Role Based Access Control and the JXTA Peer-to-Peer Framework

Amit Mathur
Symantec Corporation
Cupertino, California

Suneuy Kim
Department of Computer Science
San José State University
San José, California

Mark Stamp
Department of Computer Science
San José State University
San José, California

Abstract

Role based access control (RBAC) allows users access to resources based on their competencies and responsibilities within an organization. Typically, RBAC is provided as a security mechanism for a single host at the operating system level. This allows the operating system to ensure that authenticated users have access to resources based on their assigned roles. RBAC is more challenging in peer-to-peer (P2P) systems, due to the lack of centralized administration. In this paper we discuss RBAC over a P2P network and we present an RBAC implementation in the JXTA P2P framework. JXTA is a popular open P2P technology specification.

1 Introduction

In this paper we propose a role based access control (RBAC) mechanism applicable to a peer-to-peer (P2P) network. Before we discuss our RBAC implementation, we first provide some background material. In the next section we give a brief overview of RBAC, P2P networks, and JXTA. Following this introductory material, we discuss a content distribution application that motivates the need for RBAC in a P2P network. We then present our JXTA implementation, utilizing the example application to illustrate the concepts.

2 Background

In this section we discuss RBAC, P2P networks, and JXTA. Each of these topics is covered very briefly, but sufficient detail is provided so that the remainder of the paper is accessible.

2.1 RBAC Overview

Role based access control is a mechanism that provides for relatively simple and flexible administration while allowing users to have access to the resources that they need. In an RBAC-based system, users are associated with roles, and roles are associated with operations.

In RBAC, an administrator can define a role and assign users to—and remove users from—that role as necessary. RBAC can be contrasted with standard UNIX permissions where a user is directly associated to a resource via owner and group permissions. RBAC is particularly beneficial when, for example, the user base changes often, since roles are likely to be relatively stable. In such a scenario, RBAC will greatly reduce the administrative overhead of the access control regime.

RBAC can also provide for role hierarchies so that related roles can be expressed via inheritance relationships [9]. Consequently, instead of having to define all roles independently, an administrator can define roles with respect to other roles.

In addition, RBAC fits in well with object oriented techniques since access to a resource can be equated with access to an object method. A role then equates to the concept of an interface, where an interface is simply a set of methods that provide some capability. An interface abstracts away the implementation from the object that is implementing it.

We can also use RBAC to enforce separation of duties by not allowing a user to operate in certain multiple roles. For example, in a transaction that requires a user to initiate a payment and another user to authorize the payment, it may not be appropriate for the same user to hold both roles.

Many potential P2P applications have clearly defined roles. For example, consider a buyer-seller application, where, say, Peer 1 is the buyer and Peer 2 is the seller. In this scenario, we could use RBAC to ensure that Peer 1 only has access to the appropriate methods on Peer 2 and visa versa. We consider a specific P2P application in some detail in Section 3.

2.2 P2P Overview

As the name suggests, a peer-to-peer architecture is one where every computer on a network is considered to be on equal footing. This is in contrast to a client-server architecture, where there are tiers of responsibility and each tier includes machines dedicated to specific tasks. Peer-to-peer technology has given rise to powerful and popular applications including file sharing, instant messaging, web-based meetings, interactive gaming and sharing of comuting resources.

In a P2P architecture, a direct communication channel can be established between two parties without going through a middle-man. However, this lack of centralization creates challenges with regard to how nodes publish and discover resources. There are various classifications of P2P architectures based on the discovery mechanism [13].

The lack of central administration also brings about challenges with respect to access control. That is, without central administration, it is more difficult to restrict access to specific resources. Of course, an application can include a mechanism to restrict access to resources, but this approach requires the developer to focus on access control in addition to the development task at hand. Such a per-application approach is undesirable for many reasons, not the least of which is that it is error prone. It would be preferable if an access control technology were available that the developer could directly utilize. The purpose of this paper is to provide such a technology, based on RBAC and implemented within the open JXTA framework.

There appears to be limited prior work on RBAC for P2P systems. Tran, et al [12] discuss some issues related to this problem, but their focus is on measuring and scoring trust within a P2P system that has strong anonymity protections. The paper of Park and Hwang [8] is more closely related to the work presented here. Park and Hwang describe their own middleware-based architecture for providing RBAC in a P2P environment. In contrast, our approach is based on an open standard, JXTA. Also, Park and Hwang rely on "policy servers" to verify roles, which makes the P2P application more complex and more centralized. Our approach requires no such policy servers.

2.3 JXTA Overview

JXTA—short for "Juxtapose"—is an open standard for peer-to-peer communications [5]. As illustrated in Figure 1, JXTA defines and implements a set of protocols that allow communication and collaboration between devices on a network.

JXTA, which is maintained as an open-source project, allows disparate devices to communicate and interoperate seamlessly. For example, a portable digital assistant (PDA) may communicate with a server, and a cell phone may communicate with a PDA, all using JXTA.

In effect, JXTA creates a logical layer above the physical layer. That is, a P2P architecture is built over the physical layer, thereby reducing the complexity of interfacing with the network and physical layers.

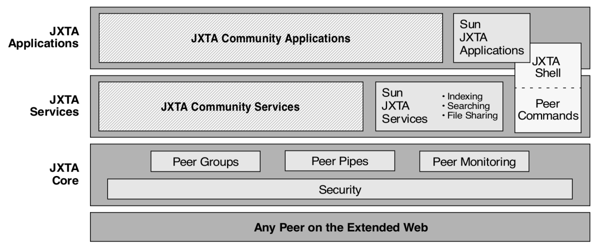

Figure 1: JXTA Architecture [5]

JXTA is being used in applications today. For example, the National Association of Realtors employs a JXTA P2P network to interconnect various realtor databases, enabling a search without importing all the data into a centralized database.

We now take a look at the basic structure of JXTA. In particular, we discuss JXTA peers, peer groups, network transport, services, and advertisements. Here, we only scratch the surface—for more details see [5] or [7].

The fundamental building block of JXTA is a peer. In the traditional P2P domain, a peer can be viewed as an application instance running on a machine. However, in JXTA a peer is defined slightly more broadly—a JXTA peer is any node capable of speaking the JXTA protocols [7].

In the non-JXTA world, different P2P applications use different protocols. For example, all peers involved in instant messaging must speak a particular protocol (such as ICQ), while peers involved in file sharing must speak a different protocol (such as Gnutella). However, in JXTA all peers speak the same "language", namely, the set of JXTA protocols. In JXTA, there is a single P2P network, which is divided into peer groups, where a peer group consists of a set of peers that provide a common service.

For JXTA peers to communicate, they must first form a peer group. A peer group is created and detected via peer group advertisements. By default, the *join* functionality for a peer group allows access to anyone, but this can be customized to be more restrictive. The default peer group is called the `netPeerGroup`, which acts as a bootstrapping mechanism to allow a peer to access the capabilities of a peer group.

The fundamental communication mechanism between peers is a *peer pipe*, which provides unidirectional communication from one peer to another. To communicate using a pipe, a peer must first identify a source endpoint and a destination endpoint. A pipe is then bound to these two endpoints. The pipe is merely a virtual artifact that indicates data is being transmitted between the two bound endpoints—the endpoints themselves interface with the network to transmit data. To implement bi-directional communication, two pipes are used. The wrapper that is used to transmit data over a pipe is known, appropriately, as a JXTA *message*.

JXTA services are the functionality provided by peers. File sharing is an example of a such a service. *Peer services* are services provided by a single peer to other peers on a network. *Peer group services* are services provided by a peer group to its members. Multiple peers in a peer group may provide the same service.

In JXTA, advertisements are used by peers and peer groups to make known the services they provide. Peers, peer groups, pipes, endpoints, and services are all defined using JXTA advertisements.

At this point, we have covered enough background information to discuss our RBAC implementation. But before doing so, we motivate our work by briefly considering a specific P2P application where RBAC would be particularly useful.

3 Content Distribution

We now turn our attention to an example application that motivates the need for an RBAC capability within a P2P environment. A similar application is also used as the model application in [8].

Consider a content distribution application where a producer works with a distributor to deliver content to a consumer. This type of application has certain advantages if implemented over a P2P network. For example, in a client-server architecture the server must maintain all content as well as workflow status and state information. The scalability of the application would therefore be limited by the scalability of the server. However, in a P2P architecture, each item is only required to pass through the peers that need to act on it as part of the workflow. In effect, a P2P implementation distributes the workflow among the peers.

If the application requires that functionality be divided among different peers based on roles—as in RBAC—then JXTA puts the onus of defining roles, defining responsibilities for the roles, and enforcing system access based on those roles on the application programmer. This could be cumbersome and error-prone for an application developer.

By implementing RBAC within the JXTA framework, we can relieve application developers of the task of implementing the access control, and allow them to focus on developing the application. This is why integrating RBAC functionality into JXTA is of value to application developers.

In our hypothetical content distribution application, we have the following roles.

- Content Producer: This role is responsible for creating the digital content.
- Distributor: The distributer is the "middle-man" who is responsible for getting content from the producer and selling it to the consumer.
- Content Consumer: This role discovers and purchases the content.

Using our RBAC implementation, the application developer would be relieved of dealing with all access control issues that are not application-specific. The developer would simply need to translate specific roles into role names and translate the corresponding responsibilities into application-specific methods.

In the next two sections we discuss our JXTA implementation of RBAC. In order to illustrate the implementation, we refer to the above application. It should be noted, however, that the example application in our reference implementation is a relatively simple arithmetic application [6]. We chose to implement this particular application simply because it is easier to verify that the RBAC restrictions are being properly enforced. Implementing the content distribution application described here would not be significantly more challenging.

4 Implementation Overview

Authentication deals with ensuring that a peer is who he claims to be while *authorization* deals with ensuring that a peer has access only to those resources that it should. RBAC only deals with authorization. Therefore, in our implementation, we assume that all peers have been properly authenticated. Many authentication methods already exist in JXTA and these will not be discussed further in this paper.

For our JXTA implementation of RBAC, we assume that access to a resource is equivalent to access to a method from a peer pipe. A method is called remotely over a JXTA socket. A remote method call object is marshaled and de-marshaled across the socket. The result is processed on the server peer and returned to the client peer. This technique is in alignment with object oriented techniques [2].

We have chosen to use the Extensible Markup Language (XML) to configure the RBAC authorization. Below, we provide two snippets of XML to illustrate the configuration process. See [4] for further information on XML.

Our RBAC framework uses JXTA sockets. JXTA sockets provide a familiar socket interface on top of JXTA and provide reliability. However, according to some sources, a potential drawback of JXTA sockets is relatively low one-way throughput [3]. This is not an issue with our reference implementation, but might conceivably create a bottleneck in certain applications. There are alternatives to JXTA sockets that can be employed if throughput should become an issue [3].

Creating a JXTA socket involves passing a pipe advertisement. JXTA sockets inherit from Java sockets, so sending and receiving objects across the wire is straightforward since techniques to transfer objects over sockets are well-known.

Every peer has the ability to request method execution on another peer as well as to provide the implementation of methods to other peers. In other words, every peer may act as both a client that makes a request and as a server that services a request—as would be expected in a P2P environment.

The following assumptions and restrictions apply to our RBAC implementation.

- All peers must have the same role definition and role mapping files before the system is started. The system enforces that this is the case, but does not provide a mechanism to distribute the files to all peers.

- Each peer can only function in one role at a time. This restriction is for simplicity and convenience and is not an inherent limitation of our approach.

- During the life of the application, a peer cannot change roles. This is a significant restriction, but it greatly simplifies the RBAC implementation. It is possible to start a new application where peers have different roles, which somewhat mitigates this limitation.

- The peers must agree on the role configuration information and the peer-to-role mapping. Below, we see that this is not difficult to enforce.

These assumptions allow us to create a simple but useful RBAC framework. Additional features and flexibility would tend to complicate the design and implementation. Since it is generally accepted that "complexity is the enemy of security" [11], we have consciously attempted to design a simple, yet useful system. For more detail on our design choices and various alternatives, see [6].

Before presenting our implementation in more detail, we want to emphasize that we are not claiming that RBAC itself is inherently more secure than alternative authorization methods. As discussed above, given the nature of P2P applications of interest, we believe that RBAC is a logical approach, which simplifies the burden of configuring and managing authorization. Perhaps it could be argued that this simplicity is itself a security benefit, but it is far from clear that the savings in complexity are sufficient to claim any real security advantage. There is, however, a slightly more subtle security concern that must be addressed. That is, we must show that our specific RBAC implementation does not create any security issues beyond those which are inherent in the nature of the application and the computing environment. We briefly address this topic at the end of Section 5.2, after we have provided more details of our RBAC implementation.

5 Implementation Details

This section covers our RBAC design in greater detail and provides a closer look at the JXTA implementation. In our implementation, each peer implements three components that we refer to as the *XML engine*, the *RBAC engine* and the *execution engine*. These are illustrated in Figure 2. Each "engine" is described in more detail below.

5.1 XML Engine

The XML engine reads XML configuration files that specify details about security processes. In particular, these files list the methods that specific roles can access and the mapping of peers to specific roles. The XML engine parses these XML files and stores the results in data structures for use by the peer.

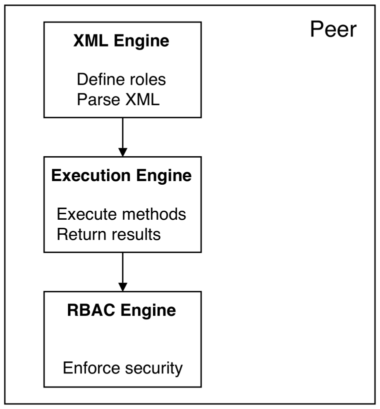

Figure 2: Peer Components

The process of defining roles consists of the following three operations. First, the application programmer must define roles and select role names. Next, the developer must define the methods that each role implements. We refer to these as the methods that the role *publishes*. Finally, the developer must specify the methods that each role can call. We refer to these as the methods that the role can *access*.

Roles are defined in `RolesConfiguration.xml`. Necessarily, roles are application-specific. Below is a sample `RolesConfiguration.xml` file for the content distribution application described in the previous section.

```
<?xml version="1.0"?>
<RolesConfig>
  <Role>
    <rolename>ContentProducer</rolename>
    <publishmethod>getContent</publishmethod>
    <accessmethod>getRoyalties</accessmethod>
  </Role>
  <Role>
    <rolename>ContentDistributor</rolename>
    <publishmethod>searchContent</publishmethod>
    <publishmethod>buyContent</publishmethod>
    <publishmethod>getRoyalties</publishmethod>
    <accessmethod>getContent</accessmethod>
    <accessmethod>getPayDetails</accessmethod>
  </Role>
  <Role>
    <rolename>ContentConsumer</rolename>
    <publishmethod>getPayDetails</publishmethod>
    <accessmethod>searchContent</accessmethod>
    <accessmethod>buyContent</accessmethod>
  </Role>
</RolesConfig>
```

In this example there are three roles and five methods. The roles are ContentProducer, ContentDistributer, and ContentConsumer, while the five methods are getContent, getRoyalties, searchContent, buyContent, and getPaymentDetails.

In the ContentProducer role, the peer provides an implementation for the getContent method. As its name implies, this method is responsible for providing the actual digital content (e.g., digital book, music, etc.). The ContentProducer peer would call the getRoyalties method periodically to settle accounts.

The ContentDistributer peer provides implementations for the following methods: searchContent, buyContent, and getRoyalties. The searchContent method enables the ContentConsumer to find what they are looking for and the buyContent method provides the content to the ContentConsumer (presumably, for a fee). The ContentConsumer peer provides the implementation for the getPaymentDetails method which provides the necessary payment information to settle the transaction.

Apart from defining roles, the mapping between peer names and roles must be specified. This is also accomplished using an XML file called `PeerRoleMapping.xml`. Again, this mapping is application-specific. Below is a sample `PeerRoleMapping.xml` file for our content distribution application.

```
<?xml version="1.0"?>
<PeerRoleMapping>
  <Peer>
    <peername>peer1</peername>
    <rolename>ContentProducer</rolename>
  </Peer>
  <Peer>
    <peername>peer2</peername>
    <rolename>ContentDistributer</rolename>
  </Peer>
  <Peer>
    <peername>peer3</peername>
    <rolename>ContentConsumer</rolename>
  </Peer>
</PeerRoleMapping>
```

The interpretation here is clear, that is, Peer 1 is in the ContentProducer role, Peer 2 is in the ContentDistributer role, and Peer 3 is in the ContentConsumer role.

To summarize, the access allowed by each role is defined in `RolesConfiguration.xml`, while the mapping of peers to roles is defined in `PeerRoleMapping.xml`. Together, these two XML files define the security policy (with respect to authorization) for the P2P system.

5.2 RBAC Engine

The RBAC engine, which enforces the security policy for role based access, is the heart of the system. The following algorithm is used to ensure proper access to methods based on roles.

1. When a peer is started in JXTA, it is given a peer name.

2. During initialization each peer reads the two XML files discussed above, namely, `PeerRoleMapping.xml` and `RolesConfiguration.xml`. The purpose of defining both of these XML files in all peers is to ensure that all peers are implementing the same security policy.

3. Both XML files are parsed into specific data structures which are then wrapped as objects. Suppose that Peer 1 sends a method execution request to Peer 2. Then Peer 1 also sends its version of the XML configuration files to Peer 2. This is accomplished by sending the objects that contain the data structures resulting from the XML parser. Peer 2 then checks to ensure that the contents of its XML files agree with those it has received from Peer 1. If the contents vary, Peer 2 throws a security exception. In this way, a malicious peer that tries to alter an assigned role will not be allowed improper access.

4. If the contents of the XML configuration files agree, then the security engine processes the request for the method execution. The first part of this request contains the peer name of the requester. Suppose the requester is Peer 1. Then Peer 2 obtains the role for Peer 1 from its data structure. Suppose that the assigned role for Peer 1 is Role A. The security engine checks the definition in `RolesConfiguration.xml` to see if Role A indeed has the permissions to access the method it seeks to execute. If the security engine finds that as Role A, Peer 1 cannot access the method that it seeks to execute, it throws a security exception.

5. If Peer 1 does have permission to access the method it wishes to execute, Peer 2 examines its role and verifies that it indeed publishes the method that Peer 1 is seeking to access. If Peer 2 does not publish the specified method, a Security Exception is thrown.

6. If Peer 2 does have permission to provide an implementation of the accessed method, then it executes that method and returns the results to Peer 1.

We have implemented this algorithm as part of the JXTA framework. This effectively relieves the application develop of the need to deal directly with these RBAC-specific issues.

Before moving on, we briefly consider two possible attacks on the algorithm, above. First, suppose that Peer 1 in Role A wants to convince Peer 2 to give it access to a resource that is only available in Role B. Peer 1 could change its XML configuration file, `RolesConfiguration.xml`, to show that Peer 1 is in Role B. Then Peer 1 could send this altered XML file to Peer 2. However, before Peer 2 would allow Peer 1 access in Role B, Peer 2 would check its own `RolesConfiguration.xml` file and discover an inconsistency with the file received from Peer 1. As stated in 3., above, this would result in a security exception. Consequently, a successful "attack" on a remote peer requires the cooperation of the remote peer. But this cannot be considered an attack, since the remote peer can always provide the requested information outside of the given security system. In other words, if the local host and remote host collude, they can work outside of the authorization scheme, and thereby bypass any security it provides.

On the other hand, Peer 1 can change its role from, say, Role A to Role B when requesting a local resource. This would succeed because in this case, Peer 1 only checks its own XML configuration file. However, it is generally infeasible to prevent a client from "attacking" itself—regardless of the access control method employed. That is, any authorization scheme would be vulnerable to an analogous attack. In other words, this attack does not reveal a flaw in our system. In fact, to limit the damage of such attacks the client would need a strong "self-defense" mechanism, which naturally leads into the realm of digital rights management (DRM), a discussion of which is beyond the scope of this paper; see [1, 10, 11] for more information.

Other attack scenarios could be considered, but these are the most obvious concerns for our RBAC system. As indicated above, our system is as secure against these attacks as could be expected, given the limitations imposed by the application and the computing environment.

5.3 Execution Engine

The execution engine consists of a set of classes that use the JXTA protocols and sockets to enable communication between peers. The purpose of this communication is to transmit method execution requests and send results in support of the security engine.

6 Conclusion and Future Work

We discussed a technique for implementing RBAC in a P2P system and we outlined an implementation within the JXTA framework. See [6] for additional details on various aspects of the design and implementation of our RBAC framework, including class diagrams, sequence diagrams, test results, etc.

Using our RBAC framework, a developer only needs to specify two XML configuration files and write the necessary application-specific code for each role. Having done so, our RBAC implementation will enforce the security policy without further effort on the part of the developer. If utilized properly, this technique should provide a relatively transparent and secure authorization regime for P2P applications.

Possible enhancements and future work include the following.

- The ability to load balance across multiple peers implementing the same role would be desirable. This could be accomplished by simply having the peer that is attempting to connect to a remote peer randomly choose from those peers providing the required role.

- The ability to have a peer change roles at run time would allow for a peer to be in a role for the minimum amount of time required. This would better support the concept of least privilege. However all peers would need to agree on the change and update their roles files accordingly. This level of coordination would significantly complicate the implementation.

- A performance analysis of the system for large numbers of objects and large numbers of peers has not been conducted.

- Finally, a DRM application that uses our RBAC P2P framework would be an interesting project. Our approach seems particularly well-suited for use in conjunction with Microsoft's Next Generation Secure Computing Base (NGSCB) [1, 11].

References

[1] R. Anderson, 'Trusted computing' FAQ, at `www.cl.cam.ac.uk/~rja14/tcpa-faq.html`

[2] J. Barkley, Implementing role based access control using object technology, First ACM Workshop on Role-Based Access Control, 1995

[3] E. Halepovic and R. Deters, JXTA messaging: analysis of feature-performance tradeoffs, Department of Computer Science, University of Saskatchewan, Saskatoon Canada

[4] E. R. Harold and W. S. Means, *XML in a Nutshell*, Third Edition, O'Reilly, 2004

[5] JXTA website, `www.jxta.org`

[6] A. Mathur, Incorporating role based access control into the JXTA peer-to-peer infrastructure, Masters thesis, Department of Computer Science, San José State University, 2005

[7] S. Oaks, B. Traversat, L. Gong, *JXTA in a Nutshell*, O'Reilly, 2002

[8] J. S. Park and J. Hwang, Role-based access control for collaborative enterprise in peer-to-peer computing environments, 8th ACM Symposium on Access Control Models and Technologies, Como, Italy, June 2–3, 2003

[9] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, Role-based access control models, *IEEE Computer*, Vol. 29, no. 2, pp. 38–47, 1996

[10] M. Stamp, Digital rights management: the technology behind the hype, *Journal of Electronic Commerce Research*, Vol. 4, No. 3, 2003

[11] M. Stamp, *Information Security: Principles and Practice*, John Wiley & Sons, Inc., 2005

[12] H. Tran, M. Hitchens, V. Varadharajan, and P. Watters, A trust based access control framework for P2P file sharing systems, *Proceedings of the 38th Hawaii International Conference on Systems Sciences*, 2005

[13] Wikipedia, the free encyclopedia, peer-to-peer, at `en.wikipedia.org/wiki/Peer-to-peer`