

Access Control in a Distributed Object Environment Using XML and Roles

Jason Crampton Hemanth Khambhammettu

Information Security Group, Royal Holloway, University of London, United Kingdom,
{Jason.Crampton,H.Khambhammettu}@rhul.ac.uk

9th December 2003

Abstract

We discuss the design of an integrated security architecture for authorization and authentication in a distributed object environment. Our architecture will have four main components: an *authentication engine*, an *interface*, a *session manager* and an *authorization engine*. The core component of our model is the session manager, which issues XML-based *session certificates* to authenticated users. A session certificate will be used by the authorization engine to establish the legitimacy of an access request by a user. We will also describe how the architecture supports dynamic revocation of session certificates and delegation.

Keywords Roles, XML, distributed object environment, session certificate

Computing Review Categories D.4.6, K.6.5

1 Introduction

There is considerable interest in the commercial and research communities in developing a scalable and practical model for access control in a distributed object environment. Such environments pose problems that traditional access control models are ill-equipped to address. Several new schemes have been proposed for access control in a distributed object environment, but they may not scale well or may not be flexible enough to be easily integrated into other existing security architectures. We are currently developing an integrated security architecture for authorization and authentication to meet such demands in a distributed environment.

Role-based access control is now widely accepted as an alternative to techniques based on the protection matrix such as access control lists. It is also widely accepted as the basis for authorization in distributed systems containing large user groups because the use of roles can considerably simplify access control administration [3]. The basic idea of associating a set of privileges or permissions with a named role and assigning that role to users is well established and is deployed in several commercial computer systems and applications (including Windows 2000, Oracle 9, Java 1.2).

Our architecture will have four main components: an *authentication engine*, an *interface*, a *session manager* and an *authorization engine*. The architecture will implement a role-based access control model incorporating a role hierarchy, a user-role assignment relation and a permission-role assignment relation. The role hierarchy is determined by the Hasse diagram of a partially ordered set of roles $\langle R, \leq \rangle$.¹ The user-role assignment relation UA is a binary relation that associates users with roles. Similarly, the permission-role assignment relation PA is a binary relation that

¹In other words, the role hierarchy is the graph of the reflexive, transitive reduction of a partial order relation \leq defined on a set of roles R .

associates permissions with roles. Hence, an access control decision is made by determining which roles, and hence which permissions, a user is entitled to use.

The core component of our model is the session manager, which issues XML-based *session certificates* to authenticated users. The session certificate will be used by the authorization engine to establish the legitimacy of an access request by a user. The session certificate is a kind of attribute certificate written in XML and is similar to a capability card [7].

In the remainder of the introduction we discuss related work and the motivation behind our architecture. In Sections 2 and 3 we describe the components and operation of our architecture. In conclusion we briefly summarize the contributions of the paper and identify potential topics for future research.

1.1 Related Work

SESAME [1] is a role-based security architecture based on Kerberos and the ATHENA project at MIT [5]. SESAME makes use of privilege attributes certificates (PACs), a specific form of access control certificate that conforms to ECMA and ISO/ITU-T standards, for implementing authorization.

In the late 1990s, Fujitsu Laboratories developed the idea of a *capability card*, a kind of attribute certificate specified using XML [7]. A capability card can be used to grant a user access privileges to resources and also supports a variety of delegation mechanisms.

In this paper we synthesize the SESAME architecture and the concept of a capability card to develop a new role-based security architecture that employs XML-based session certificates. These certificates include a set of roles, which are used by the access control decision point to determine whether a user's access request can be granted based on the assignment of capabilities to roles. Our contribution is the development of a practical model that separates rights-granting mechanisms from decision-making mechanisms, and which uses XML and role based access control to represent authorization related information.

1.2 Objectives and Design Decisions

In this section we discuss what we believe to be important criteria when developing an integrated security architecture and the design decisions that have led to the satisfaction of those criteria.

- *Independence of authentication and authorization schemes* We believe that authorization schemes should be independent of the authentication scheme used. We have therefore separated the responsibility for authentication and authorization.
- *Avoid third-party trust* Authorization scenarios have very different design features compared to authentication scenarios [6]. Third-party trust in authorization scenarios may not be ideal as the access to resources owned by a user/system can not be decided based on the level of trust placed on a third-party. Hence we have avoided the use of any trusted authority in our architecture.
- *Inter-operability and extensibility* In an increasingly interconnected world, it should be possible to integrate the respective security architectures when applications are required to interact. Furthermore, authorization schemes should be extensible in order to meet the requirements of diverse and developing applications [7].

XML offers data portability and reusability across different platforms and devices. It is also flexible and extensible, allowing new elements to be added without breaking an existing document structure. Hence XML satisfies our requirements and is used to represent the authorization and authentication data structures in our architecture. An XML-based session certificate can be easily integrated in to other security architectures and extended to add any further information [7].

- *Scalability* With the exponential growth of Internet usage and e-commerce in recent years, distributed environments are becoming vitally important in the delivery of electronic services. Such services typically have large numbers of users and complex business logic. Therefore, distributed environments and any associated security systems need to be highly scalable in order to provide fast and efficient services to its users.

The modular design of our architecture distributes the load across various components of the system and should therefore provide an extensible and scalable framework for security.

- *Avoid bottlenecks at access control decision point* A large number of users in a distributed environment implies a large number of concurrent access requests initiated by them. Hence it is possible that access control decision points may act as bottlenecks thereby creating a potential threat to the system from denial-of-service (DoS) attacks.

We avoid the problem of bottlenecks at the access control decision point by filtering out malicious and malformed access requests. In our system, the interface checks every access request for a genuine binding between the requestor and his session certificate, and only allows genuine access requests to reach the authorization engine (access control decision point).

- *Separation of the rights-granting mechanism from the decision-making mechanism* Most existing authorization models propose a coordinated or combined mechanism for granting access rights and deciding on the access requests. The rights granting mechanism and decision-making mechanism are two very different operations and need not be coordinated or combined together [4]. We have implemented this idea in our system and is explained in Sections 2.2 and 2.3.
- *Support for audit requirements* An audit trail is an essential part of any security architecture in order to support after-the-fact investigation of security breaches. In our system every access request is bound to the requestor's identity, thereby providing the basis for a strong security audit feature.

2 System Architecture

Our architecture has four main components: an *authentication engine*, an *interface*, a *session manager* and an *authorization engine*. The core component of our model is the session manager, which issues XML-based *session certificates* to authenticated users. An overview of the architecture and its operational behaviour is shown in Figure 1.

The system also maintains the following information: a set of users U , a set of roles R , a role hierarchy RH , a user-role assignment relation UA and a permission-role assignment relation PA . Broadly speaking, U is associated with the authentication engine, UA and RH with the session manager, and PA with the authorization engine.

2.1 Authentication Engine

The initial point of contact for a user is the authentication engine, to which the user will submit credentials as part of some logon procedure. The authentication engine attempts to validate these credentials against a user data repository. If the logon process is successful, the authentication engine creates an *authentication token*, digitally signs it and forwards it to the interface.

2.2 Interface

The interface has two main functions: to assess the validity of authentication tokens, session certificates and access requests; and to act as an access control enforcement point. The interface handles two distinct types of requests: session certificate requests, which are forwarded to the

session manager in order to create a session certificate; and access requests, which are forwarded to the authorization engine in order to make an access control decision.

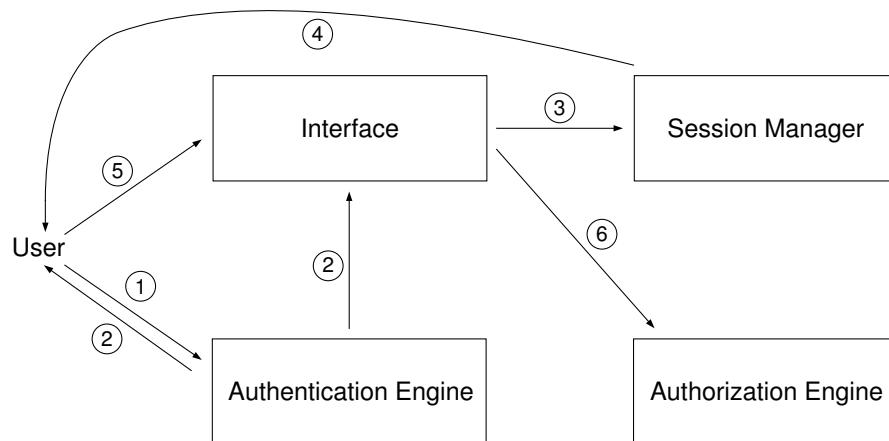
2.3 Session Manager

The session manager issues XML-based *session certificates* to authenticated users in response to session certificate requests from the interface. The session manager only accepts requests from the interface.

Essentially, a session certificate identifies a set of roles allocated to a user for the lifetime of the session. Session certificates are discussed in more detail in Section 3.3. Indirectly, a session certificate grants a user a number of permissions and hence is similar to a capability card [7] or a privilege attribute certificate [1].

2.4 Authorization Engine

The authorization engine is an access control decision point and only responds to access requests received from the interface. The session certificate will be used by the authorization engine to establish the legitimacy of an access request by a user. Decisions are made by the authorization engine on the basis of the role(s) contained in the session certificate and the permission-role assignment relation.



- 1 User attempts to authenticate to Authentication Engine
- 2 Authentication Engine generates user's public/private key pair and authentication token: returns private key to user; forwards public key and authentication token to Interface
- 3 Interface forwards authentication token to Session Manager
- 4 Session Manager encrypts session certificate and sends to user
- 5 User sends session certificate and digitally signed access request to Interface
- 6 Interface verifies the authenticity of access request and forwards to Authorization Engine

Figure 1: Overview of system architecture and operation

3 System Operation

3.1 Authentication

Users will be expected to supply their credentials as part of some logon procedure. We anticipate that authentication will be performed using a variety of existing hardware and software mechanisms.

If the user is successfully authenticated to the system, the authentication engine generates a public/private key pair and an authentication token. The key pair will be generated using ID-based public-key cryptography (ID-PKC) [2, 9]. ID-PKC provides several advantages over traditional PKC [2, 8]; we find it particularly useful because it does not rely on notions of third party trust. The private key is sent to the user over a secure communications channel by the authentication engine.² The public key of the key pair is included within the authentication token.

3.1.1 Authentication tokens

The authentication token is an XML document that conforms to a schema defined by the interface. In addition to the public key of the user, the authentication token contains information about the identity, location and trustworthiness of the user. (The latter is used to generate delegation information within the session certificate.) The XML schema for an authentication token is shown in the appendix.

3.2 Session Certificate Requests

The authentication engine signs the authentication token and a hash of the token and forwards it to the interface. This authentication token is essentially a request to create a session certificate for the authenticated user. The interface attempts to verify the hash and digital signature of authentication token. If the digital signature and hash value are correct, the interface forwards the decrypted authentication token to the session manager, requesting the session manager to issue a session certificate for the user. If either the signature or the hash are incorrect, the authentication token is ignored.

3.3 Session Management

The session manager receives an authentication token from the interface, which will contain the public key of the user. The session manager consults the user-role assignment relation and any relevant authorization policy statements and decides on the role(s) that the user may activate in the session. (Authorization policy statements may refer to separation of duty requirements and limitations on role activation given the location and trustworthiness of the user's machine. Consideration of such policy details is beyond the scope of this paper.)

The session manager returns a session certificate to the user that is hashed and encrypted with the public key of the user. Hence the user can be confident that he is communicating with the intended service if he can confirm the hash value and decrypt the session certificate. In other words, our architecture implicitly provides mutual authentication between the user and the system.

3.3.1 Session certificates

A session certificate includes information about authentication, authorization, encryption techniques and delegation. The XML schema for an authentication token is shown in the appendix. The authentication information is generated from the authentication token; the authorization data is supplied by the session manager. The encryption data is used to determine the method under which the certificate can be encrypted, signed and hashed. We discuss how delegation data is used in Section 3.6.

3.3.2 Dynamic revocation of session certificates

A session certificate is a static binding of a user identity to a set of roles. Hence, changes to the user-role assignment relation and to the structure of the hierarchy during the lifetime of a session certificate may affect the validity of a session certificate. In the simplest case, for example, a user u

²Although at this stage we do not propose any particular protocol for the exchange of the private key, we anticipate that a Kerberos-like protocol will be used.

could have been issued with a session certificate containing a role r and could have his assignment to a role r revoked. Any subsequent request by u to use a permission p assigned to r should be denied by the system. In other words, the authorization engine must return a negative decision to the interface in response to such a request.

Our solution is to update the session certificate when such a request is made. Specifically, the session manager is aware of changes to the user-role assignment relation and the role hierarchy and can hence compute the users that are affected by any such changes. The session manager then sends a revised session certificate for each affected user to the interface. When the interface receives an access request from an affected user, the revised session certificate replaces the original certificate in the access request. The access request is then forwarded to the authorization engine as before. The interface also returns the new session certificate to the user encrypted with his public key.

3.3.3 Session timeout

Session certificates are associated with a timeout period that specifies their lifetime period. Hence, the interface may receive access requests accompanied by an obsolete certificate. In this case, the interface initiates a request to the session manager for a revised session certificate with an extended lifetime. The revised certificate is returned to the interface and the access request proceeds.

3.4 Access Requests

Every access request is digitally signed by the user and is accompanied by the user's session certificate. All access requests to the authorization engine are mediated by the interface, which acts as an access control enforcement point, thereby ensuring that the authorization engine only considers genuine access requests. Specifically, the interface retrieves the public-key of the user from the session certificate and only forwards the access request if it can verify the signature of the access request.

3.5 Authorization

The session certificate is the basis for all access decisions in response to access requests. Decisions are made by the authorization engine on the basis of the role(s) contained in the session certificate and the nature of the access request. Specifically, a request to user permission p will be granted if there exists a role r contained in the session certificate such that $(p, r) \in PA$. The authorization engine returns a decision to the interface, which is enforced by the interface.

3.6 Delegation

The architecture allows the users/servers to delegate their privileges to other users/servers whom they trust. The ability to delegate their privileges is determined by the authentication engine and is defined in the authentication token. The session certificate includes a delegation element, which determines whether the certificate can be delegated and constrains the number of delegation certificates that can be created.

A user delegates his privileges by creating a delegation certificate that conforms to an XML schema as defined by the interface. The user can only create a delegation certificate if the `<width>` and `<depth>` elements contained in his session certificate are greater than 0. The width determines the maximum number of delegation certificates that the session certificate can create. The depth determines whether a delegated user can further delegate any session certificates he acquires.

The user then forwards the delegation certificate and information about the delegated user to the authentication engine as part of a "proxy" authentication, which subsequently establishes a public/private key pair and a session certificate for the delegated user. Delegated users are required to sign access requests with the private key supplied to them. The details of this scheme are beyond the scope of this paper.

4 Conclusions and Future Work

We have developed a security architecture that is independent of authentication schemes and third party trust frameworks, extensible, employs a modular design, incorporates division of labour, and separates the mechanisms for granting access rights and assessing access requests. The latter feature will make a significant contribution to scalability and manageability of the system. The proposed authorization scheme also provides a framework for incorporating delegation mechanisms and audit facilities.

The ability for a user to delegate privileges to other users is becoming increasingly important in computer security. We have included support for delegation in our session certificate but a full treatment of delegation is beyond the scope of this paper. Delegation mechanisms and the development of a complete Java-based API for the interaction amongst the various components of the system will be the immediate priorities in our future work.

Acknowledgement We are grateful to Geraint Price and Kenny Paterson for their helpful comments on ID-PKC.

References

- [1] Paul Ashley. Authorization for a large heterogeneous multi-domain system. In *Proceedings of Australian Unix and Open Systems Group National Conference*, pages 159–169, 1997.
- [2] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In J. Kilian, editor, *Advances in Cryptology – CRYPTO 2001, 21st Annual International Cryptology Conference*, volume 2139 of *LNCS*, pages 213–229. Springer-Verlag, 2001.
- [3] V. Gligor. Characteristics of role-based access control. In *Proceedings of First ACM Workshop on Role-Based Access Control*, pages II9–II14, Gaithersburg, Virginia, 1995.
- [4] A.H. Karp and K. Smathers. Three design patterns for secure distributed systems. *Information Security Bulletin*, pages 49–54, 2003.
- [5] B.C. Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications*, 32(9):33–38, 1994.
- [6] Z. Nochta, P. Ebinger, and S. Abeck. PAMINA: A certificate based privilege management system. In *Proceedings of Network and Distributed System Security Symposium Conference*, 2002. Available at <http://philby.ucsd.edu/~bsy/ndss/2002/html/2002/papers/nochta.pdf>.
- [7] K. Otani, H. Sugano, and M. Mitsuoka. Capability card: An attribute certificate in XML. IETF Internet Draft, 1998. Available at <http://www.globecom.net/ietf/draft/draft-otani-ccard-00.html>.
- [8] K.G. Paterson. Cryptography from pairings: A snapshot of current research. *Information Security Technical Report*, 7(3):41–54, 2002.
- [9] A. Shamir. Identity-based cryptosystems and signature schemes. In G.R. Blakley and D. Chaum, editors, *Advances in Cryptology – CRYPTO '84*, volume 196 of *LNCS*, pages 47–53. Springer-Verlag, 1984.

Appendix – XML Schema

Authentication token (Token.xsd)

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="authToken" type="AuthenticationTokenType"/>
  <xs:complexType name="AuthenticationTokenType">
    <xs:sequence>
      <xs:element name="userId" type="xs:string"/>
      <xs:element name="publicKey" type="xs:string"/>
      <xs:element name="domainAddress" type="xs:string"/>
      <xs:element name="delegationFlag" type="xs:string"/>
      <xs:element name="expiresBy" type="xs:date"/>
      <xs:element name="timeStamp" type="xs:time"/>
      <xs:element name="hash" type="xs:string">
        <xs:attribute name="algorithm" type="xs:string" use="required"/>
      </xs:element>
      <xs:element name="signature" type="xs:string">
        <xs:attribute name="algorithm" type="xs:string" use="required"/>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Session certificate (Certificate.xsd)

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="certificate" type="CertificateType"/>
  <xs:complexType name="CertificateType">
    <xs:sequence>
      <xs:element name="certId" type="xs:string"/>
      <xs:element name="issuer" type="IssuerType"/>
      <xs:element name="authenticationData" type="AuthenticationDataType"/>
      <xs:element name="authorizationData" type="AuthorizationDataType">
        <xs:element name="hash" type="xs:string"/>
        <xs:attribute name="algorithm" type="xs:string" use="required"/>
      </xs:element>
      <xs:element name="signature" type="xs:string">
        <xs:attribute name="algorithm" type="xs:string" use="required"/>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="IssuerType">
    <xs:sequence>
      <xs:element name="domainAddress" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="AuthenticationDataType">
    <xs:sequence>
      <xs:element name="userId" type="xs:string"/>
      <xs:element name="publicKey" type="xs:string"/>
      <xs:element name="domainAddress" type="xs:string"/>
      <xs:element name="expiresBy" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="AuthorizationDataType">
    <xs:sequence>
      <xs:element name="role" type="xs:string"/>
      <xs:element name="policy" type="xs:string"/>
      <xs:element name="delegation" type="DelegationType"/>
      <xs:element name="expiresBy" type="xs:date"/>
      <xs:element name="timeStamp" type="xs:time"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="DelegationType">
    <xs:sequence>
      <xs:element name="delegationFlag" type="xs:string"/>
      <xs:element name="width" type="xs:integer"/>
      <xs:element name="depth" type="xs:integer"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```