A Secure Interface for a Universal Data Server

Flávia Linhalis*
flavialin@bol.com.br

Dilvan de Abreu Moreira*
dilvan@icmc.sc.usp.br

* University of São Paulo
Mathematic and Science Computing Institute
São Carlos, Brazil

First author address
Rua Miguel Alves Margarido, 360 apto 314
Edifício Solar dos Engenheiros
Cidade Jardim
São Carlos – SP - Brasil
13566-580

First auhor phone number: (16) 261-6786

Second author address
Av. Dr. Carlos Botelho, 1465
Cx Postal 668
13560-970 São Carlos – SP – Brazil

Second author phone number: (16) 273-9684

Paper presenter: Flávia Linhalis

Keywords: security policy, authentication, access control, universal data server, mobile agents.

A Secure Interface for a Universal Data Server

Abstract

This paper describes the implementation of a Secure Interface for a Universal Data Server (SIUDS). The SIUDS's main role is to securely interface mobile agents with the database of a universal data server. Mobile agents can serve data, stored in this database, in all sorts of formats and protocols. In doing this, they implement the core of the universal data server.

The SIUDS main functions are to receive mobile agents, authenticate them and provide access to database objects and resources securely. The mobile agents are implemented in Java. They have, in a universal data server, the same role as query languages have in a relational database. However, as software agents, they have the advantage of enjoying all the computational power provided by the Java language and environment. For this reason, they can fulfill the role of query agents much more efficiently than traditional query languages could.

Keywords: security policy, authentication, access control, universal data server, mobile agents.

1 Introduction

This paper proposes and implements a Secure Interface for a Universal Data Server (SIUDS). The SIUDS's main role is to securely interface unknown mobile agents with the database of a universal data server. Some of these agents can be hostile, this arises the need for an interface to protect the database objects.

This paper is organized in eight sessions. Session 2 presents the concept and importance of universal data servers. Session 3 proposes the SIUDS. It describes the objective and modules of the SIUDS. Session 4 describes the security mechanisms necessary to implement SIUDS roles. Session 5 presents the security resources of the Java 2 platform. Session 6 describes the SIUDS implementation and the Java 2 resources used in each module. Session 7 presents the InfoAgent, a test agent implemented to be authenticated and executed by the SIUDS. Finally, session 8 presents the conclusions of this work.

2 The Concept of Universal Data Server

There are many kinds of information servers available today, such as HTML and SQL servers. Their basic functionality is to provide information to client requisitions. Because of the variety of servers, information sites have to keep several different servers running and the store the same information they serve in different formats. It would be very useful to have all information stored in only one format and all data stored in just one server that could provide data in different formats and protocols. This server could allow its clients to send and retrieve information in a variety of different formats (HTML, SQL, etc.). This is exactly what a universal data server should do.

A universal data server [1] has to be able to respond to different kinds of requests using the same information set. All information has to be stored in a database that can be read and updated by any of the information services provided. There is only one source of data to be kept and many ways to access and update it.

The database of a universal data server has to be able to store any kind of data. It can be implemented using an Object Oriented Database Management System (OODBMS) or an Object-Relational Database Management System (ORDBMS). There are commercial database options from Oracle, Informix or Sybase [2].

3 The Interface

The SIUDS aims to securely control the access to the objects of a universal data server's database and to the resources of its host system. This access is done by foreigner mobile agents [3, 4]. The union between the SIUDS and an object-oriented (or object-relational) database can be considered as a universal data server, because the mobile agents can access any type of object in the database and serve its information to their clients in many different ways.

The SIUDS provides security by authentication and authorization. The authentication validates the identity of a mobile agent owner. The authorization defines what privileges a mobile agent will have when accessing the objects and system resources. Figure 1 gives a general view of the SIUDS. It is composed by four components: the Database Interface, the Gateway, the Pool of Agents and the Security Manager.

A mobile agent accesses the database through the Database Interface. This interface defines methods that allow the agents to create groups, store objects and associate access permissions to objects and groups. An agent will access only permitted objects. Access privileges will depend of permissions associated to the group an agent belongs to.

If the methods of the Database Interface were to be accessed only by local users, a simple access control could guarantee the security of the objects. However, as unknown mobile agents may access these methods, authentication

becomes a necessity. Unknown hostile agents could invade and damage the system. The Gateway agent provides authentication, it receives agents from remote hosts and verifies their digital signature and associated certificates. If the verification succeeds, the agent owner is trustful and the agent can run in the Pool of Agents. The function of the Pool is to run mobile agents and control their lifetime. The lifetime of each agent depends on which group it belongs to.

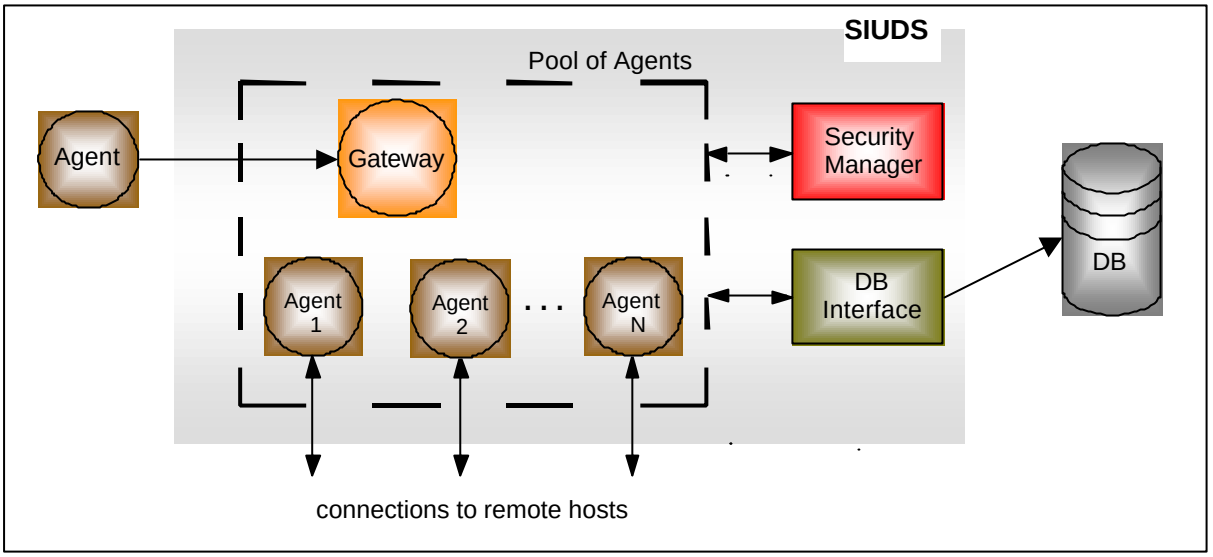

Figure 1 – SIUDS General View

The SIUDS and the agents in the Pool are implemented in Java [5]. The agents have the advantage of enjoying all the power provided by the Java language and environment. They could, depending on their group permissions, read and write to the file system, establish connections to remote hosts, modify systems properties and so on. The Java SecurityManager class controls the access to the system resources according to permissions associated to each group of agents.

As the SIUDS and mobile agents are implemented in Java, it naturally becomes the best query language to access the data stored in the database. The substitution of a query language by mobile software agents makes the applications more flexible and powerful. An application can create its own query language or even change the server behavior (for example, creating a HTTP server on port 80).

The authentication provided by the Gateway, the database access control provided by the Database Interface and the access control to system resources provided by the SecurityManager can guarantee that the Pool of Agents is an open and secure environment for mobile software agents execution. An agent can establish connections to remote resources, store objects in the database and make many other operations safely, that is, according to the privileges of its group.

4 Security Mechanisms

According to Chin [6] a secure system has the properties of confidentiality, integrity, authentication, access control, non-repudiation and availability of service. The security mechanisms used to achieve these security properties are: cryptography, digital signature, certification and authorization.

- *Cryptography*: The cryptography [7] consists of encrypting the message in the origin through an encryption method. The encrypted message is sent through the network and, in the destiny, the message is decrypted. A key is used to increase the security of the encryption mechanism. The encrypted message varies according to the encryption key used. Cryptography guarantees the confidentiality property; it does not guarantee the authenticity of the message sender. To do so, cryptography has to be used with digital signatures and certificates.
- *Digital Signature*: The role digital signatures [6, 7] have in the electronic world is the same as paper signatures in the real world. If a private key is only known by its owner, the utilization of the key is identity evidence. Digital signatures are usually used with hash functions. They are one-way functions that map messages of arbitrary length to a fixed number of bits, where for any hash value $hash(m_1)$ it is computationally infeasible to find another message m_2 such as that $hash(m_1) = hash(m_2)$ and find m_1 given $hash(m_2)$.
- *Certification*: A message can be authenticated only if the public key surely belongs to the entity that signed the message. A certificate documents the association between public keys and entities. It guarantees the authenticity of the public key. Digital signatures and certificates are used to meet the needs for integrity, authentication, access control and non-repudiation [6].

- *Authorization*: Authorization mechanisms [7] are related to access control. They grant permission to do something or to act in a particular hole.

5 Java Security

The J2SDK (Java 2 Software Development Kit) provides tools and APIs that allow the implementation of the security mechanisms described in the previous session. The Java 2 platform security resources are described below

5.1 Security Policy Definition in Java 2

The most important security aspect of Java 2 technology is the SecurityManager class. If an application tries to do an operation that could be harmful to the system (for example, writing to the file system), the SecurityManager will verify if the operation is allowed to the application in question. In the security model of Java 2, the SecurityManager allows the establishment of a security policy that can restrict the operations of a Java program [8].

An application can have, for example, access to the file system but not to the network. It means that the file system is part of the application protection domain, but the network is not. The protection domains are composed by permissions. Each permission represents access to a system resource. If an application running under the SecurityManager tries to access some system resource, the permission associated to that resource has to be explicitly granted.

The permissions associated to each application or entity is specified in special files called policyfiles. A policyfile can be generated using a single editor or the policytool, included in the J2SDK. A policyfile can define permissions associated to an entity or URL.

5.2 Digital Signatures in Java 2

Java has the tools keytool and jarsigner and a security API to provide digital signatures. The API provides the signature and verification of data, while the tools provide digital signature for applications including certificates. If used together, the API and tools guarantee the authentication of an application owner [9].

Keytool manages keys and certificates that are stored in a keystore file are protected by a password. There are two kinds of entries in a keystore associated to an alias:

- *Key pair entry*: it includes the private key and the public key certificate. It is used to sign applications and embed the public key certificate in the signed application.
- *Valid certificates entry*: A certificate is valid if the keystore owner trusts its public key. A valid certificate is necessary if the keystore owner wishes to receive data/applications from entities and authenticate them.

Certificates can be exported and imported. Export a certificate means to extract it from the keystore and send it to an entity that needs to authenticate a public key. The entity that received the certificate has to contact its owner and verify its authenticity. Only valid certificates should be imported to the keystore.

Jarsigner generates and verifies digital signatures to applications embedded in a JAR (Java Archive) file. The JAR format joins several files in one. The JAR file has a manifest file containing one session to each file that composes the JAR. The digital signature is generated from these sessions.

The private key and its public key certificate have to be stored in a keystore, so the JAR can be signed. When a signature is generated, two files are added to the JAR: the signature file (.SF) and the signature block (.DSA). Figure 2 shows the generation of a digital signature in a Jar file.

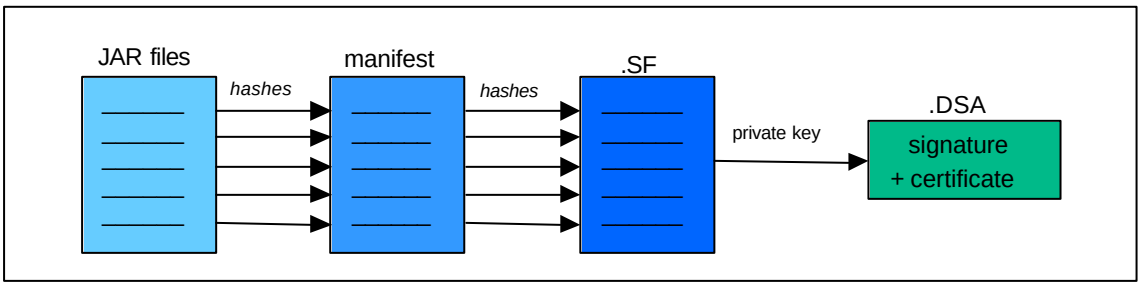

Figure 2 – JAR file signature generation

The JAR verification succeeds if the signature is valid and if the JAR file was not altered while in transit. The verification process done by the jarsigner tool does not include the certificate verification. To do so, the referred certificate should have been imported to a keystore. The Java 2 API has methods to extract the JAR certificate and compare it to the certificates in a keystore. This scheme guarantees the public key authenticity.

The signature does not guarantee the JAR file confidentiality. It may be intercepted while in transit and its contents may be read, but not modified. A solution to this problem is the Java Cryptography Extension (JCE). It is an additional package used to encrypt and decrypt data.

6 SIUDS Implementation

Figure 1 presents a general view of the SIUDS. Its modules are better described in this session.

6.1 The Gateway agent

The mobile agents will join the Pool only if the Gateway authenticates them. Its functions are to provide an entry to all the other agents into the Pool, to verify their digital signature and to check if the certificate of the agent owner is trustful. To be authenticated by the Gateway, an agent has to be properly signed and its certificate must have been imported to the SIUDS keystore. To fulfill its role, the Gateway is composed by three basic classes: ServerFPT, JarVerifier and CertificateVerifier.

The mobile agents will use the class ServerFTP to establish a connection to the Gateway. ServerFTP will transfer the agent from the client file system to the local file system. After the agent is received, the class JarVerifier verifies if the agent's code is properly signed. The verification is done by the jarsigner, described in session 5.2. The class CertificateVerifier does the certificate verification. It gets the public key certificate for the agent and extracts its alias (the signer's name). Then, the CertificateVerifier checks if that alias is stored at SIUDS keystore. If so, the agent's certificate is compared to the certificate stored at the keystore for that alias. If both are equal, the signer certificate had already been imported to the keystore and accepted as trustful. Then, the Gateway can inform the Pool that there is an agent ready to run.

6.2 The SecurityManager

If a mobile agent is running in the Pool it can access all Java platform resources. It can, for example, write to the local file system and establish connections to remote hosts. However, it would not be safe if every agent running in the Pool had unrestricted access to the platform resources. The SecurityManager class has to be installed to avoid unsafe actions to the system.

The SecurityManager allows applications to implement a security policy. To do so, explicit permissions have to be granted in one or more policyfiles or a subclass of SecurityManager has to be implemented. The second choice is harder and error prone.

The SIUDS prototype uses a policyfile to provide security. The SIUDS policyfile has two defaults entries, the first is associated to the *superusers* group and the second is associated to the *commonusers* group. The *superusers* have all permissions while *commonusers* are able to read and write some kinds of files and establish connections to some hosts. When the SecurityManager is installed, the permissions in the policy file start to work. An agent can access a resource only if the permission associated to that resource has been granted to its group in the SIUDS policyfile.

6.3 The Pool of Agents

The Pool of Agents runs the mobile agents and controls their lifetime. To achieve its goals the Pool of Agents is composed by three main classes: StartUp, JarRunner and TimeCounter.

The class StartUp is the first one to be executed. Its first action is to run the Gateway agent. The JarRunner class installs the SecurityManager and runs the agents. An agent will execute in a thread started by JarRunner only if the Gateway authenticates its. The TimeCounter class controls the lifetime of the agents and is called by the JarRunner every time an agent starts running. An agent lifetime can vary from seconds to years depending on the group it belongs to.

6.4 The Database Interface

The Database Interface is a Java interface with methods that allow the agents to manipulate the SIUDS groups and objects in the database and to establish associations between objects and groups (to control which groups can read or write specific objects). The Database Interface, as all Java interfaces, does not contain the implementation of any method. It just defines parameters and the return types.

Two types of BDMS could be used to store the objects of a universal data server: a OODBMS, like ObjectStore or Jasmine, or a ORDBMS, like Oracle or Informix. To use the SIUDS with one of these DBMS the Database Interface methods have to be implemented by a Java class. The JDBC (Java Database Connectivity) package can be used to establish the connection with some of the databases.

All the information that the SIUDS uses is stored in the database that it interfaces. It guarantees that the SIUDS is as robust as the DBMS it works with. A database for the SIUDS prototype was implemented in memory and an mobile agent, the InfoAgent, was created, both for test purposes.

7 The InfoAgent: a test agent

The InfoAgent was implemented to be authenticated and executed by the SIUDS. Its function is to provide information about the SIUDS state. It opens an Internet connection and waits requisitions from web browsers. When it receives a requisition it sends a HTML page containing information about the SIUDS, such as the names of the stored groups, the names of the agents in execution and which group each agent belongs to.

The InfoAgent has all the requirements necessaries to be executed by the Pool of Agents. It is a signed JAR and its owner is in the *superusers* group. This group is already stored in the database and has a valid certificate at the SIUDS keystore. As described in session 6.2, by default the *superusers* and *commonusers* public key certificates are imported to the SIUDS keystore and their permissions to systems resources are granted in a policyfile.

The InfoAgent is packed in a signed JAR file and sent to the Gateway. Its signature verification succeeds because the InfoAgent contents were not altered while in transit. Then the Gateway extracts the InfoAgent's signer name and verifies if there is a valid certificate in the SIUDS keystore. As the *superusers* certificate has already been imported to the SIUDS keystrore, the Gateway successfully finishes the InfoAgent verification, after that the InfoAgent is finally executed in the Pool.

Other mobile agents were use to test the certificates verification. Only agents that had the *superusers* or *commonusers* signature were successfully verified by the Gateway. Tests were conducted to check if the Pool correctly executed the mobile agents (concurrently) and if it was able to control their lifetime. The *superusers* agents can run indefinitely, while the *commonusers* agents have only one hour of execution. The tests confirmed the concurrent execution of agents and the Pool finished the *commonusers*' agents after their one-hour time.

8 Conclusion

The SIUDS provides an open and secure environment for the execution of mobile agents. The agents can use system resources to establish connections to remote resources. The agents can also store and retrieve objects from the database. The access to the objects (in the database) and to the system resources is controlled. An agent can access an object or resource only if its group has the permissions to do so.

The SIUDS together with an object-oriented (or object-relational) database provides an environment where mobile software agents can be loaded, run and access data, securely forming the bases of a universal data server. Moreover, the universal servers implemented using the SIUDS can use agents written in Java as their query language. A language that is more complete and efficient for object-oriented data query than traditional query languages.

References

[1] J. R. Davis. "Universal servers: the players, part 2", *DBMS*, vol. 10, no. 8, pp75-81, 1997.

[2] V. Krishnamurthy, S. Banerjee and A. Nori, "Bringing Object-Relational Technology to the Mainstream", *ACM Sigmod Record*, vol. 28, no. 2, pp. 513, June 1999.

[3] S. Franklin, and A. Graesser. "Is it an agent or just a program?: a taxonomy for automonous agents", *Proc. Third International Workshop on Agent Theories, Architecture and Languages*, Institute for Intelligent Systems, University of Memphis, 1996.

[4] D.A. Moreira and L.T. Walczowski, "Using Software Agents to Generate VLSI Layouts", *IEEE Intelligent Systems*, vol. 12, no. 6, pp. 26-32, November/December 1997.

[5] D. Wong, N. Paciorek and D. Moore, "Java-based mobile agents", *Communications of the ACM*, vol. 42, no. 3, pp. 92-102, 1999.

[6] S.K. Chin, "High-confidence design for security", *Communications of the ACM*, vol. 42, no. 37, pp. 33-37, 1999.

[7] G. Coulouris, J. Dollimore and T. Kindberg, *Distributed systems: concepts and design*, 2. ed. Addison-Wesley, Great Britain, 1994, 644p.

[8] L. Gong, *Java security architecture*, October 1998, available over the Internet at http://www.java.sun.com/products/jdk/1.3/docs/guide/security/spec/security-spec.doc.html. [online 17/03/2000].

[9] M. Dageforde. *The Java tutorial: trail - security in Java 2 SDK 1.2*. Available over the Internet at http://web2.java.sun.com/docs/books/tutorial/security1.2/index.html. [online 18/03/2000].