Kent Academic Repository

Full text document (pdf)

Citation for published version

Chadwick, David W. (2002) The PERMIS X.509 Based Privilege Management Infrastructure. IS Institute, University of Salford Salford M5 4WT England.

DOI

Link to record in KAR

https://kar.kent.ac.uk/13813/

Document Version

UNSPECIFIED

Copyright & reuse
Content in the Kent Academic Repository is made available for research purposes. Unless otherwise stated all content is protected by copyright and in the absence of an open licence (eg Creative Commons), permissions for further reuse of content should be sought from the publisher, author or other copyright holder.

Versions of research
The version in the Kent Academic Repository may differ from the final published version.
Users are advised to check **http://kar.kent.ac.uk** for the status of the paper. **Users should always cite the published version of record.**

Enquiries
For any further enquiries regarding the licence status of this document, please contact:
researchsupport@kent.ac.uk

If you believe this document infringes copyright then please contact the KAR admin team with the take-down information provided at **http://kar.kent.ac.uk/contact.html**

KAR Kent Academic Repository

University of Kent

Internet-Draft
AAAarch RG
Intended Category: Informational

David Chadwick
University of Salford

Expires: 11 October 2002

11 April 2002

The PERMIS X.509 Based Privilege Management Infrastructure
<draft-irtf-aaaarch-permis-00.txt>

STATUS OF THIS MEMO

This document is an Internet-Draft and is in full conformance with all the provisions of Section 10 of RFC2026 [1].

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft expires on 11 October 2002.

Comments and suggestions on this document are encouraged. Comments on this document should be sent to the AAAarch working group discussion list:

aaaarch@fokus.gmd.de

or directly to the author.

ABSTRACT

This document describes the PERMIS X.509 Based Privilege Management Infrastructure, which is a trust management system as described in RFC 2704 [2]. The PERMIS Infrastructure is compared with the AAA Authorisation Framework described in RFC 2904 [4], and is shown to be compatible with it.

1. Introduction

RFC 2704 describes the KeyNote trust management system, which provides a unified approach to specifying and interpreting authorisation policies, credentials, and relationships for use by Internet services.

RFC 2904 provides an architectural framework for understanding the authorisation of Internet resources and services.

Version 4 of X.509 [3] describes a Privilege Management Infrastructure that uses attribute certificates to store a user's

privileges/credentials. X.509 PMIs can provide some of the components for trust management systems.

ISO/IEC 10181-3 [5] describes an access control framework for use by open systems, and separates the authorisation gatekeeping function into two components, the application dependent Access Control Enforcement Function (AEF) and the application independent Access Control Decision Function (ADF).

The EC PERMIS project has built a trust management system for use by Internet applications that is based on an X.509 PMI and the above frameworks. The access control decision function (ADF) is written in Java, whilst the authorisation policy is written in XML. The policy and the attribute certificates are stored in LDAP directories so that they can be accessed via the Internet. This allows the administration of privileges to be widely distributed over the Internet, and for authorisation decisions to be delegated to external organisations.

This InternetDraft/RFC is one of a set of three documents. It describes the PERMIS PMI, and shows how it relates to and is consistent with the prior work in RFCs 2904 and 2704. The other two documents describe the XML policy DTD [7], and the Java API [8] in detail.

2. A brief introduction to X.509 PMIs.

In order to control access to a resource, both authentication and authorization are needed. Early versions of the ITU-T X.509 standard [3] have concentrated on standardizing strong authentication techniques, based on digital signatures, public key certificates, and Public Key Infrastructures (PKIs). 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.

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 signs an attribute certificate is called an Attribute Authority (AA). The root of trust of a PKI is sometimes called the root CA or the trust anchor, whilst the root of trust of a 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 trusted 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).

3. Trust Management

RFC 2704 says that a trust-management system has five basic components:

i) A language for describing `actions', which are operations with security consequences that are to be controlled by the system.
ii) A mechanism for identifying `principals', which are entities that can be authorized to perform actions.
iii) A language for specifying application `policies', which govern the actions that principals are authorized to perform.
iv) A language for specifying `credentials', which allow principals to delegate authorization to other principals.
v) A `compliance checker', which provides a service to applications for determining how an action requested by principals should be handled, given a policy and a set of credentials.

X.509 attribute certificates specify mechanisms for ii) and iv). Principals are the holders of ACs and can be identified by their X.500/LDAP distinguished names [6][9] or by reference to their public key certificates (issuer name and serial number). Credentials are specified as X.500 attributes [6], which comprise an attribute type and value. The PERMIS policy (iii) and action (i) language have been specified in XML, and the DTD for these is described in [7]. The compliance checker (v) is actually the same as the ADF of ISO/IEC 10181-3. The PERMIS compliance checker is written in Java, and the API to this is briefly described in section 7 and more fully in [8].

4. Authorisation Frameworks

RFC 2904 describes the different entities in an authorisation infrastructure, these being:
i) the user who wants to access a resource
ii) the user's home organisation (UHO) that authorises the user to access the resource
iii) the service provider of the resource, comprising
iii A) the resource's AAA Server which authorizes the user's service request based on an agreement with the UHO, but without specific knowledge of the individual user
iii B) the resource's Service Equipment that provides the service itself. This might, for example, be a print server in the Internet Printing service.

RFC 2904 further describes the interactions between the entities when
a) the UHO and service provider are in the same domain, and
b) the user is roaming, and the UHO and service provider are in different domains.

ISO/IEC 10181-3 further breaks down the AAA server into the application dependent Access Control Enforcement Function (AEF) and application independent Access Control Decision Function (ADF). As previously stated, the ADF is the compliance checker of a trust management system.

5. The PERMIS Privilege Management Infrastructure

The PERMIS PMI is described primarily using terminology from the AAA Authorisation Framework. The UHO is the entity that allocates privileges to users, in the form of digitally signed X.509 attribute certificates. The UHO is a privilege allocator in PERMIS terminology and an SOA in X.509 terminology. Once created, the ACs may either be

stored in an LDAP directory local to the UHO (the pull model), or given to the user for him to use as required (the push model). If the UHO supports delegation, then there may be subordinate AAs within the UHO, who are also authorised to issue ACs to users. The privileges, or authorisations within the ACs, are allocated in the form of X.500 attributes, comprising an attribute type and value. As PERMIS has implemented a role based access control infrastructure, the attributes are considered to be roles. For any given attribute type, for example, "employment role", the role values may form a role hierarchy, for example: director > departmental manager > project leader > team leader > employee. Then the privileges given to the subordinate roles are automatically inherited by the superior roles. The authorisation policy dictates which roles have which access privileges.

The concept of a role within PERMIS has been generalised to cover any title, certificate, membership or other role that can be given to a user. So for example, a university degree is considered to be a role (where the UHO is the university, the attribute type is "degree", and the attribute value is the degree classification); an ISO 9000 certificate can be a role (where the UHO is the certification body assessing the organisational unit, the user is the organisational unit that was assessed, the attribute type is "ISO certified" and the attribute value is the number of the ISO standard against which the organisational unit was assessed); membership of the Internet Society can be a role (where the UHO is the Internet Society, the attribute type is "membership number" and the attribute value is the membership number). In general, Service Providers will determine which roles are required for access to their Service Equipment, and will authorise UHOs to allocate them. The act of authorisation takes place by some inter-organisational contract, and is technically enabled when details of the UHO and its roles are written into the role assignment policy (see later) that controls access to the service.

One can immediately see that PERMIS allows multiple UHOs to be associated with a single service. This is because users may be given different roles by different UHOs. (For example, I have a frequent flyer card allocated by my favourite airline, another one from the hotel chain I use, and a credit card from Visa. I might need all these when making a hotel booking across the Internet. Once all of these exist electronically as X.509 ACs, I should no longer need to carry the plastic cards around with me.)

When the user tries to access a service, his request is either intercepted by the AAA server (as in Figure 1), or relayed to it by the service equipment. Either way, it is the AAA Server that makes the authorisation decision (and the authentication and accounting decisions as well, but these are not discussed further in this document).

The AAA Server is decomposed in Figure 1 into its constituent parts according to the ISO 10181-3 framework. The AEF is passed the user's request, and this is first authenticated by the authentication service. If authentication is successful, the user's X.500/LDAP distinguished name is passed to the ADF via the PERMIS Java API. The PERMIS Java API is briefly described in section 6 below.

The PERMIS model is the same for the single domain case and the roaming user case. The only difference is that in the single domain case, the ADF will only retrieve ACs from the local LDAP server, whereas in the roaming user case, the ADF will retrieve ACs from both

the local and remote LDAP servers. The list of LDAP servers is passed to the ADF at API construction time.

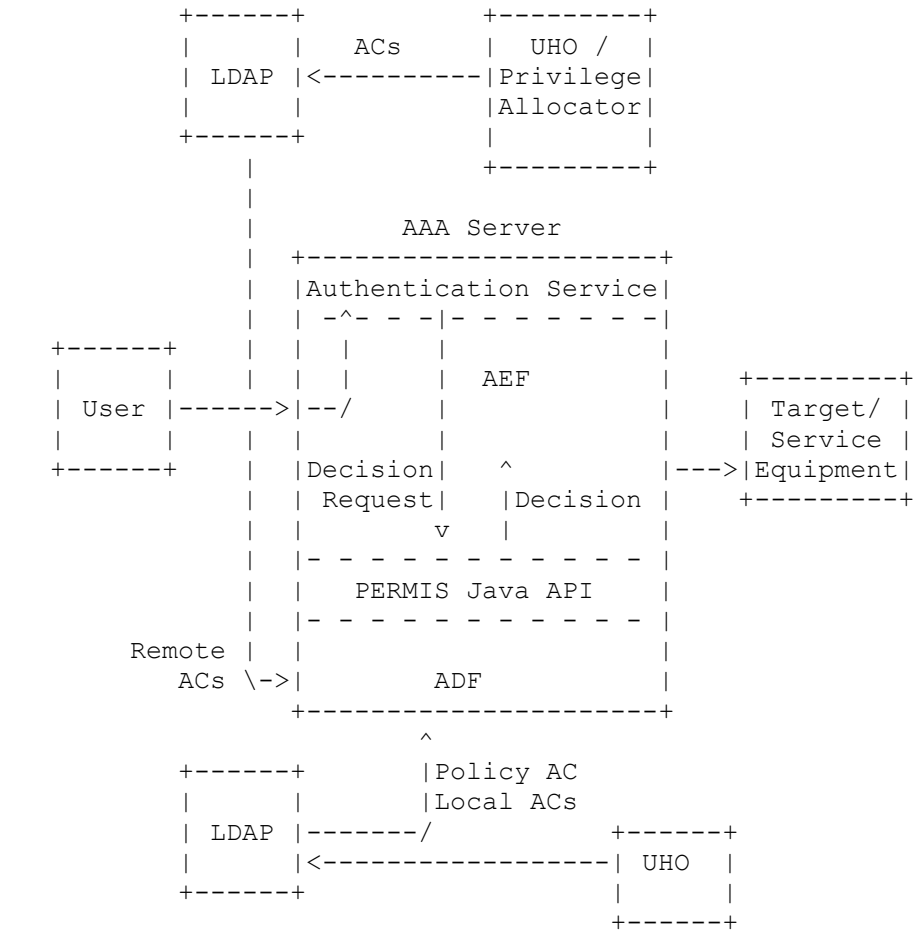

Figure 1. The PERMIS Infrastructure

6. The PERMIS Authorisation 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 discretionary access control (DAC) 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 local SOA/UHO) to specify the authorization policy for the whole domain, and all targets will then be controlled by the same set of rules.

The PERMIS authorisation policy uses the hierarchical RBAC model for specifying authorizations. RBAC has the advantage of scalability over DAC, and can easily handle large numbers of users, which is especially important for Internet applications, as there are typically far fewer roles than users.

The PERMIS project decided to use XML as the 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.

The Data Type Definition (DTD) for the PERMIS X.500 PMI RBAC Policy comprises the following components:

- SubjectPolicy – this specifies the subject domains i.e. only users from a specified subject domain may be authorized to access resources covered by this policy.
- RoleHierarchyPolicy – this specifies the different roles recognised by this policy 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. The first SOA in the list is the one for the local domain, and subsequent SOAs are from trusted remote domains. This is actually a form of cross certification of remote authorisation domains.
- 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.
- 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 [7].

7. The PERMIS Java API

The PERMIS Java 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 local UHO (the SOA that is the root of trust for authorisation), the Object Identifier of the authorisation policy, and a list of LDAP URIs from where the ADF can retrieve the policy AC and role ACs. The first URI in the list must be that of the local LDAP server. The policy AC is always retrieved from the first URI in the list, from the entry with the LDAP DN [9] of the SOA. 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.

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. If the user authenticated by digitally signing the opening message, verification of the signature will yield the user's LDAP DN from the user's PKC. If the user authenticated by another method, then the AEF will need to map the user's authenticated name into an LDAP DN. The ADF uses this DN to retrieve all the role ACs of the user from the list of LDAP URIs passed at initialisation 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, and returned to the AEF as a subject object. 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 subject object, 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 authorisation 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 authorisation policy and be ready to make access control decisions again.

A full description of the PERMIS Java API can be found in [8].

8. Acknowledgements

The author would like to thank the European Union for partially funding this project under the Information Society Initiative For Standardization (ISIS) program Contract Number 503163.

9. Copyright

Copyright (C) The Internet Society (date). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be

revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

10. References

[1] S. Bradner. "The Internet Standards Process -- Revision 3", RFC 2026, October 1996.
[2] Blaze, M., Feigenbaum, J., Ioannidis, J. "The KeyNote Trust-Management System Version 2", RFC 2704, September 1999
[3] ITU-T Rec. X.509(2001) The Directory: Authentication Framework
[4] J.Vollbrecht et al. "AAA Authorization Framework", RFC 2904, August 2000
[5] ITU-T Rec X.812 (1995) | ISO/IEC 10181-3:1996 "Security Frameworks for open systems: Access control framework"
[6] ITU-T Rec. X.501(1993) The Directory: Models
[7] Chadwick, D.W., Otenko, A. "RBAC Policies in XML for X.509 Based Privilege Management" to be presented at IFIP SEC 2002, Egypt, May 2002
[8] Chadwick, D.W., Otenko, A. "The PERMIS Java Authorisation API". Internet Draft to be written.
[9] Wahl, M., Kille, S., Howes, T. "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", RFC2253, December 1997
[10] D. Chadwick, S.Legg. "Internet X.509 Public Key Infrastructure - Additional LDAP Schema for PKIs and PMIs", Internet Draft <draft-pkix-ldap-schema-01.txt>, September 2000

11. Authors Address

David Chadwick
IS Institute
University of Salford
Salford M5 4WT
England

Email: d.w.chadwick@salford.ac.uk
Tel: +44 161 295 5351