization decision conflicts are handled in accordance with the wishes of the author of the current conflict resolution policy.

The XACML standard has a reasonably comprehensive section describing policy combining algorithms. XACMLv2 [1] defines the following policy combining rules:

- Deny-overrides (both Ordered and Unordered) – A Deny result over-rides all other results, but otherwise the other results (Permit, Indeterminate and NotApplicable) may still be returned.
- Permit-overrides (both Ordered and Unordered) – A Permit result over-rides all other results, but otherwise the other results (Deny, Indeterminate and NotApplicable) may still be returned.
- First-applicable – After a policy has computed a Deny or Permit result, processing of all further policies stops and this first result is returned.
- Only-one-applicable – If from the set of policies only one policy is applicable to the request, then the result of evaluating this is returned. If however, multiple policies are applicable then Indeterminate is returned.

In addition, XACMLv3 [2] defines the Deny-unless-permit and Permit-unless-deny algorithms. The purpose of these is to ensure that an Indeterminate or NotApplicable result is never returned to the PEP by the PDP.

We can use the XACML policy combining rules as the basis for the Master PDP's conflict resolution policy. However, the above rule set does not have a rule that allows the Master PDP to preferentially choose the authorization policy of the authoritative source. We need an additional rule, specifically:

- Identified Author's policy overrides – this rule states that the authorization policy written by the identified author takes precedence over all other authorization policies.

The conflict resolution policy may also contain sets of

obligations that should be returned with either Deny or Permit results, in addition to those returned by the PDPs.

2.6 Obligations Service

Obligations may be required *before* the user's action is performed, *after* the user's action has been performed, or simultaneously *with* the performance of the user's action [8]. We call this the *temporal type* of the obligation, which can be set to either *before*, *with*, or *after*. Examples are as follows: before the user is given access, increase the amount of logging to monitor what he is doing; after the user has been given access, record the amount of cpu that was used; simultaneously with the user's access, decrement his account balance. As described in [8] the failure semantics of each are as follows: a *before* obligation will be enacted even if the user's action subsequently fails, an *after* obligation may fail to be performed even if the user's action succeeds, and a *with* obligation should only succeed if the user's action succeeds, and should fail if the user's action fails. The presence of a *with* temporal type means that these obligations (at least) have to support two-phase commit, and be prepared to rollback their effects if the user's action fails. If the user's action succeeds then the *with* obligation can be told to commit to its actions.

According to the XACML model, each obligation has a unique ID (a URI). The obligations service will know which obligations it can support, by being configured at construction time with the set of obligation IDs that are supported. When passed a set of obligations by the AIPEP, the obligations service will walk through this list and call the appropriate application independent obligation service. If any single obligation service returns an error, then the obligations service must stop further processing and return an error to the AIPEP. If all obligations are processed successfully, a success result can be returned. Each of the application independent obligation services must be of temporal type *before*, otherwise they cannot be enacted by the AIPEP.

3. CONCLUSIONS AND FUTURE PLANS

The advanced authorization infrastructure described here is currently being constructed as part of the EC TAS³ project. We have already constructed the AIPEP, the obligations service and the CVS and integrated these with both the PERMIS and XACML PDPs using the protocols specified in [11], [14] and [15]. We have implemented state based Break The Glass policies using the AIPEP, obligations service and a stateless PDP. A live demo of BTG is available at http://issrg-testbed-2.cs.kent.ac.uk/. Our next step is to implement the Master PDP and conflict resolution policy. We then plan to define an application independent protocol for carrying sticky policies between systems using the AIPEP instead of the PEP, as described in [19].

4. ACKNOWLEDGMENTS

The research leading to these results has received funding from the EC's FP7 programme under grant agreement n° 216287 (TAS³ - Trusted Architecture for Securely Shared Services).

5. REFERENCES

[1] OASIS "eXtensible Access Control Markup Language (XACML) Version 2.0"OASIS Standard, 1 Feb 2005

[2] OASIS. "eXtensible Access Control Markup Language (XACML) Version 3.0". Committee draft 1. 16 April 2009

[3] D.W.Chadwick, G.Zhao, S.Otenko, R.Laborde, L.Su and T.A.Nguyen. "PERMIS: a modular authorization infrastructure". Conc. Comp. Prac. Exp, Vol.20, Issue 11, 10 Aug 2008. Pages 1341-1357.

[4] W3C: The Platform for Privacy Preferences 1.0 (P3P 1.0). Technical Report. 2002

[5] M.Blaze, J.Feigenbaum, J.Ioannidis. "The KeyNote Trust-Management System Version 2", RFC 2704, Sept. 1999.

[6] See http://www.tas3.eu

[7] Alqatawna, J.; Rissanen, E.; Sadighi, B. "Overriding of Access Control in XACML".Proc. 8th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY '07) 13-15 June 2007. Pages:87 – 95

[8] D.W.Chadwick, L.Su, R.Laborde. "Coordinating Access Control in Grid Services". Conc. Comp. Prac. Exp., Vol. 20, Issue 9, 25 June 2008, Pages 1071-1094.

[9] D.W.Chadwick, S.Otenko, T.A. Nguyen. "Adding Support to XACML for Multi-Domain User to User Dynamic Delegation of Authority". Int.J. Inf. Sec. Vol. 8, No 2 / April, 2009 pp 137-152

[10] OASIS "SAML 2.0 profile of XACML v2.0" Committee Draft, 16 April 2009

[11] D.W.Chadwick, L.Su, R.Laborde. "Use of XACML Request Context to access a PDP". OGF GWD-R-P. 25 June 2009

[12] OASIS, "WS-Trust 1.3", OASIS Standard, 19 March 2007

[13] OASIS. "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard, 15 March 2005

[14] D.W.Chadwick, L. Su. "Use of WS-TRUST and SAML to access a Credential Validation Service". OGF GWD-R-P, 25 June 2009.

[15] V. Venturi, T. Scavo, D.W. Chadwick, "Use of SAML to retrieve Authorization Credentials", OGF GWD-R-P, 25 June 2009

[16] D.W.Chadwick, S.Anthony. "Using WebDAV for Improved Certificate Revocation and Publication". LCNS 4582, "Public Key Infrastructure. Proc of 4th European PKI Workshop, June, 2007, Spain. pp 265-279

[17] N. Zhang, L. Yao, A. Nenadic, J. Chin, C. Goble, A. Rector, D. Chadwick, S. Otenko and Q. Shi; "Achieving Fine-grained Access Control in Virtual Organisations", Conc. Comp. Prac. Exp., Vol. 19, Issue 9, June 2007, pp. 1333-1352.

[18] M.C.Mont, S.Pearson, P.Bramhall. "Towards accountable management of identity and privacy: sticky policies and enforceable tracing services". Proc 14th Int Workshop on Database and Expert Systems Applications, 1-5 Sept. 2003. Page(s): 377 – 382

[19] D.W.Chadwick, S.F.Lievens. "Enforcing "Sticky" Security Policies throughout a Distributed Application". MidSec 2008. December 1-5, 2008, Leuven, Belgium