Alternative Java security policy model[1]

S. Cloutier, C. Gustave, R. Khoury, D. Nassour, A. Robison, F. Samson, N. Tawbi
{Christophe.Gustave, Andrew.Robison}@alcatel.com {Simon.Cloutier, Raphael.Khoury, Dani.Nassour, Frederic.Samson, Nadia.Tawbi}@ift.ulaval.ca

ABSTRACT
Java language and technology [3, 4, 10] were proposed with security in mind, yet there are some limitations especially when it comes to running Java applications in a distributed context. In this work we propose a new security policy together with the relevant verification mechanisms. This model is aimed at controlling the access to the system resources in a trustable and flexible way. This model extends the existing security mechanisms offered by Java in two ways: first the authentication is based on public keys rather than global names which is more flexible. Second our model supports delegation which is of major importance in a distributed context. The new model has been integrated in a Jini-based platform.

KEY WORDS: Distributed Systems, Security, Authorization, Jini, Java

1 INTRODUCTION

The distributed nature of todays open environments creates new requirements and challenges for deploying services across network boundaries in a ubiquitous and secure way. In this context, establishing and subsequently maintaining security properties is a challenging process. Indeed, heterogeneous components - part of the overall information system - have to share common security policies and interoperate with various security models.

Many approaches have been proposed for securing distributed architectures. However, most of them are limited to a subset of security policies for two main reasons: inherent complexity due to the number of components, protocols and data format, and diversity of such architectures. Our approach is to provide the necessary flexibility to implement various kinds of authorization policy constraints without increasing the systems complexity. In this paper, we present the result of our research work aiming at enhancing the authentication mechanism and the authorization policy to be implemented into Jini-based platforms. We present the Distributed and Secure Java Virtual Machine (DSJVM). In this framework the authentication mechanism is more flexible than the classical one and the authorization scheme is extended in order to support delegation. The extensions to the authorization mechanism are inspired by Ponder, a formal language for expressing security policies.

1.1 Security Needs Into Service Brokers

The advent of distributed e-business service broker infrastructures, in essence vulnerable to various network-based threats, brings out new requirements in term of implementation of the underlying services platform. Security can not be considered anymore just as an after-thought design process. Rather, it has to be implemented at the heart of the system, ensuring both scalability and extensibility from a security deployment perspective. Furthermore, network infrastructures are undoubtedly as secure as their weakest link. Thus, it is crucial that security functions are coordinated

[1]This work has been funded by Research & Innovation, Alcatel, 600 March Road Kanata, ON K2K 2E6

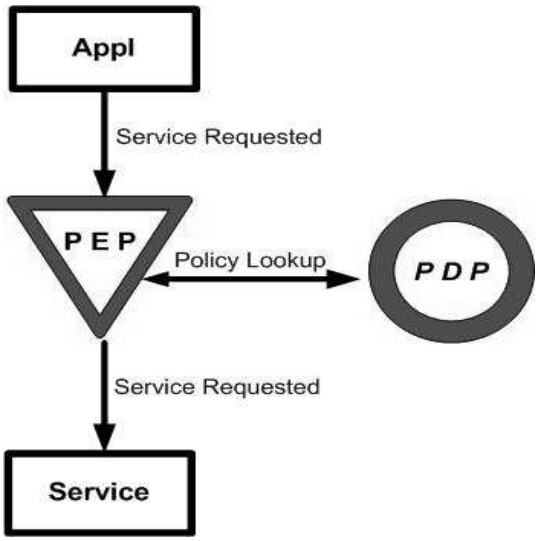

Fig. 1: Access Request Enforcement

and exchange information in a way which avoids exposing sensitive assets to unauthorized and potentially malicious third-party entities. Application services accessing underlying communication brokers need firstly to be authenticated and consequently should only be able to interact with the service entities that they are authorized for. In the context of access control policy decision, at this point, we need to distinguish two different kinds of authorization entities: The first is the identified subject or participant - Human, machine, or process - initiating the request and the second is the object - ultimately a set of network services resource - to be accessed.

A sound access control model should clearly identify the different kind of subjects operating in the system, along with the scheme used to map them to the various controlled resources. In a distributed environment, this can be a rather complicated process requiring a lot of digging and prone to administrative errors. Generally, the PDP (Policy Decision Points) duty is to assign the specific access control rules that applies. Ultimately, the PEP (Policy Enforcement Point) launch a query-access ressouce to the appropriated PDP. The PDP applies the rules and replies subsequently by denying or allowing access to the resource. Figure 1 shows the basic interactions involved when a subject, such as an application, request access to a network resource entity. Access requests to the targeted service must be regulated through the PEP engine.
One of the key points is to maintain a coherent access control semantic throughout the whole system. Indeed, two different policies could interfere between each other, and enforcing a policy could result in an inconsistent access rule, putting in place a back door or digging an unexpected security hole in the core system. This brings the needs for an access control policy model capable to streamline the security administration process, decreasing the risk of administrative error while globally setting up a consistent security policy.

In the scope of our research, we carried out a new access control security policy model that meets the challenging requirements of todays complex networking environments. This model is mainly based upon the Ponder formal language, whose inherent expressiveness allowed us to define easily and consistently a wide range of security policy constraints. On the other hand, the authentication

mechanism is based on public keys as identities of the participants. While focusing, in the next section, on the different security properties that are enforced in our platform, we present, in Section 3, the Jini-based platform along with how we added new security properties. In Section 4, we discuss different alternatives to our solution along with potential limitations. In Section 5, we draw a brief conclusion.

2 ENFORCED SECURITY PROPERTIES

The Java security policy model is mainly aimed at enforcing authorization policies. The extensions we bring to this model are inspired by the Ponder Specification Language [2]. This Language allows the expression of very powerful security policies. It is aimed at enhancing the expression power of security policy specification related to access control. Namely it allows the expression of delegations, exceptions and negative authorizations.

Authorization are granted or denied depending on the identity of a participant. Hence, as a first step, authentication of the participants has to be performed.

The security properties that we wish to implement in our system are authentication, authorization, data integrity, and confidentiality. In authorization, we wish to implement more precise properties. They are positive and negative authorization, exceptions, and constraints. We also implemented delegation in our system.

3 DISTRIBUTED AND SECURE JAVA VIRTUAL MACHINE PLATFORM

The platform of our distributed system is based on the Jini network technology. The major motivation underlying this choice is that Jini offers a flexible platform in which service providers and clients could connect themselves very easily. Furthermore, the Jini leasing mechanism garantees a good reliability of services. A service is connected for a laps of time after which the lease has to be renewed. This is not possible if the service is no more available.

The integration of our model into this technology consists on one hand, in integrating a new policy verifier that controls the access to the system resources and in creating a new authentication module.

A description of the Jini network technology together with our modified security policy administration are given in the sequel.

3.1 Jini's Service Delivery Model

Introduced in January 1999 [5], the Jini technology is a network technology that enables developers to implement services that are centralized, meaning that they can be accessed by a lookup service that acts like a central server. It is often said that Jini federates because the services that are part of the community join it like they would join a federation; they work together and offer their services to all the other clients and services of the network. The machines and applications offer their resources to the other members of the federation. They can be implemented in either software or hardware. The resources of the services on the network appear to the clients as objects in the Java programming language. It is a set of services that are able to communicate with each other with little outside intervention. Jini provides a framework for these different services to communicate with each other.

Jini is most often defined using the hardware services as an example. A person could come into a room that he or she has never been before with a laptop and connect it to the network and

the laptop would immediately know what services are available on the network. If there were a printer available on the network, the laptop would find it easily, installing and downloading just-in-time through the network all the necessary drivers, without any end-user intervention. This is the important part; the advantage of Jini is its simplicity. The clients and services can join and leave the network with little intervention and configuration from users.

3.2 Extending The Security Policy Administration

The first step in the creation of a network that uses our system is to create the security policy file. This file is a text file that helps the system to decide whether to grant an access or not.

When a service provider (server) is started, this server can receive requests by clients. When this happens, the client is required to authenticate itself to the server and the server is required to authenticate back to the client.

After the authentication has been performed, sensitive operations eventually happen. At that time, the client sends its request to the server and the server verifies its security policy. If the security policy grants this user permission to perform that operation, the operation is performed and the result is returned. If the policy denies this user permission to perform that operation then the operation is not performed.

Our implemented JAAS (Java Authentication and Authorization Service) [11] login module is responsible for authenticating users. The client is identified as a public key. Using the SSL protocol, the client authenticates itself on the server and the server authenticates itself on the client. Only after that happens will further communication occur. Also, SSL authentication results in a symmetric encryption key known only by the two participants in the authentication protocol. That key is used to encrypt the data on the network to ensure confidentiality and data integrity. Notice that the use of symmetric key is more efficient than the public one.

The Login And Authentication Modules. We designed and implemented a JAAS Login Module that leverage the SSL protocol to authenticate users. This SSL-based login module also generates the secure sockets that are used in the communication. Those secure sockets are created using the symmetric key generated during the authentication process and known only by those two communicating parties. All the information sent between the client and the server transit through those secure sockets and is therefore encrypted and decrypted using that symmetric key.

Generating Public And Private Keys In Java. Before being able to use the network, a public and a private key must be generated for each participant using Keytool. Keytool is a Java program that generates keys and requests for public-key certificates. Those requests should be sent to a certification authority that will delivers public-key certificates. The keys together with their associated certificates are stored in a keystore repository.

Authorization Policies Verification. Our system uses policy files written in the XML language. A policy input tool helps the administrator to create the policy file. This tool receives the policy property in a user-friendly way. It then translates it into an XML syntax and stores it in the policy file. Those files contains permissions specifying what actions the users of the system are allowed to perform.

When a sensitive operation is attempted, the system verifies those permissions and decides whether the user is allowed or not to trigger that operation.

The verification process of the permissions is the core part of the system where the security policy file is analyzed to determine if users are granted access to perform certain actions or not.

The whole architecture of the Java security management does not change except that we removed the standard Java policy provider and we replaced it with our own policy provider called XMLPolicy. In this policy provider, the verification process is more complex than the standard one because one has to check for the delegations, the constraints, the exceptions, the negative authorizations as well as the positive ones in order to grant an access or not.

Notice that, only the owner of a resource is able to grant or to deny access to this resource. The verfication related to the access request is performed on the owner site. This is how the consistency of all the policy files on the distributed system is garanteed. On the other hand, consistency of a local policy file is garanteed because negative authorizations have priorities over positive ones if they are related to the same resource.

4 DISCUSSION

This section discusses alternatives to Jini that we could have used for our distributed system. We studied distributed systems such as CORBA (Common Object Request Broker Architecture) and Microsoft .NET. This section also looks into XACML (eXtensible Access Control Markup Language), an access control mechanism based on XML[2].

4.1 Jini Vs. CORBA

CORBA [7, 9] was a possible choice for a distributed system because it is possible to use it in Java. CORBA, developed by the Object Mangagement Group (OMG), is also a very mature distributed system that offers a wide variety of possibilities. It is independent of the programming language used by the developers. This means that a client working on a system programmed in C can communicate with a server programmed in Java.

CORBA uses a standard protocol called IIOP[3] to communicate. IIOP is the protocol that makes it possible for CORBA clients and servers to communicate using an ORB[4] regardless of the programming languages used to build them.

The reasons underlying the choice of Jini against CORBA are twofold. First, CORBA support code mobility but it is more efficient in Jini using Java. Second, Jini has built-in components like leasing that makes it easier to build stable networks.

4.2 Jini Vs. Microsoft .NET

We also looked into Microsoft .NET [8], which is a set of Microsoft Internet software technologies used to connect people and devices together [1]. Using a Common Language Runtime (CLR), it

[2]eXtensible Markup Language: A standard for writing and exchanging structured documents especially on a network.

[3]Internet Inter-ORB Protocol.

[4]Object Request Broker: A CORBA object that takes care of sending the request from the client to the server and routing the response back to the client.

lets different applications developed in different programming languages communicate together. A programming language called C# was developed specifically for use with Microsoft .NET.

In the centre of this technology are web services. Web services are applications designed to efficiently answer requests coming from the Internet.

Microsoft .NET uses standards such as XML and SOAP (Simple Object Access Protocol) to communicate information on the Internet from web services to clients. Using standard protocols makes it simpler for anyone building applications to use Microsoft .NET.

Still, we did not use Microsoft .NET because, while it supports multiple languages, it does not support Java and we were looking for a distributed system that uses the Java programming languages.

4.3 Ponder Vs. XACML

XACML [6] is an XML-based standard for defining security policies to perfrom access control. It has become necessary to have a standard for expressing security policies because they are often stored in many different places and then read by many different systems. This section is an introduction to XACML.

A subject wants to perform a certain action that requires using a protected resource. On a network, the subject would make a request on a server that manages the resource. The server is the PEP. The PEP creates a request using various information such as the identity of the subject, the type of action that he or she wants to perform, and exactly which protected resource it needs to perform the action.

The PEP sends this request to the PDP. The PDP receives this request and compares it to the policies that are in effect at this point. It looks for policies that control the protected resources present in the request. Using those policies, it can decide if the request should be granted or not. There are actually four possible answers that can come from the PDP. They are *permit*, *deny*, *indeterminate*, and *not-applicable*. Using that response, the PEP then either grants or denies access. The PEP and PDP are separate entities, they can be distributed on a network or together on the same server.

XACML provides a policy language for writing the security policies. It provides a way to efficiently find the security policies that apply to a given resource and then using them to evaluate the request and efficiently decide whether access should be granted or not.

XACML could have been used. The only advantage it offers against our solution is that it is easier to communicate security policies over the network. In our platform, this is not mandatory because every service owner administrates its security policy locally.

4.4 Design Decisions

Most of our design decisions were driven by security aspects. In this case, we concentrated our efforts on security features into distributed systems but some of our results can be used in non-distributed systems as well. To attain our goal of security, we made several design decisions that we explain in this section.

For our research, we decided to use the Java programming language. Besides the fact that Java supports code mobility, Java was created with security and safety in mind. Java is strongly typed, does not support explicit pointer manipulation. Moreover, Java Virtual Machine offers a verifier and a security management engine. Furthermore, Java security mechanisms are offered in a very flexible way and could be easily extended.

Our next choice was on which distributed system we could make more secure. Since we already decided to use Java, we needed a distributed system that could use it. After studying several distributed systems, we decided to use Jini. Jini is a technology that enables the creation of distributed systems written in Java in a flexible and dynamic way. It therefore inherits all the security features found in the language. It also offers code mobility via the RMI mechanism offered by Java.

The Ponder Specification Language enables administrators to express very powerful security policies. We used Ponder to extend the authorization mechanism offered by Java.

Using a global name space as a basis for authenticating participants is very convenient when the number of participants is limited. Basing the authentication on public-keys enables us to avoid the use of global name space and to enhance the flexibility of our platform. Hence, clients and servers are free to choose their names on a network even if the name they choose is already used elsewhere on the network, public key certificates ensuring the association between a public-key and its corresponding participant.

To perform authentication, we implemented a new JAAS login module that uses the secure sockets layer. SSL has been improved over the years. The last version SSL 3.0 could be considered as a trustable protocol [12].

4.5 Platform Limitations

Our security model could be easily extended with other authorization policies such as obligations. Obligations are a set of actions that a subject must perform under specific circumstances. On the other hand, the constraints we used are restricted to date and time. The policy provider, which is responsible of granting and denying accesses to resources, could be easily modified to support other types of constraints, such as the operating system on which the application is running, the resolution of the monitor, or the speed of the network connection.

Lastly, we did not test the scalability of our system. This refers to the possibility of a system running as efficiently with a big number of users as it does with a small number of users. We did test our system with a small number of users using the calculator application. We think that Jini-based platform should be able to scale very easily. The authentication based on public-keys is another feature that would improve the scalability.

5 CONCLUSION

As of today, many distributed systems co-exist, each one featuring its own strengths and weaknesses. The inherent nature of distributed platforms brings out potential communication failures, mainly due to both service brokers interoperability problems (incorrect message formats) and specific networking issues. Our experiment has shown that using Jini as the underlying broker platform is a good choice to foster the development and ease the deployment of distributed services. Indeed, in unstable environments, often service locations change and various consumer actors needs to access

not a-priori known services. In this context, Jini distribution model appears as the most suitable solution to build up a ubiquitous and pervasive services platform.

This formal approach enables defining a broad range of security policy requirements. In this case, we focused more specifically on authentication and authorization features, but this could be extended as well to other security-related properties such as security audit. The DSJVM leverage Jini as the underlying service-oriented communication broker. The overall security architecture of Jini has been strengthened, and authorization policy management has been dramatically simplified. The flexibility is enhanced and strengthened by the fact that authentication is performed on public-key bases avoiding the use of a global name space. The public key is used only in the authentication phase, the information exchange is performed using a symmetric key, which is more efficient.

References

[1] James Conard, Patrick Dengler, Brian Francis, Jay Glynn, Burton Harvey, Billy Hollis, Rama Ramachandran, John Schenken, Scott Short, and Chris Ullman. *Introducing .NET*. Wrox Press Inc., 2000.

[2] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The Ponder Specification Language. *Lecture Notes in Computer Science*, 1995:18–39, 2001.

[3] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification Second Edition*. Addison-Wesley, Boston, Mass., 2000.

[4] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, Second edition, 1999.

[5] Scott Oaks and Henry Wong. *Jini in a Nutshell*. O'Reilly & Associates, Inc., 2000.

[6] OASIS. eXtensible Access Control Markup Language (XACML) version 1.1, July 2003. http://www.oasis-open.org.

[7] Object Management Group. CORBA Specification v1.2, December 1998.

[8] David S. Platt. *Introducing Microsoft .NET, Third Edition*. Microsoft Press, 2003.

[9] Jeremy Rosenberger. *Sams' Teach Yourself CORBA in 14 Days*. sams, 1998.

[10] R. F. Stärk, J. Schmid, and E. Börger. *Java and the Java Virtual Machine: Definition, Verification and Validation*. Springer-Verlag, 2001.

[11] Sun Microsystems Inc. Java^{TM} Authentication and Authorization Service v1.0 Specification, 1999.

[12] David Wagner and Bruce Schneier. Analysis of the ssl 3.0 protocol. In *Proceedings of the Second USENIX Workshop on Electronic Commerce, USENIX Press*, pages 29–40, November 1996.