The PERMIS X.509 Role Based Privilege Management Infrastructure

D.W.Chadwick, O.Otenko, ISI, University of Salford, Salford, M5 4WT

ABSTRACT

This paper describes the output of the PERMIS project, which has developed a role based access control infrastructure that uses X.509 attribute certificates (ACs) to store the users' roles. All access control decisions are driven by an authorization policy, which is itself stored in an X.509 attribute certificate, thus guaranteeing its integrity. All the ACs can be stored in one or more LDAP directories, thus making them widely available. Authorization policies are written in XML according to a DTD that has been published at XML.org. The Access Control Decision Function (ADF) is written in Java and the Java API is simple to use, comprising of just 3 methods and a constructor. There is also a Privilege Allocator, which is a tool that constructs and signs attribute certificates and stores them in an LDAP directory for subsequent use by the ADF.

Categories and Subject Descriptors

C.2.0 [**General**] Security and protection K.6.5 [**Security and Protection**] Authentication J.1 [**Administrative Data Processing**] Government

General Terms

Management, Design, Security.

Keywords

X.509, Privilege Management Infrastructure, Authorization, RBAC, Attribute Certificates, XML, Policies

1. INTRODUCTION TO X.509

In order to control access to a resource, both authentication and authorization are needed. Early versions of the ITU-T X.509 standard [7] have concentrated on standardizing strong authentication techniques, based on digital signatures, public key certificates, and Public Key Infrastructures (PKIs). This paper assumes that the reader is already familiar with the general concepts of PKIs, and these will not be repeated here. Readers wishing to learn more about PKIs may consult texts such as Adams [1], Austin [2] or Housley [6].

The latest version of X.509, due to be published in 2002, is the first edition to standardize an authorization technique and this is based on attribute certificates and Privilege Management Infrastructures (PMIs).

A PMI is to authorization what a PKI is to authentication. Consequently there are many similar concepts shared between PKIs and PMIs. These are summarized in Table 1.

Concept	PKI entity	PMI entity
Certificate	Public Key Certificate (PKC)	Attribute Certificate (AC)
Certificate issuer	Certification Authority (CA)	Attribute Authority (AA)
Certificate user	Subject	Holder
Certificate binding	Subject's Name to Public Key	Holder's Name to Privilege Attribute(s)
Revocation	Certificate Revocation List (CRL)	Attribute Certificate Revocation List (ACRL)
Root of trust	Root Certification Authority or Trust Anchor	Source of Authority (SOA)
Subordinate authority	Subordinate Certification Authority	Attribute Authority (AA)

Table 1. A Comparison of PKIs and PMIs

A public key certificate (PKC) is used for authentication and maintains a strong binding between a user's name and his public key, whilst an attribute certificate (AC) is used for authorization and maintains a strong binding between a user's name and one or more privilege attributes. The entity that digitally signs a public key certificate is called a Certification Authority (CA), whilst the entity that digitally

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SACMAT'02, June 3-4, 2002, Monterey, California, USA.

Copyright 2002 ACM 1-58113-496-7/02/0006…$5.00.

signs an attribute certificate is called an Attribute Authority (AA). The root of trust of a PKI is sometimes called the root CA[1] whilst the root of trust of the PMI is called the Source of Authority (SOA). CAs may have subordinate CAs that they trust, and to which they delegate the powers of authentication and certification. Similarly, SOAs may delegate their powers of authorization to subordinate AAs. If a user needs to have his signing key revoked, a CA will issue a certificate revocation list (CRL). Similarly if a user needs to have his authorization permissions revoked, an AA will issue an attribute certificate revocation list (ACRL).

2. IMPLEMENTING AUTHORIZATION SCHEMES WITH X.509

Various authorization schemes have been devised in the past. The traditional, most popular and well-known model is the Discretionary Access Control (DAC) scheme [9]. In the DAC scheme, users are optionally given access rights to resources by the resource administrator. In traditional systems the access rights are typically held as access control lists within each target resource. In an X.509 PMI, the access rights are held within the privilege attributes of attribute certificates issued to users. Each privilege attribute within an AC will describe one or more of the user's access rights. A target resource will then read a user's AC to see if he is allowed to perform the action that he is requesting.

Another authorization scheme, popular with the military, is the Multilevel Secure (MLS) system, which is a type of Mandatory Access Control (MAC) scheme. In the MLS scheme, every target is given a security label, which includes a classification, and every subject is given a clearance, which includes a classification list. The classification list specifies which type of classified target the subject is allowed to access. A typical hierarchical classification scheme used by the military is: unmarked, unclassified, restricted, confidential, secret, and top secret. A typical security policy, designed to stop information leakage, is "read down and write up". This specifies that a subject can read targets with a lower classification than his clearance, and can write to targets with a higher classification. A user with clearance of confidential, who logs in as such, under this policy could read from unmarked to confidential targets, and write to confidential to top secret targets. The same user could also log in with a lower clearance level, say, unclassified and write to an unclassified target. X.509 supports MLS, by allowing subjects to be given a clearance AC. The privilege attribute in the AC now holds the users clearance. Targets can be securely configured with their own security label and the security policy that is to direct them.

More recently, research has focused on Role Based Access Controls (RBAC) [10]. In the basic RBAC model, a number of roles are defined. These roles typically represent organizational roles such as secretary, manager, employee etc. In the authorization policy, each role is given a set of permissions i.e. the ability to perform certain actions on certain targets. Each user is then assigned to one or more roles. When accessing a target, a user presents his role(s), and the target reads the policy to see if this role is allowed to perform this action. X.509 supports simple RBAC by defining role specification attribute certificates that hold the permissions granted to each role, and role assignment attribute certificates that assign various roles to the users. In the former case, the AC holder is the role, and the privilege attributes are permissions granted to the role. In the latter case the AC holder is the user, and the privilege attributes are the roles assigned to the user.

The hierarchical RBAC model is a more sophisticated version of the basic RBAC model. With this model, the roles are organized hierarchically, and the senior roles inherit the privileges of the more junior roles. So for example we might have the following hierarchy:

employee > programmer > manager > director.

If a privilege is given to an employee role e.g. can enter main building, then each of the superior roles can also enter the main building even though their role specification does not explicitly state this. If a programmer is given permission to enter the computer building, then managers and directors would also inherit this permission. Hierarchical roles mean that role specifications are more compact. X.509 supports hierarchical RBAC by allowing both roles and privileges to be inserted as attributes in a role specification attribute certificate, so that the latter role inherits the privileges of the encapsulated roles.

Another extension to basic RBAC is constrained RBAC. This allows various constraints to be applied to the role and permission assignments. One common constraint is that certain roles are declared to be mutually exclusive, meaning that the same person cannot simultaneously hold more than one role from the mutually exclusive set. For example, the roles of student and examiner, or the roles of tenderer (one who submits a tender) and tender officer (one who opens submitted tenders) would both be examples of mutually exclusive sets. Another constraint might be placed on the number of roles a person can hold, or the number of people who can hold a particular role. X.509 only has a limited number of ways of supporting constrained RBAC. Time constraints can be placed on the validity period of a role assignment attribute certificate. Constraints can be placed on the targets at which a permission can be used, and on the policies under which an attribute certificate can confer privileges. Constraints can also be placed on the delegation of roles. However many of the constraints,

[1] Unfortunately X.509 did not standardize the term root CA or the term for the root of trust. The latter is sometimes also known as the trust anchor, and disparate meanings for the former have evolved.

such as the mutual exclusivity of roles, have to be enforced by mechanisms outside the attribute certificate construct e.g. within the privilege management policy enforcement function.

3. THE EC PERMIS PROJECT

The EC funded PERMIS project has been given the challenge of building an X.509 role based PMI that can be used by very different applications in 3 cities of Europe. The project has members from the cities of Barcelona (Spain), Bologna (Italy) and Salford (UK). All three centers already have experience of running pilot PKIs, and so it was natural for them to want to add a PMI capability so as to complete the strong authentication and authorization chain. The chosen applications of the 3 cities are very different in character, and it will be a good test of the generality of the developed PMI if it can accommodate them.

In the case of Bologna, the city wants to be able to allow architects to download street maps of the city, to update the maps with their proposed plans, and to upload the new building plans and requests for building licenses to the city planning office's server. This should significantly improve the efficiency of the current system, as the plans and requests are currently sent by post as paper documents to the city hall.

Barcelona is a major tourist and commercial center and has many car hire locations throughout the city and at the airport. However, parking in Barcelona is very restricted, and many parking tickets are frequently issued to hired cars. By the time the car hire companies receive the parking tickets, the hirers have long since left the country. The plan is to give the car hire companies on line access to the city's parking ticket database, so that when cars are returned at the end of their hire period, the company can instantly check to see if any parking tickets have been issued for this car. The company will be able to send the details of the driver to the city, thereby transferring the fine to the individual. Data protection legislation requires that a car hire company can only access the tickets issued to its own cars, and not to those of other car hire companies, and so authorization will need to be at the record level.

Finally Salford is implementing an electronic tendering application. The tendering process will start when the city places the Request for Proposal documents on its web site, allowing anyone to download them. However, in some restricted tendering instances, only companies previously authorized by Salford will be able to submit tenders. In other cases it may be a requirement that a company has ISO 9000 or other certification in order to submit a tender. Once the tenders have been submitted, they must remain anonymous until the winner has been chosen. The city tender officers must not be given access to the electronic tender store before the closing date of the RFP, and tenderers must not be allowed to submit tenders after the closing date of the RFP.

The challenge for the PERMIS project is to build a role based X.509 privilege management infrastructure that can cater for these very different applications, and in so doing indicate that it will be useful to a much wider range of applications.

4. The PERMIS PMI ARCHITECTURE

The PERMIS PMI architecture comprises a privilege allocation subsystem and a privilege verification subsystem. The privilege allocation subsystem (see Figure 1) is responsible for allocating privileges to the users. There can be any number of these in a full system. The privilege allocator (see later) issues X.509 role assignment attribute certificates to users and stores these in an LDAP directory for subsequent use by the privilege verification subsystem. In addition to privilege allocation, each user will also need to be issued with an application specific authentication token. If a PKI is being used, this will be a digitally signed public key certificate, if a conventional authentication system is being used it will be a username/password pair. The privilege verification subsystem (see Figure 2) is responsible for authenticating and authorizing the users. Authentication is performed in an application-specific manner, but authorization is performed in an application-independent manner according to the PERMIS RBAC authorization policy. In this way one policy can control access to all resources in a domain.

4.1 The Authorization Policy

The authorization policy specifies who has what type of access to which targets, and under what conditions. Domain wide policy authorization is far more preferable than having separate access control lists configured into each target. The latter is hard to manage, duplicates the effort of the administrators (since the task has to be repeated for each target), and is less secure since it is very difficult to keep track of which access rights any particular user has across the whole domain. Policy based authorization on the other hand allows the domain administrator (the SOA) to specify the authorization policy for the whole domain, and all targets will then be controlled by the same set of rules.

The PERMIS project decided very early on to use the hierarchical RBAC model for specifying authorizations. RBAC has the advantage of scalability over DAC, and can easily handle large numbers of users, as there are typically far fewer roles than users.

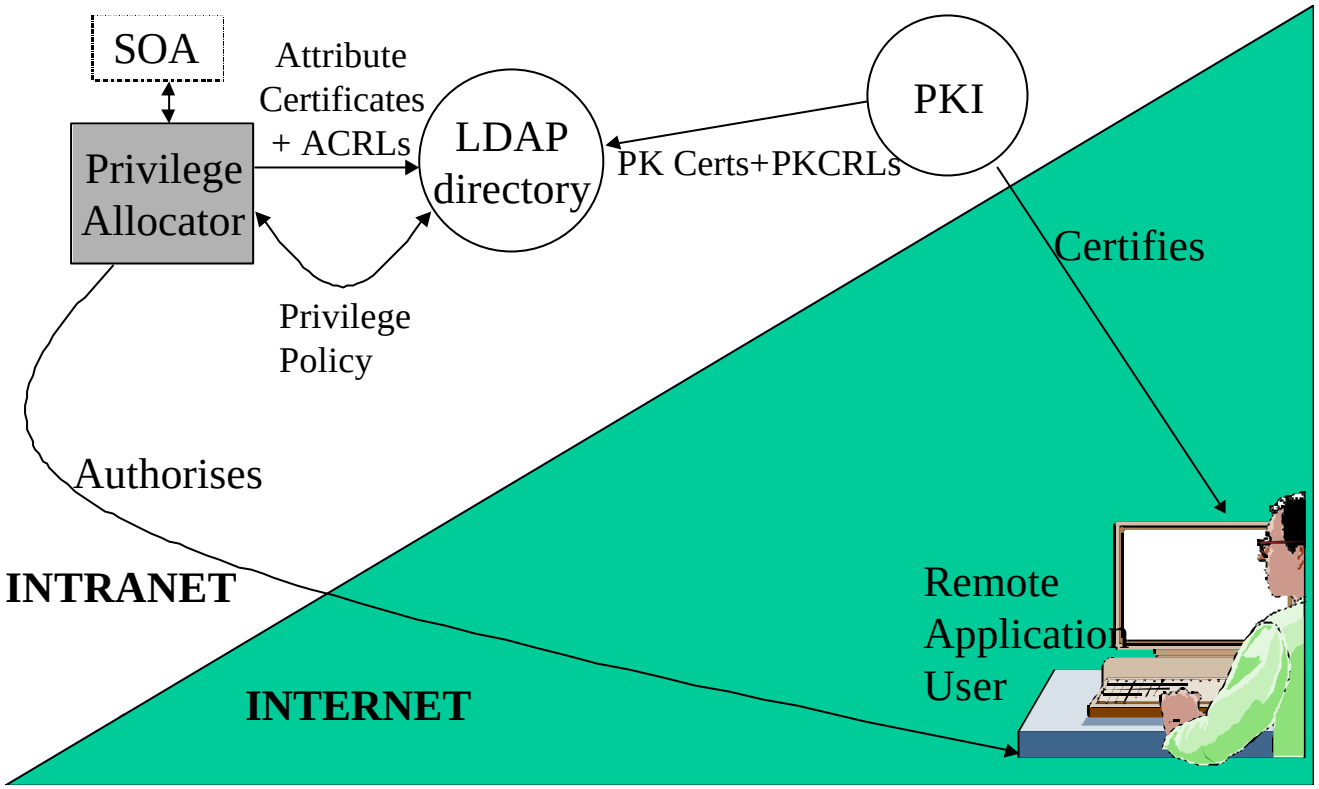


Figure 1. The Privilege Allocation Subsystem

The PERMIS project wanted to specify the authorization policy in a language that could be both easily parsed by computers, and read by the SOAs (with or without software tools). We looked at various pre-existing policy languages e.g. Ponder [5], Keynote [3], but found that none were ideally suited to our needs. We decided that XML was a good candidate for a policy specification language, since there are lots of tools around that support XML, it is fast becoming an industry standard, and raw XML can be read and understood by many technical people (as opposed to ASN.1[2] for example, which uses a binary encoding).

First we specified a Data Type Definition (DTD) for our X.500 PMI RBAC Policy. The DTD is a meta-language that holds the rules for creating the XML policies. The DTD comprises the following components:

- SubjectPolicy – this specifies the subject domains i.e. only users from a subject domain may be authorized to access resources covered by the policy
- RoleHierarchyPolicy – this specifies the different roles and their hierarchical relationships to each other
- SOAPolicy – this specifies which SOAs are trusted to allocate roles, and permits the distributed managements of role allocation to take place
- RoleAssignmentPolicy – this specifies which roles may be allocated to which subjects by which SOAs, whether delegation of roles may take place or not, and how long the roles may be assigned for
- TargetPolicy – this specifies the target domains covered by this policy
- ActionPolicy – this specifies the actions (or methods) supported by the targets, along with the parameters that should be passed along with each action e.g. action Open with parameter Filename

[2] Abstract Syntax Notation One (ASN.1) is the syntax in which X.509 certificates are specified.

- TargetAccessPolicy – this specifies which roles have permission to perform which actions on which targets, and under which conditions. Conditions are specified using Boolean logic and might contain constraints such as “IF time is GT 9am AND time is LT 5pm OR IF Calling IP address is a subset of 125.67.x.x”. All actions that are not specified in a Target Access Policy are denied.

A full description of the policy can be found in [4].

Table 2 shows a portion of the DTD that specifies the rules for the Role Assignment Policy, and below it is an example Role Assignment Policy for the Salford application.

Table 2. The Role Assignment DTD and an Example Role Assignment Policy

```
<!ELEMENT RoleAssignmentPolicy (RoleAssignment)+ >
<!ELEMENT RoleAssignment  (SubjectDomain,Role,Delegate,SOA,Validity) >

<!ELEMENT SubjectDomain EMPTY>
<!ATTLIST SubjectDomain ID IDREF #REQUIRED>

<!ELEMENT Role EMPTY >
<!ATTLIST Role Type IDREF #IMPLIED
        Value IDREF #IMPLIED >

<!ELEMENT SOA EMPTY>
<!ATTLIST SOA ID IDREF #REQUIRED>

<!ELEMENT Validity (Absolute?, Maximum?, Minimum? ) >
<!ELEMENT Absolute EMPTY>
<!ATTLIST Absolute Start CDATA #IMPLIED
            End CDATA #IMPLIED >
<!ELEMENT Maximum EMPTY>
<!ATTLIST Maximum Time CDATA #IMPLIED >
<!ELEMENT Minimum EMPTY>
<!ATTLIST Minimum Time CDATA #IMPLIED >

<!ELEMENT Delegate EMPTY >
<!ATTLIST Delegate Depth CDATA #IMPLIED >
```

```
<RoleAssignmentPolicy>
   <RoleAssignment>
<!-- Role assignment for tender officers. They must be employees of Salford City Council.  Valid only from close of tender.  Delegation not permitted -->
     <SubjectDomain ID="Employees"/>
```

```
    <Role Type="permisRole" Value="TenderOfficer"/>
    <Delegate Depth="0"/>
    <SOA ID="Salford"/>
    <Validity>
      <Absolute Start="2001-09-21T17:00:00"/>
    </Validity>
  </RoleAssignment>
  <RoleAssignment>
<!-- Role assignment for tenderers. They must be dot com or co.uk companies.  Valid only until close of tender. Delegation not permitted -->
    <SubjectDomain ID="Companies"/>
    <Role Type="permisRole" Value="Tenderer"/>
    <Delegate Depth="0"/>
    <SOA ID="Salford"/>
    <Validity>
      <Absolute End="2001-09-21T17:00:00"/>
    </Validity>
  </RoleAssignment>
  <RoleAssignment>
<!-- Role assignment for companies who are ISO9000 Certified. They must be dot com or co.uk companies.  Valid only for a maximum of one year, as companies have to be annually re-accredited. Certificates are issued by BSI. Delegation not permitted -->
    <SubjectDomain ID="Companies"/>
    <Role Type="ISOCertified" Value="ISO9000"/>
    <Delegate Depth="0"/>
    <SOA ID="BSI"/>
    <Validity>
      <Maximum Time="+01"/>
    </Validity>
  </RoleAssignment>
</RoleAssignmentPolicy>
```

The SOA creates the authorization policy for the domain using his favorite XML editing tool, and stores this in a local file, say MyPolicy.XML. This file will subsequently be used by the Privilege Allocator to create the policy AC. Each XML policy is given a globally unique object identifier by the SOA during creation, to ensure that the correct policy is used during operation.

4.2 The Privilege Allocator

The Privilege Allocator (PA) is a tool used by the SOA or an AA to allocate privileges to users, and also to digitally sign the authorization policy. The PA can be run by anyone who has a private/public key pair. Since PERMIS is using RBAC, the SOA uses the PA to allocate roles to users in the form of role assignment ACs. A role in PERMIS is simply defined as an attribute type and value. We are using two attribute types *permisRole* and *ISOCertified*, whose values are IA5 strings. In the case of Bologna, there are two permisRole values: Map-Readers and Architects. Map-Readers can download any maps produced by the municipality, whereas Architects are allowed to download maps and upload digitally modified maps. In the case of Barcelona, there are also two permisRoles defined: Generalised and Authorised. Any citizen or business can be allocated the Generalised role. Anyone with the Generalised role has permission to read their own pending car parking fines. Businesses that have signed an agreement with the Barcelona city council are given the Authorised role. Authorised roles can read their own pending fines and also may modify the details of them (e.g. update the driver's name and address). Salford is different to

the other sites, in that whilst it will allocate two permisRoles, that of Tenderer and Tender-Officer, it will also rely on an external SOA (in this case the British Standards Institute) to allocate the ISOCertified roles to users. (In fact in the project we plan to set up a proxy BSI SOA to allocate these roles, as BSI is not a project partner.)

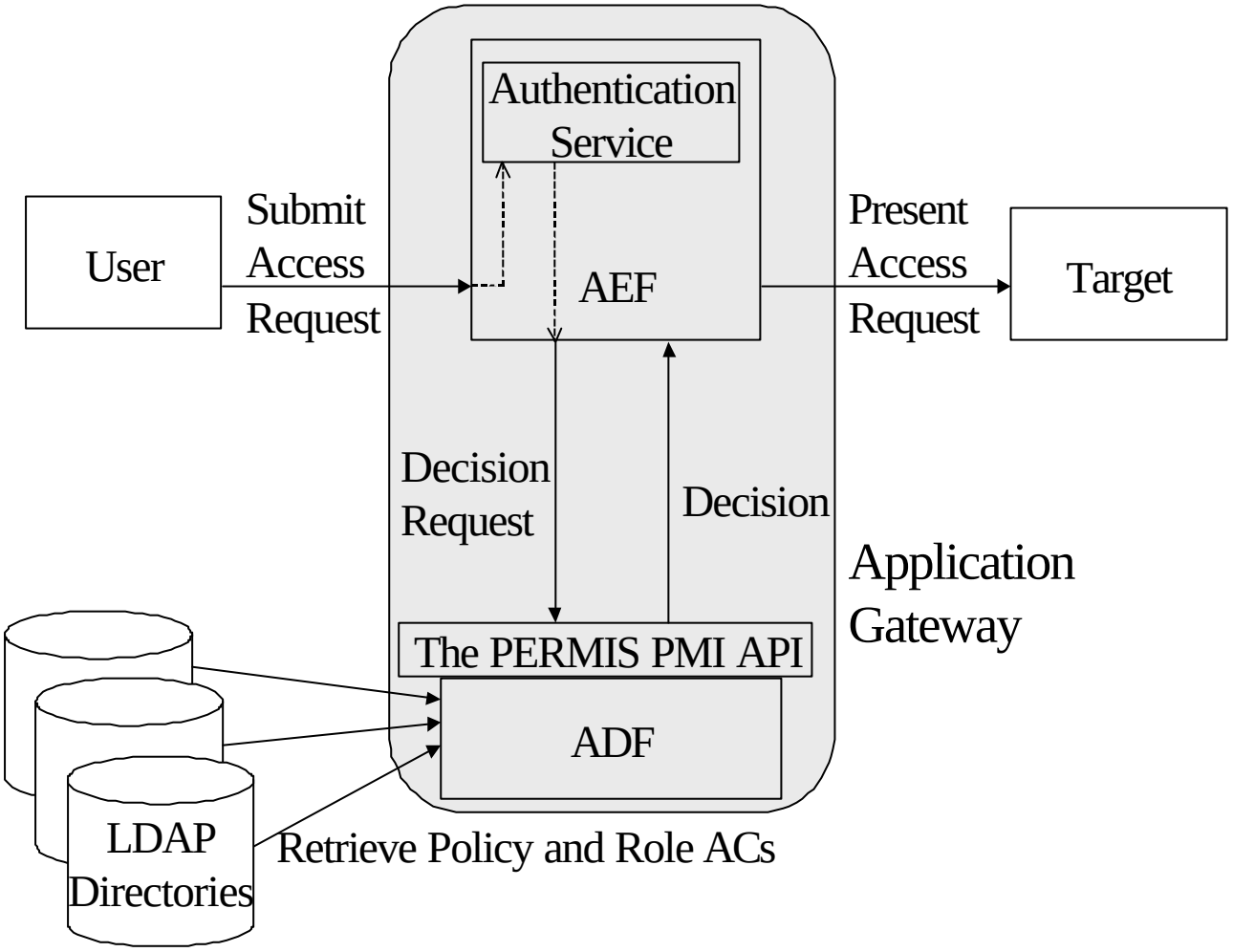


Figure 2. The Privilege Verification Subsystem

Once the role assignment ACs have been created by the PA they are stored in an LDAP directory. Since ACs are digitally signed by the SOA/AA who issued them, they are tamper-resistant, and therefore there is no modification risk from allowing them to be stored in a publicly accessible LDAP directory. This also means that authorities that issue digital ACs can store them locally, but give global access to them. We think this will be particularly useful in the case of ISO 9000 certificates, for example. Anyone wishing to know if an organization has ISO 9000 certification may access the BSI LDAP directory and retrieve the organization's X.509 AC. The ACRLs of revoked certificates (if any) will also be stored here. Thus there is little advantage in general of distributing the ACs to their holders, since a relying party will still need to access the issuing authorities LDAP directory to retrieve the latest ACRL (or call an OCSP responder once these become available). We see that this mechanism can be extended to any type of privilege certification e.g. Microsoft Certified Engineer, BSc (Hons) Maths University of Salford etc. and that these "roles" can easily be built into our API via the authorization policy. We are already using this with an electronic prescribing system that we are building, to allow the Royal College of Pharmacy to allocate pharmacist roles to qualified pharmacists, and the General Medical Council to allocate prescriber roles to qualified doctors.

Another function of the PA is to digitally-sign the authorization policy and create a policy AC. The policy AC is a standard X.509 AC, with the following special characteristics: the holder and issuer names are the same (i.e. that of the SOA), the attribute type is pmiXMLPolicy and the attribute value is the XML policy created as described above. The PA prompts the SOA for the name of the policy file (e.g. MyPolicy.XML) and then it copies the contents into the attribute value. After the SOA has signed the policy AC, the PA stores it in the SOA's entry in the LDAP directory.

4.3 The Privilege Verification Subsystem

The privilege verification subsystem (see Figure 2) is responsible for authenticating and authorizing the user. Authentication is application-specific, but authorization is application-independent. The Open Group has already defined a standard authorization API. It is called the AZN API [11], and is specified in the C language. It is based on the ISO 10181-3 Access Control Framework [8], and specifies the interface between the application-specific Access Control Enforcement Function (AEF) and the application-independent Access Control Decision Function (ADF). PERMIS has drawn on this work and made the following changes to it. Firstly we have specified the PERMIS PMI API in Java rather than in C, and secondly we have significantly simplified the AZN API by assuming that the Target and the AEF are either co-located or can communicate with each other across a trusted LAN. Without this latter simplification the authorization token carried from the AEF to the Target would need to be protected in some way, for example as an X.509 attribute certificate, and so we would have gained very little from our design. We also assume that only a single authorization service will be available, and that the API doesn't need to support the export of authorization tokens, as ACs are already in an exportable format.

In this model, a user accesses resources via an application gateway. The AEF authenticates the user, and then asks the ADF if the user is allowed to perform the requested action on the particular target resource. The ADF accesses one or more LDAP directories to retrieve the authorization policy and the role ACs for the user, and bases its decision on these.

4.4 The PERMIS PMI API

The PERMIS API comprises 3 simple methods: GetCreds, Decision, and Shutdown, and a Constructor. The Constructor builds the PERMIS API java object. For construction, the AEF passes the name of the trusted SOA (the root of trust for authorization), the object identifier of the policy, and a list of LDAP URIs from where the ADF can retrieve the policy AC and subsequently the role ACs. The Constructor is usually called immediately the AEF starts up. After construction of the API has completed, the ADF will have read in and validated the XML policy that will control all future decisions that it makes. As each XML policy was given a globally unique object identifier by the SOA during its creation, the ADF can ensure that the correct policy is read in, and if multiple policies exist, which one to keep.

When a user initiates a call to the target, the AEF authenticates the user, then passes the LDAP DN of the user to the ADF through a call to GetCreds. In the 3 cities the users will be authenticating in different ways. In Salford the user will be sending an S/MIME email message to the AEF, in Barcelona and Bologna he will be opening an SSL connection. In all cases the user will be digitally signing the opening message, and verification of the signature will yield the user's DN. The ADF uses this DN to retrieve all the role ACs of the user from the list of LDAP URIs passed at initialization time (the "pull" model). The role ACs are validated against the policy e.g. to check that the DN is within a valid subject domain, and to check that the ACs are within the validity time of the policy etc. Invalid role ACs are discarded, whilst the roles from the valid ACs are extracted and kept for the user. GetCreds also supports the "push" model, whereby the AEF can pass a set of ACs to the ADF, instead of the ADF retrieving them from the LDAP directories.

Once the user has been successfully authenticated he will attempt to perform certain actions on the target. At each attempt, the AEF passes the target name and the attempted action along with its parameters, to the ADF via a call to Decision. Decision checks if the action is allowed for the roles that the user has, taking into account all the conditions specified in the TargetAccessPolicy. If the action is allowed, Decision returns Granted, if it is not allowed it returns Denied. The user may attempt an arbitrary number of actions on different targets, and Decision is called for each one. In order to stop the user keeping the connection open for an infinite amount of time (for example until after his ACs have expired), the PERMIS API supports the concept of a session time out. On the call to GetCreds the AEF can say how long the session may stay open before the credentials should be refreshed. If the session times out, then Decision will throw an exception, telling the AEF to either close the user's connection or call GetCreds again.

Shutdown can be called by the AEF at any time. Its purpose is to terminate the ADF and cause the current policy to be discarded. This could happen when the application is gracefully shutdown, or if the SOA wants to dynamically impose a new authorization policy on the domain. The AEF can follow the call to Shutdown with a new Constructor call, and this will cause the ADF to read in the latest authorization policy and be ready to make access control decisions again.

5. PROGRESS TO DATE

Version 7 of the PERMIS X.500 PMI RBAC Policy DTD has been published at http://www.xml.org, and is also available from http://sec.isi.salford.ac.uk/permis/Policy.dtd. Version 1 of the Privilege Allocator tool has been released to the PERMIS project

participants, as has Version 1.0 of the PERMIS PMI API. Public releases of both are available for non-commercial use after first agreeing to an appropriate license agreement available at our web site.

The generality of the PERMIS API has already proven its worth. In another research project at Salford we are designing an electronic prescription processing system. We have found that the PERMIS API can be easily incorporated into the electronic dispensing application. With a suitable policy the ADF is able to make decisions about whether a doctor is allowed to issue a prescription or not, whether a pharmacist is allowed to dispense a prescription or not, and whether a patient is entitled to free prescriptions or not. It is expected that many more applications will use the API in due course.

6. ACKNOWLEDGMENTS

The authors would like to thank the European Union for partially funding this project under the Information Society Initiative For Standardization (ISIS) program Contract Number 503163, and also the UK EPSRC for funding the Electronic Prescription Processing project under grant number GR/M83483. The authors would also like to thank Entrust Technologies for making their PKI security software available to the University of Salford on preferential terms.

7. REFERENCES

[1] Adams, C., Lloyd, S. (1999). "Understanding Public-Key Infrastructure: Concepts, Standards, and Deployment Considerations". Macmillan Technical Publishing, 1999

[2] Austin, T. "PKI, A Wiley Tech Brief", John Wiley and Son, ISBN: 0-471-35380-9, 2000

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

[4] Chadwick, D.W., Otenko, A. "RBAC Policies in XML for X.509 Based Privilege Management" to be presented at SEC 2002, Egypt, May 2002

[5] Damianou, N., Dulay, N., Lupu, E., Sloman, M. "The Ponder Policy Specification Language", Proc Policy 2001, Workshop on Policies for Distributed Systems and Networks, Bristol, UK 29-31 Jan 2001, Springer-Verlag LNCS 1995, pp 18-39

[6] Housley, R., Polk, T. "Planning for PKI: Best Practices Guide for Deploying Public Key Infrastructure". John Wiley and Son, ISBN: 0-471-39702-4, 2001

[7] ITU-T Rec. X.509 (2000) | ISO/IEC 9594-8 The Directory: Authentication Framework

[8] ITU-T Rec X.812 (1995) | ISO/IEC 10181-3:1996 "Security Frameworks for open systems: Access control framework

[9] Sandhu, R. and Samarati, P. "Access controls, principles and practice". IEEE Communications, 32(9), pp 40-48, 1994

[10] Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E. "Role Based Access Control Models". IEEE Computer 29, 2 (Feb 1996), p38-43.

[11] The Open Group. "Authorization (AZN) API", January 2000, ISBN 1-85912-266-3