Implement role based access control with attribute certificates

Wei Zhou
Computer Science Department
University of Trier
D-54286 Trier, Germany
zhouwei48@hotmail.com

Christoph Meinel
Computer Science Department
University of Trier
D-54286 Trier, Germany
meinel@ti.uni-trier.de

Abstract

Nowadays more and more activities are performed over the Internet. But as more people are involved in the transaction circle, security and authorization control becomes one of the biggest concerns. Hence, We are motivated by the need to manage and to enforce a strong authorization mechanism in large-scale web-environment. Role based access control (RBAC) provides some flexibility to security management. Public key infrastructure (PKI) can provide a strong authentication. Privilege management infrastructure (PMI) as a new technology can provide strong authorization. In order to satisfy mentioned security requirements, we have established a role based access control infrastructure and developed a prototype that uses X.509 public key certificates (PKCs) and attribute certificates (ACs). Access control is performed by access control policies that are written in XML. Policies and roles are stored in ACs. PKCs and ACs are all stored in LDAP servers. A new solution for policy management is described. The main components of the prototype are administration tool and access control engine. The access control engine provides a service that mediates the data between the users and the resources, which is also responsible for authentication and authorization. The administration tool can create key pairs, PKCs and ACs, manage users' information, and so on.

Keywords

Role based access control, X.509, public key infrastructure, public key certificates, privilege management infrastructure, attribute certificates, authentication, authorization, XML.

1. Introduction

Nowadays more and more activities are performed over the Internet. This trend creates new business opportunities and posts new technical challenges. One of the most challenging problems in managing large networked systems is the complexity of security administration, particularly access control. The traditional access control list (ACL) can not always provide satisfied quality of security management when there are many subjects and objects. So new access control mechanisms are needed in order to cater for various applications in Internet.

Role based access control (RBAC) emerged rapidly in the 1990s as a proven technology for managing and enforcing security in large-scale enterprise wide systems [1, 2, 3]. It can provide more flexibility to security management over the traditional approach of using user and group identifiers.

Another important technology that can be used for access control is privilege management infrastructure (PMI) [10]. The main function of PMI is providing a strong authorization after the authentication has taken place. Some research and development efforts have been done in this area [5, 7, 8, 9], but these efforts are still in primary phase, and no authorization mechanism is widely accepted.

We were motivated by the need of using PKI, PMI and RBAC concepts to construct an authorization mechanism. After assessing several works that have been done in this area [5, 7, 8, 19, 20, 21, 22], we decided to adopt a model that is similar to the PERMIS [5]. The main idea of the PERMIS model is that user's roles are stored in ACs, access control decisions are driven by an authorization policy, and the authorization policy is also stored in an AC. The main difference between them is that our model supports multi-policy. A prototype has been developed.

This paper is organized as follows. Section 2 overviews RBAC and PMI technologies. Section 3 describes our approach. Section 4 compares our work to some related works. Finally, section 5 mentions some future works.

2. Main related technologies introduction

2.1. Role based access control

The central notion of RBAC is that permissions are associated with roles, and users are assigned to appropriate roles. A general RBAC model as depicted in Figure 1 was defined by Sandhu [1]. The model is based on three sets of entities called users (U), roles (R) and permissions (P). The user assignment (UA) and permission assignment (PA) relations are both many-to-many relationships. Role hierarchy (RH) in RBAC is a natural way of organizing roles to reflect the organization's lines of authority and responsibility. A senior role can inherit permissions from junior roles. A user establishes a session during which he activates some subset of roles that he is a member of. Constrains are an effective mechanism to establish higher-level organization policy, they can apply to any of the proceeding components. RBAC also supports three important security principles: least privilege, separation of duties, and data abstraction.

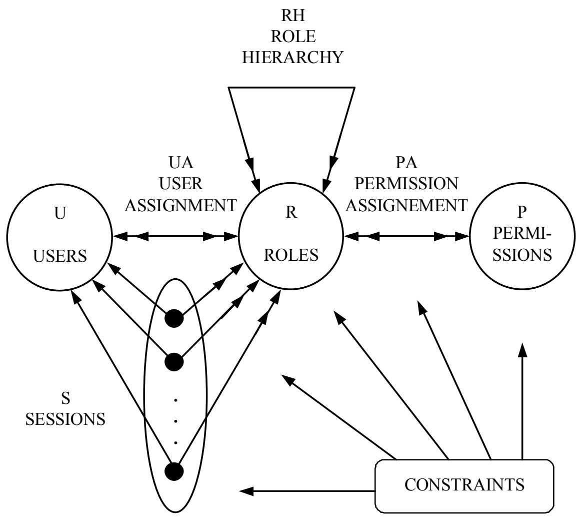


Figure 1. Basic RBAC model

With RBAC, system administrators can create roles for the various job functions in an organization, grant permissions to those roles, and then assign users to the roles on the basis of their specific job responsibilities and policy. Users can be easily reassigned from one role to another. Roles can be granted by new permissions, as new applications and systems are incorporated, and permissions can be revoked from roles as needed. Moreover, the access control policy can evolve incrementally over the system life cycle. The ability to modify policy to meet the changing needs of an organization is an important benefit of RBAC.

More information about RBAC may consult [2, 3].

2.2. Privilege management infrastructure

PMI was specified by the ITU-T and ISO/IEC [10]. The main function of PMI is providing a strong authorization after the authentication has taken place. It has a number of similarities with PKI [12]. The basic data structure in a PMI is a X.509 attribute certificate (AC) [11]. Like public key certificate (PKC) strongly binds a public key to its subject, AC strongly binds a set of attributes to its holder. PMI and PKI infrastructures are linked by information contained in the attribute and identity certificates. For example the field holder in an AC contains the serial number and issuer of a PKC. The attribute certificate, identity certificate and their relation are depicted in Figure 2.

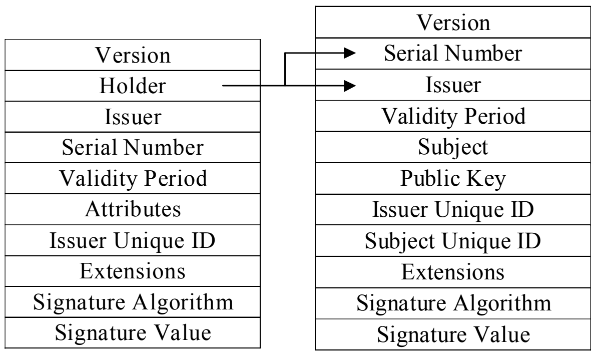


Figure 2. Relation between attribute and identity certificate

In PMI the ACs' issuer is called attribute authority (AA). ACs are digitally signed by the AA, so they are tamper-resistant. The trusted root is called source of authority (SOA). When a user's authorization permissions need to be revoked, AA will issue an attribute certificate revocation list (ACRL) containing the list of ACs no long to be trusted.

There are two primary models for distribution of attribute certificates: the "push" or "pull" model. The "push" model is suitable when the client's rights should be assigned within the client's "home" domain, whereas the "pull" model is suitable when the client's rights should be assigned within the server's domain.

More information about PMI may consult [10, 11].

3. Implementation

3.1. System overview

Our access control system is designed to support RBAC using X.509 PKCs and ACs. The authentication is implemented by PKI, and the authorization is implemented by PMI. Role information is stored in role ACs. All the access control decisions are made based on authorization policies, they are written in XML and stored in policy ACs. ACs and their corresponding PKCs are all stored in LDAP servers [16]. In our prototype there are two kinds of policy: root policy and authorization policy. We use the PERMIS X.500 PMI RBAC policy [6] as our RBAC policy. Its Data Type Definition (DTD) has been published at http://www.xml.org. A policy management strategy makes it easy to add or remove a policy without influence to the software and other policies. The basic components of the access control system as depicted in Figure 3 are described as follows.

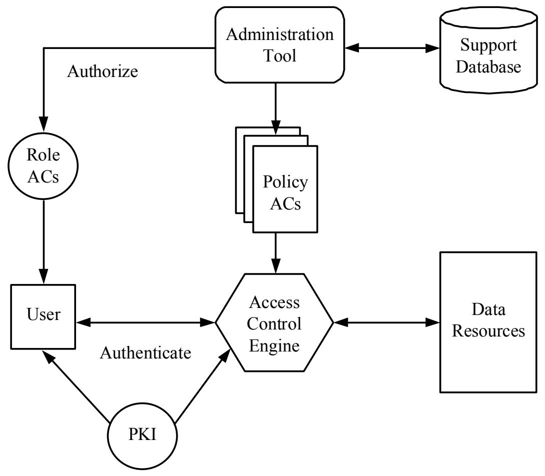


Figure 3. The overview of the access control system

- Administration Tool – is used for creating key pair, PKCs, policy and role ACs, manage users' information and so on.
- Support Database – stores the information that is used by the administration tool, e.g. the users' information [17].
- Access Control Engine – executes the functions of authentication and authorization, then accesses the target on behalf of the user.
- Access Control Policies – specify which roles have which rights on which targets. Each access control decision is made based on them.
- Data Resources – they may be web servers, data servers, file systems or other format data resources.

3.2. Administration Tool

The administration tool can complete three main functions: creating key pair and its PKC, creating policy and role ACs, managing PKCs and ACs. The key pair tool can create key pair and its self-signed PKC for CA or AA, or non self-signed PKC for normal users. When Trusted Third Party services (TTPs) are needed, users must provide their PKCs to the AA for creating their role ACs. The AC tool can create policy and role ACs. The policy AC's attribute value is gotten from a XML file. Policy AC is bound to an AA's PKC and signed by the AA. The role AC's attribute value consists of one or several role names, e.g. "Manager", "Clerk", etc. Role AC is bound to a user's PKC and signed by the AA. In our prototype we adopt AC "pull" model, so the role ACs are not given to users and does not need the ACRL. The policy and role ACs are all stored in LDAP servers. Since they have been signed by the AA, so they are tamper-resistant and no modification risk from allowing them to be stored in a publicly accessible repository. The certificate tool is used for managing PKCs and ACs, for example, imports or exports PKCs or ACs from LDAP servers. The administration tool also has some other useful functions, for example, manages key store, manages users' information, and so on.

3.3. Access control engine

The access control engine is implemented by a Java servlet [15]. It is responsible for authentication and authorization and provides a service that mediates the data between the users and the targets. Its structure is depicted in Figure 4. Our access control framework conforms to the basic principle of ISO 10181-3 Access Control Framework that is defined by the Open Group [13]. This framework separates authentication from authorization, and comprises four components:

Initiator (e.g. a browser), Target (e.g. a database), Access Control Enforcement Function (AEF) and Access Control Decision Function (ADF). The initiator submits access request that specifies an operation to be performed on a target. The AEF mediates access requests, it submits decision request to ADF through the authorization API (aznAPI) [14], a decision request asks whether a particular access request should be granted or denied. AEF uses the aznAPI to presents Access Control Information (ACI) that is a set of the information that might be relevant to an access control decision to ADF. ADF decides whether access requests should be granted or denied, it makes access control decisions based on access control policies and Access Control Decision Information (ADI) that describes security-relevant properties of the initiator, the target, the access request, and the system and its environment. The aznAPI is responsible for deriving ADI from the ACI and presenting the ADI to the ADF. At last AEF enforces access control decisions made by ADF.

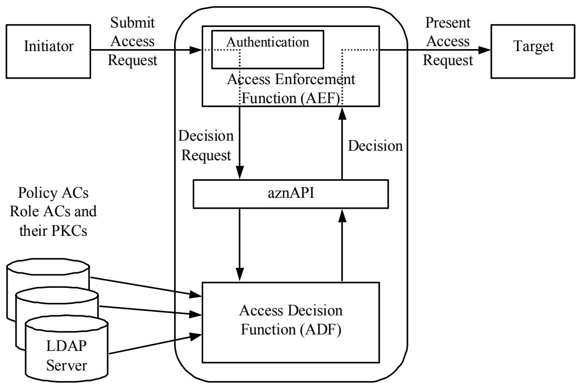


Figure 4. The access control engine

In our model, a user accesses resources via an access control engine. First he uses his private key to sign some information, then uploads this signature file with a browser. On the server side, the AEF yields his LDAP distinguished name (DN) from the file, and gets the user's PKC from a LDAP server using the LDAP DN. The AEF authenticates the user through verifying his signature with his PKC, if he is a valid user, the AEF passes his LDAP DN to ADF through calling aznAPI in order to get the user's roles, otherwise refuses his connection request. The ADF uses the LDAP DN to retrieve the user's role ACs and checks whether they are issued by a trusted AAs and still valid, the invalid ACs are discarded. From the valid ACs the user's roles and their validity time will be extracted. These roles and their validity time, together with refresh time and session time are returned to the AEF. The AEF keeps the information that comes from user's signature, PKC and ADF, then establishes a session for him. If the refresh times out, AEF recall the aznAPI to get the user's roles again. If the session times out, the AEF either closes the user's connection or informs the user to reconnect. Refresh time provides a compromised solution between reading ACs every time and never reading ACs in the whole session time. Too often reads user's role ACs will lead to obvious inefficiency, whereas too rarely renews user's role list will lead to unexpected roles still in action when they have been revoked, in the same time, it can also lead to new rights inactivity when they have been assigned. The session time can avoid that a user keeps the connection open for an infinite amount time until his ACs expires. How long the refresh time and the session time should be set depend on the application requirements. They have to be configured into the system at start up.

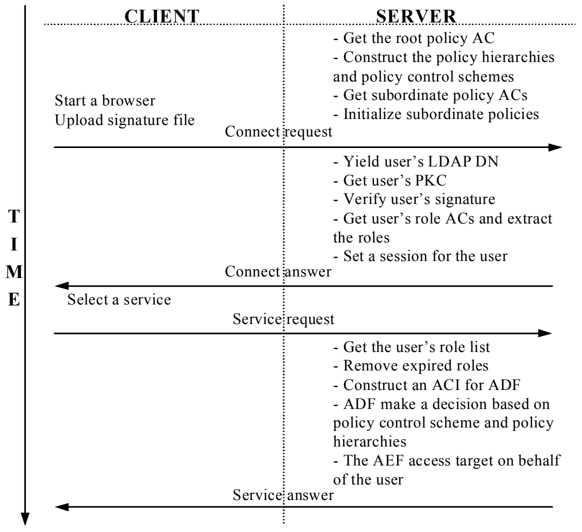

CLIENT
SERVER
- Get the root policy AC
- Construct the policy hierarchies and policy control schemes
- Get subordinate policy ACs
- Initialize subordinate policies
Start a browser
Upload signature file
Connect request
- Yield user's LDAP DN
- Get user's PKC
- Verify user's signature
- Get user's role ACs and extract the roles
- Set a session for the user
TIME
Connect answer
Select a service
Service request
- Get the user's role list
- Remove expired roles
- Construct an ACI for ADF
- ADF make a decision based on policy control scheme and policy hierarchies
- The AEF access target on behalf of the user
Service answer

Figure 5. Authentication, authorization and service sequence

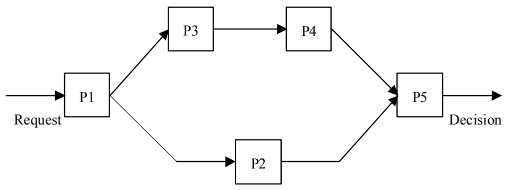
Secondly, after passing the authentication, the user can select a service and submit the request to the access control engine. The AEF constructs ACI using the access request information, e.g. action information and target information, and roles information that has been kept for the user in the connection request phase. But before constructing the role information, the AEF will check every role's validity time, and remove the expired roles from the role list. After that the ACI is submitted to ADF through the aznAPI. The ADF get the roles, action and target information from the ACI, then checks if the roles are allowed to perform the action to the target according to the corresponding access control policies. If the action is allowed, "PERMIT" is return, otherwise "NO PERMIT" is returned. In the case of permission, the AEF will access the target on behalf of the user and return the result to the user, if no permission the user's request is refused. The authentication, authorization, and the service sequence are depicted in Figure 5.

3.4. Policy management

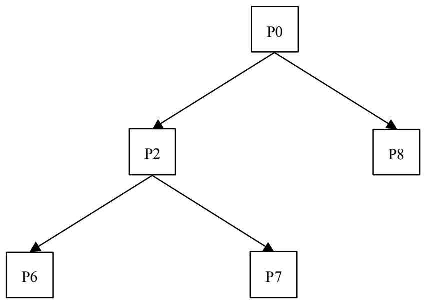
In big organizations, for example the government or enterprises, it is difficult to assume that all the resources are controlled under one authorization policy. Otherwise it means that the policy is either too complex to maintain, or too simple to provide fine-grained access control. We bring forward a new solution for policy management to solve this problem. In our model, every domain comprises a root policy and some subordinate policies. The root policy specifies which policies are used in a domain, where to find them, how to verify them and their validity time. It also specifies the policy hierarchies and the policy control schemes that describe which access request should be checked by which policies. Subordinate policies specify which roles have which rights on which targets, they can be centralized or distributed. All policies are stored in ACs, thus guaranteeing their integrity. The root policy is stored in a self signed AC. In a domain there maybe are lots of self-signed policy ACs, but only one keeps the root policy, the information about this AC is stored in a configuration file that must be kept in a security place. When the access control engine starts up, the system gets the information about the root policy AC from the configuration file, and reads the root policy AC in. If the AC passing the audit process, the root policy is extracted, then the policy control schemes and the policy hierarchies are constructed. According to the information in the root policy, the system reads in other policy ACs, verifies them and extracts the policies from them, then initializes these policies.

An example about policy control scheme and policy hierarchy is depicted in Figure 6. In the policy control scheme there comprises five policies (P1, P2, P3, P4 and P5). The meaning of the scheme is that a request must either satisfy P1, P3, P4 and P5, or satisfy P1, P2 and P5. In the policy hierarchy there comprises five policies (P0, P2, P6, P7 and P8). The P0 is root policy in the tree, and the others are subordinate policies that inherit from the root policy and superior policies. In the runtime, when P2 is checked in the policy control scheme and the result is "PERMIT", the system finds it is included in a policy tree and inherits from P0, then P0 will be checked and the result will be the final result of P2 in the scheme.

A refresh time is set for the root policy, if it times out, the system reads the root policy and initializes every thing again. The refresh time can assure to renew the access control information in a tolerable time. How long the refresh time should be set depends on the application requirements. It has to be configured into the system at start up



(a) Policy control scheme



(b) Policy hierarchy

Figure 6. Examples of policy control scheme and policy hierarchy

Each policy control scheme is set a validity time, if it is expired, this scheme will not be used, and all the access requests relate to this scheme will be refused. Similarly, each policy is also set a validity time. When a policy in a policy scheme or a policy hierarchy is expired, how to deal with this situation depends on which kind of tag has been added to a policy, i.e. "critical" or "non-critical". In the case of "critical" checking this policy must return "NO PERMIT", otherwise ignore this policy and return "PERMIT".

In conclusion, in our opinion the concepts of policy control scheme and policy hierarchy can provide more flexibility to policy management and accommodate the complicated application.

4. Related works

There are two important analogous systems, namely the PERMIS [5] and Akenti [7].

There are two major differences between PERMIS and our model. The first is that in PERMIS one policy governs all aspects of access to the targets in the local domain, so it does not support the policy hierarchies. On the contrary, in our model every domain has a root policy that specifies which policies are used, how to get them and their validity periods. These subordinate policies can be distributed and hierarchical, several policies can form a policy control scheme and in a domain can have lots of policy control schemes. The second is that PERMIS is authentication agnostic and leaves it up to the application to determine what type of authentication to use, whereas our model requires the user to be PKI enabled and to present an X.509 PKC at authentication time.

There are three major differences between Akenti and our model. The first is that in Akenti the ACs are in a non-standard format. The second is that in Akenti the access control essentially is a classical access control list (ACL) model, whereas our model has implemented role based access control. The third is that their policy hierarchies are not specified in a secure way, because these policies must be stored in secure directory hierarchies, and the directory hierarchy determines the policy hierarchy [18]. On the contrary, we specify the policy hierarchies in the root policy.

5. Future works

Our future research work includes two parts, one is that improves our model so that it can be used in distributed environment, the other is that extends RBAC in order to support the dynamic aspects of many modern information systems like the workflow.

References

[1] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman, Role based access control models, IEEE Computer, 29 February 1996.
[2] David F. Ferraiolo, R.S. Sandhu, Serban Gavrila, D. Richard Kuhn and Ramaswamy Chandramouli, Proposed NIST Standard for Role-Based Access Control, ACM Transactions on Information and Systems Security (TISSEC), Volume 4, Number 3, August 2001.
[3] R.S. Sandhu, Bhamidipati and Qamar Munawer The ARBAC97 Model for Role-Based Administration of Roles, ACM Transactions on Information and Systems Security (TISSEC), Volume 2, Number 1, Februrary 1999.
[4] J.S. Park, R. Sandhu, G. Ahn, Role-based access control on the web, ACM Transactions on Information and System Security, Volume 4, Number 1, February 2001.
[5] D.W. Chadwick, A. Otenko, The PERMIS X.509 role based privilege management infrastructure, Future Generation Computer Systems, Volume 19, Issue 2, February 2003, Pages 277-289.
[6] D.W. Chadwick, A. Otenko, RBAC Policies in XML for X.509 Based Privilege Management, to be presented at SEC 2002, Egypt, May 2002.
[7] M. Thompson, A. Essiari, S. Mudumbai, Certificate-based Authorization Policy in a PKI Environment, Submitted to a special issue of ACM Transactions on Information and System Security, August 2003.
[8] B. Blobel, P. Hoepner, R. Joop, S. Karnouskos, G. Kleinhuis and G. Stassinopoulos, Using a privilege management infrastructure for secure web-based e-health applications, Computer Communications, Volume 26, Issue 16, 15 October 2003, Pages 1863-1872.
[9] Javier Lopez, Antonio Mana, Juan J. Ortega, Jose M. Troya and Manemma I. Yague, Integrating PMI services in CORBA applications, Computer Standards & Interfaces, Volume 25, Issue 4, August 2003, Pages 391-409.
[10] ITU-T Rec. X.509 ISO/IEC 9594-8, The Directory: Public-key and Attribute Certificate Frameworks, May 3, 2001.
[11] S. Farrell, R. Housley, An Internet Attribute Certificate Profile for Authorization, Internet-draft April 2002, http://www.ietf.org/rfc/rfc3281.txt.
[12] R. Housley, W. Ford, W. Polk, D. Solo, Internet X.509 Public Key Infrastructure Certificate and CRL Profile, http://www.ietf.org/rfc/rfc2459.txt.
[13] ITU-T Rec. X.812(1995)|ISO/IEC 10181-3:1996, Security frameworks in open systems: Access control framework.
[14] The Open Group, Authorization (AZN) API, ISBN: 1-85912-266-3, January 2000.
[15] Jakarta Tomcat servlet container, http://jakarta.apache.org/tomcat/.
[16] OpenLDAP, the Open Source Lightweight Directory Access Protocol (LDAP), http://www.openldap.org/.
[17] MySQL, the Open Source SQL database, http://www.mysql.com/.
[18] A. Otenko, D.W. Chadwick, A Comparsion of the Akenti and PERMIS Authorization Infrastructures, http://sec.isi.salford.ac.uk/.
[19] M. Blaze, J. Feigenbaum, J. Ioannidis, The KeyNote Trust-Management System, Version 2, RFC 2074, September 1999.
[20] L. Pearlman, V. Welch, I. Foster, K. Kesselman and S. Tuecke, A Community Authorization Service for Group Collaboration, IEEE Workshop on Policies for Distributed Systems and Networks (2002).

[21] The Virtual Organization Membership Service (VOMS), http://grid-auth.infn.it/.
[22] M. Erdos, S. Cantor, Shibboleth-Architecture DRAFT v05, http://shibboleth.internet2.edu/.