ISSN: 2321-7782 (Online)
Volume 4, Issue 2, February 2016
International Journal of Advance Research in Computer Science and Management Studies
Research Article / Survey Paper / Case Study
Available online at: www.ijarcsms.com

A Comparative Study of Security Architectures for Distributed Systems

Ranjit Kaur
Assistant Professor: Department of Computer Science and Applications
Kanya Maha Vidyalaya
Jalandhar - India

Abstract: This paper provides a comparative study of the four popular architectures for distributed systems applications, their security implications and the issues associated with them. The architectures analyzed are Globus, CORBA by the OMG, COM+ from Microsoft and Java by Sun. When only individual systems need to be protected, such as one computer with all files residing locally and with no need to connect to any outside resources, security is not as complex as with distributed systems. Security can have varying levels of difficulty for implementation in distributed systems. With distributed systems architecture, there are different nodes and resources. One major issue with distributed systems is application security. There is the question of how security is handled in distributed applications, and how the client handles applications coming from an unknown source. It is extremely important for developers to consider the security implications when designing distributed applications, as many of these applications offer access to crucial resources: financial, medical, and military information etc.

Keywords: COM+, CORBA, Distributed Security, Globus, JAVA.

I. INTRODUCTION

Security in distributed systems is an important issue. Security in a computer system is strongly related to the notion of dependability. Dependability includes availability, reliability, safety, and maintainability. Several elements of distributed system security are identified, like authentication, authorization, encryption and system protection. In initial days, the security management environment was based on single authority systems but now the focus is on the development of per activity, authorities and groups with shared responsibilities. However, if we are to put our trust in a computer system, then confidentiality, integrity and availability **(CIA triad)** should also be taken into account. Confidentiality of information, integrity of information and availability of information.

Confidentiality is protecting the information from disclosure to unauthorized parties. Information has value, especially in today's world. Bank account statements, personal information, credit card numbers, trade secrets, government documents. Everyone has information they wish to keep a secret. Protecting such information is a very major part of information security. The ways to ensure information confidentiality include encryption, enforcing file permissions and access control list to restrict access to sensitive information.

Integrity of information refers to protecting information from being modified by unauthorized parties. Information only has value if it is correct. Information that has been tampered could prove costly. As with data confidentiality, cryptography plays a very major role in ensuring data integrity. Commonly used methods to protect data integrity includes hashing the data you receive and comparing it with the hash of the original message. However, this means that the hash of the original data must be provided to you in a secure fashion.

Availability of information refers to ensuring that authorized parties are able to access the information when needed. Information only has value if the right people can access it at the right times.

The CIA triad is a very fundamental concept in security. Often, ensuring that the three facets of the CIA triad is protected is an important step in designing any secure system.

The general security attacks on the distributed systems are eavesdropping (gaining secret information), masquerading (making assumptions on the identity of users), and message tempering (changing the content of the message), replaying the message and denial of services.

Computer security frequently consists of two parts: authentication and access control.

Authentication involves the verification and identification of a valid user.

Access control strives to prevent unwanted tampering with data files and system resources. In an isolated, centralized, single-user system such as a PC, security is accomplished by locking up the room where the computer is stored and locking up the disks. Thus, only the user with the key to the room and disks may access the system resources and files. This accomplishes both authentication and access control. The security is only as good as the keys that lock the computer and the room. In isolated centralized systems with multiple users, security is a bit more complex. In this scenario, authentication involves the verification and identification of a valid user and is usually managed by some type of password and user identification combination. Control of access to the files, data, and resources to prevent unwanted tampering is generally accomplished through capabilities or access lists and is managed as part of the operating system.

Before discussing the architectures for distributed systems and their security implications an overview of distributed system architecture is presented.

II. Distributed Systems

Researchers defined different definitions for the distributed system. Colourise et al., have defined a distributed system as "a system where the hardware and software components have been installed in geographically dispersed computers that coordinate and collaborate their actions by passing messages between them". When there are some systems works in the collaborative manner with other system for the purpose of communicating each other for, processing data transfer, data storage etc. these systems may be scattered by distance it is commonly known as a distributed system. According to these definitions given by researchers a distributed system have been built with the objective of attaining the following.

- Transparency
- Openness
- Reliability
- Performance
- Scalability

To achieve these main objectives of distributed system attention must be pay on a fundamental requirement of system i.e. security at every stage of designing, implementation, operation and management by the user.

2.1 Why is it so difficult to implement security on distributed systems?

➢ **Resources can span many geographic domains.**

The system must keep track of where its resources are located

<u>Ex</u>: An employee on subnetwork A in the U.S. wishes to access a file stored on the company server on subnetwork C located in Germany.

- **For some resources, only certain classes of users can be allowed access.**

  Falls under the authorization domain.

  <u>Ex</u>: Suppose we have two low-level employees, John and Sam. John should not be able to access Sam's employment record even though they both work for the same company. However, Sam's supervisor should have access rights.

- **Most distributed systems are heterogeneous**

  They are comprised of mixed types of computers and resources.

  Need to define protocols to safely access system components no matter where such components reside.

III. Distributed Systems Security Architecture

The four most popular architectures for the distributed systems that are developed by keeping security requirements of distributed systems in mind are:

- Globus
- Java
- CORBA
- COM+

3.1 Globus

Globus is used in large scale distributed computations where many hosts, files and other resources are simultaneously used for computation. Resources in such environments are often located in different administrative domains that may be located in different parts of the world. Because the resources are vast and spread in different domains, security is essential.

<u>The security policy for Globus includes following eight statements:</u>

- **The environment consists of multiple administrative domains.**

  Multiple existing disjoint subsystems (most likely heterogeneous) need to be seamlessly integrated under a minimal number of standardized security protocols.

  <u>Ex:</u> Suppose that companies A, B, & C, offering the same service, have merged after independently operating for years. Now suppose that the 3 companies have merged and wish to integrate their websites.

  Most likely, each website is managed by a different security policy and has different critical resources.

  The goal is to integrate the subsystems in such a way so that membership in the global union does not override the local security policy of any subsystem.

  Thus, the global authority need only consider actions that span multiple subsystems.

- **Local operations (i.e., operations that are carried out only within a single domain) are subject to a local domain security policy only.**

  Complements the first assumption.

  <u>Ex:</u> Suppose that an entity in domain B wishes to utilize a file housed in domain B.

- Local operation.
- Just as if domain B was running independently of Globus and any other subsystems.
- Globus should recognize local operations and not impose further restrictions.

- **Global operations (i.e., operations involving several domains) require the initiator to be known in each domain where the operation is carried out.**

  Operations that affect more than 1 domain in the distributed system.

  Require the initiator to be known in each domain where the operation is executed.

  Ex: If Adam wishes to update a file located in domains A, B, and C, each domain must authenticate him before he can be allowed to perform the update.

- **Operations between entities in different domains require mutual authentication.**

  Requires mutual authentication by both domains.

  Extends the previous statement in both directions.

  Ex: Suppose Adam located in domain A wishes to use a mail server located in domain B.

  Not only must Adam be authenticated by domain B to use the mail server, but the mail server must also be authenticated by domain A.

  The latter condition ensures Adam that he is indeed using the mail server from domain B and not malicious software.

- **Global authentication replaces local authentication.**

  With many users, authenticating each resource request locally makes scalability impossible.

  Global authentication by Globus supersedes local authentication.

  Improves performance of the distributed system.

  Ex: Suppose Adam located in domain A wants to perform operations in domains B and C.

  Adam must be authenticated in domains B and C.

  Instead, allow Globus to globally authenticate Adam.

  Adam can now be considered authenticated in domains B and C if both domains recognize Adam.

  Recognition takes less processing time than authentication.

- **Controlling access to resources is subject to local security only.**

  Subject to local security only.

  Globus would become overwhelmed if it had to track the security status of the numerous resources.

  Ex: Suppose Adam located in domain A wishes to modify a file located in domain B and has already been globally authenticated.

  Even though Adam has been authenticated, his access rights to the file still must be checked.

  Access rights are checked by the file’s local domain – in this case domain B.

- **Users can delegate rights to processes.**

  A user should be able to copy his/her authentication and access rights to created processes.

  Ex: Suppose that Adam located in domain A has been globally authenticated. He wishes to deploy processes P_1 and P_2 to repeatedly poll files located in domains B and C respectively.

  Adam can pass his authentication certificate to processes P_1 and P_2.

  P_1 receives access rights for the files needed in domain B and P_2 receives access rights for the files needed in domain C.

  Globus globally authenticates the processes (verifying that they are from Adam).

  The processes can then begin working in their respective domains as if they were Adam himself.

  Efficient for processes that initiate many remote operations.

  Otherwise, domain B would have to contact Adam for authentication every time that P_1 initiated an operation.

  The communication cost for this persistent validation may prove to be prohibitive especially if domains A and B are great distances apart.

- **A group of processes in the same domain can share credentials.**

  Multiple processes attributed to a single user located in a remote domain can share one set of authentication and access rights.

  Known as credentials.

  Ex: Suppose Adam located in domain A has multiple processes P_1, P_2, and P_3 operating in domain B.

  The 3 processes share a set of credentials.

  Credentials are easily modified even if there are a large number of processes.

  Easily scalable – space is conserved since only one copy of the credentials need be present per domain.

The Globus security policy allows its designers to concentrate on developing an overall solution for security. By assuming each domain enforces its own security policy, Globus concentrates only on security threats involving multiple domains. Globus primarily needs mechanisms for cross-domain authentication, and making a user known in remote domains. The Globus security architecture essentially consists of entities such as users, user proxies, resource proxies, and general processes. These entities are located in domains and interact with each other. The security architecture defines four different protocols, as shown in Fig 1.

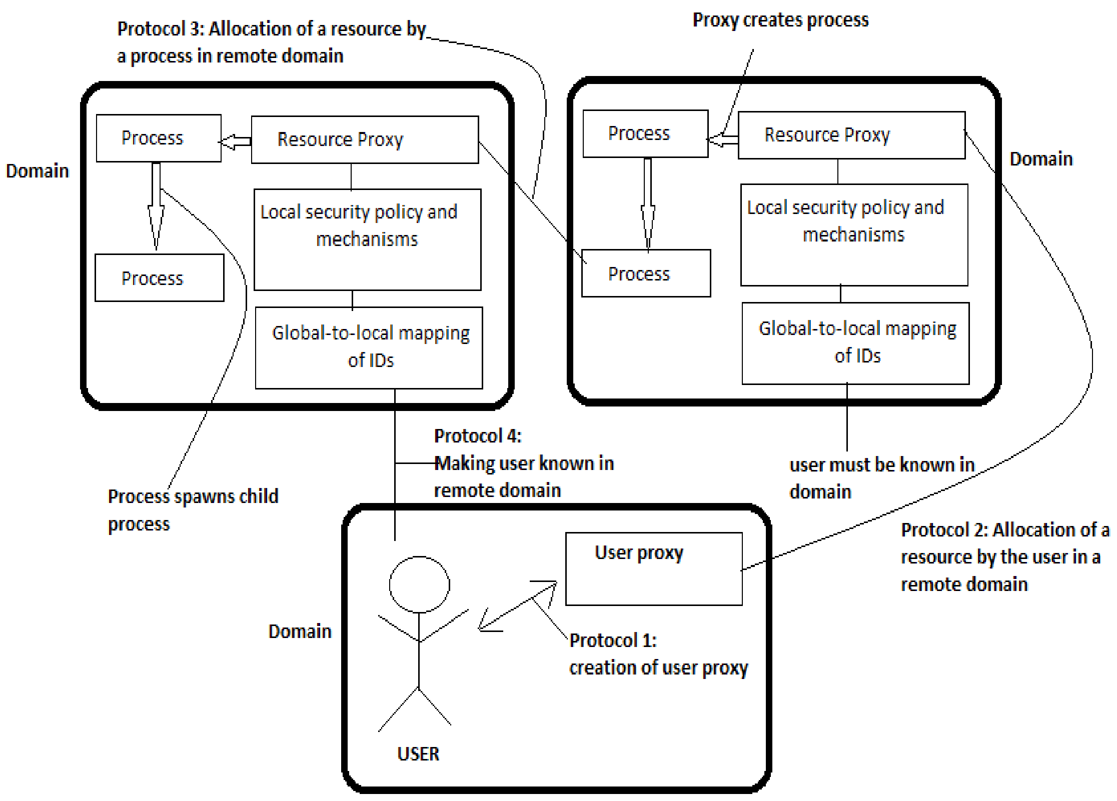
Protocol 3: Allocation of a resource by a process in remote domain
Proxy creates process
Domain
Process
Resource Proxy
Process
Local security policy and mechanisms
Global-to-local mapping of IDs
Domain
Process
Resource Proxy
Local security policy and mechanisms
Process
Global-to-local mapping of IDs
Protocol 4: Making user known in remote domain
user must be known in domain
Process spawns child process
Protocol 2: Allocation of a resource by the user in a remote domain
User proxy
Domain
Protocol 1: creation of user proxy
USER

Fig 1. The Globus security architecture

Globus architecture is described using entities:

- Users
- User proxies: processes that are given permission to act on behalf of a user temporarily.
- Resource proxies: processes used to translate a remote user’s requests into operations that do not violate a resource’s local security policy.
- General processes

Globus has defined four protocols using entities.

The first protocol (Creation of a User Proxy & Delegating User Rights): describes how a user can create a user proxy and delegate rights to that proxy. In order to let the user proxy act on behalf of its user, the user gives the proxy an appropriate set of credentials.

- First, a process is created by the user in his/her local subsystem.
- To act as the user, the process must be given an identifying key linked to the user.
    - This key is a tuple comprised of the user’s id, name of the local host, authentication lifetime, etc.
    - The user then digitally signs the tuple, indicating the validity of the proxy to remote domains.
- The proxy is then provided with the key and allowed to execute in a remote domain.
    - It is the responsibility of the local security policy to protect this key.

The second protocol (Requesting the Allocation of a Remote Resource): specifies how a user proxy can request the allocation of a resource in a remote domain.

- A user proxy (UP) locates the resource proxy (RP) for the resource it wishes to access.
- The UP and RP authenticate each other.
- The RP checks if the resource is available.
    - If the request can be honored, the RP allocates the resource to the UP.
    - Otherwise, the RP denies access and it is up to the UP to try again after some specified passage of time.

The third protocol (Allocation of a Resource by a Remote Process): tells a resource proxy to create a process in the remote domain after mutual authentication has taken place.

- Essentially follows the second protocol.
- **Ex:** A resource allocated to a UP may spawn process P_α that requires an additional resource R_1.
    - The request to R_1's RP must come from the UP, NOT P_α.
        - Advantages: Simplicity and greater security.
            - Allowing only UPs to initiate resource requests decreases the potential locations for security breaches.
        - Disadvantage: scalability is limited due to a single point of requests (UP).

The fourth protocol (Recognizing a User in a Remote Domain): in the Globus security architecture is the way a user can make himself known in a domain. Assuming that a user has an account in a domain, what needs to be established is that the system wide credentials as held by a user proxy are automatically converted to credentials that are recognized by the specific domain. The protocol prescribes how the mapping between the global credentials and the local ones can be registered by the user in a mapping table local to that domain.

- First, the user authenticates globally with Globus.
- A subsystem-wide mapping of the global authentication to a local authentication is accessible by the RP.
- The exact implementation varies with each domain. Examples include trees and linked lists.
- The user is considered authenticated for domain X if the RP can find a mapping from the user's global authentication to his/her local authentication for domain X.
    - The remote domain thus recognizes the user.

Issues

Globus security architecture reflects its security policy according to the above eight statements and protocols. The mechanisms used to implement that architecture, particularly the protocols are common to many distributed systems. Important implementation issues are: How a user is represented in a remote domain and How remote resources are allocated to users.

3.2 Java

The Java architecture was designed for the distributed systems keeping security requirements in mind. The need to create programs that are executed on remote distributed systems was accomplished through the Java architecture. The source code is written and then converted to byte code and is stored as a class file, which is interpreted by the Java Virtual Machine (JVM) on the client. Class loaders then load any additional classes that are needed by the applications.

Several security checks are put between the remote server distributing the program, and the client executing it, such as the "sandbox" security model, the byte code verifier, the applet class loader, the security manager, and through other security measures that can be implemented through Java's security APIs.

Sandbox Security Model

Sandbox security model was developed to run all applets in a protected form. The end users of distributed systems determine which applets to run on their systems. Most of the users are not capable to agree on whether a particular applet is worthwhile or not. Applets that run from a remote site would be permitted only limited access to the system, while code run locally would have full access. If the applet is signed and trusted, then it can run with full local system access. Permissions can be set by a security policy that allow the administrator to define how the applets should be run.

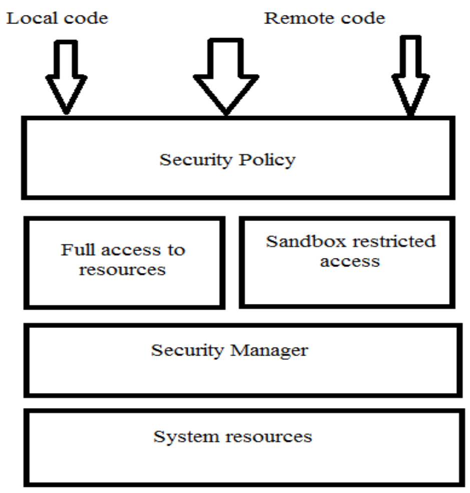

Fig 2. Java model

Byte Code Verifier

The byte code verifier looks at the class files that are to be executed and analyzes them based on specific checks. The code will be verified by three or four passes depending on whether or not any methods are invoked. Some of the checks are performed to ensure that the proper format is used for the class, to prevent stack overflow, to maintain type integrity, to verify that the data does not change between types, and that no illegal references to other classes are made. The byte code verifier ensures that jumps do not lead to illegal instructions, that method signatures are valid, access control, initialization of objects, and that "subroutines used to implement exceptions and synchronized statements are used in FIFO order".

Applet Class Loader

As a Java application is executed, additional classes may be called. When they are called the applet class loader loads the specified applets. Classes in Java are organized by name spaces, and each class loader is responsible for one name space. The class loaders are therefore responsible to protect the integrity of the classes in its name space. Java has built-in classes that reside locally, however, that are loaded automatically without any security checks. The path to these classes is indicated by the CLASSPATH environment variable.

Security Manager

In order to protect variables and methods from being modified by classes, classes are grouped into packages. Depending on the package that a class belongs to, the class will have different access to the other classes in the package, so security could be compromised if an unauthorized class attaches itself to the package. The security manager makes sure that only classes that actually belong to the package in question are able to declare themselves in this package. Browsers and applet viewers have a security manager, but by default Java applications do not. Java has provided developers the means to create their own security manager.

API Security

Java provides security through several security APIs. Among the different APIs provided, the developer can make use of signed applets, digital signatures, message digests, and key management. When an applet is signed it is given full access to the system as if it were run locally. The default Java Runtime environment provides digital signatures, message digests, and key management, and encryption can be implemented through the Java Cryptography Extension (JCE).

Issues

Java distributed architecture contains several outstanding security problems. One problem is with the CLASSPATH system environment variable. CLASSPATH variable is used to determine the location of the built-in Java system classes. If the CLASSPATH variable is altered, it could point to a set of altered classes that may execute what the original classes intended, but also insert malicious code. Another problem is the degradation of service attack, although the security policies can be created still this does not prevent the applet from consuming sensitive resources such as CPU and memory. There are several Java vulnerabilities if a computer serving Java applications is either compromised from the inside, or if an attacker is able to compromise an account on the server. The vulnerabilities are due to design choices rather than software defects. The problem of auditing is also there in java architecture. There is no known work presently being done to implement auditing capabilities in Java

3.3 CORBA

CORBA is a standard defined by the OMG (Object Management Group). It describes an architecture, interfaces, and protocols that distributed objects can use to interact with each other. It is a standard architecture for distributed object systems. It allows a distributed, heterogeneous collection of objects to interoperate. Part of the CORBA standard is the Interface Definition Language (IDL), which is an implementation-independent language for describing the interfaces of remote objects. Any application from any vendor, on almost any computer, operating system, programming language, and network, can interoperate with a CORBA based program from the same or another vendor, on almost any other computer, operating system, programming language, and network. CORBA-based applications interact with objects. Information is passed to the IDL, which in turn brokers it to the correct object, which interprets the information, and sends any requested information back to the caller through the IDL. The central part of a CORBA implementation is the Object Request Broker. This serves as the 'object-bus'. Each object instance has a unique entry in the Object Request Broker (ORB) which handles requests between objects and between user and objects. Objects with similar security requirements are grouped into domains, and a security policy is applied to the domain, which is enforced by the ORB. Communication between ORBs is handled by "bridges, gateways, and inter-ORB protocols like the General Inter-ORB protocol (GIOP) and the Internet Inter-ORB Protocol (IIOP)".

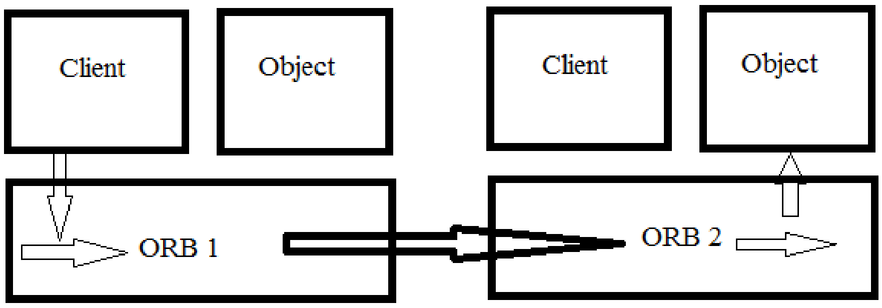

Fig 3. CORBA model

There are several different security aspects that are covered in the CORBA specification.

Key security features of CORBA include:

- Identification and authentication
- Authorization and access control
- Security auditing
- Non-repudiation
- Administration.

Identification and Authentication

It verify that the user really is who they claim to be. A user will authenticate by using password. This is used as a means for accountability, for access to objects with different permissions, for identification of the principal sending information, for controlling access to different objects, message signing, and usage charging for object implementation.

Authorization and Access Control

As users are authenticated, the applications will use those credentials to access other objects through the ORB, where the CORBA security service operates. A security policy defines what objects the user has been given access to and through the policy's implementation at the ORB level, access is either granted or denied.

Security Auditing

It is the key component to security. Auditing enables the administrator to detect intrusions, attempted intrusions, or other security anomalies. There are two types of auditing in CORBA: system and application. Which events are to be logged is determined by the respective audit policies. System policies could include the logging of events such as the authentication of principals, when privileges change, whether the invocation of an object was successful or not, and events relating to administration. The application-level auditing could be specific to the application, such as the auditing of specific transactions.

Non-Repudiation

It ensure that the user (principal) is held accountable for their actions. Evidence is maintained that will either prove or disprove a particular action. CORBA provides for non-repudiation of creation and non-repudiation of receipt, the former proving whether or not a principal created a message, and the latter whether or not a principal received the message.

Administration

Security is administered in CORBA through the use of domains, which refer to the scope or boundaries of the items being examined, grouped by some commonality. There are three security domains: security policy domains, security environment domains, and security technology domains. In security policy domains, certain items must be administered: the domains themselves, the members of the domain, and the policies associated with the domains. The environment domains refer to the "characteristics of the environment and which objects are members of the domain". Technology domain administration may refer to establishing and maintaining the security services, the trusts between the different domain, and any other entities such as principals and keys, that would be within the scope.

Issues

There is inconsistency among the different security models for the applications being integrated through CORBA, there may be an opportunity for an intruder to compromise the architecture's security. Koutsogiannakis and Chang (2002) state that "A Corba implementation is labor- and resource-intensive". Since, security is based on all requests being brokered through the ORB. It does not guarantee that the ORB cannot be bypassed and that the data used by CORBA's security services are properly protected. Also, Non-repudiation is not at the ORB level, and has to be implemented at the application level, can weaken the security.

3.4 COM+

COM+ represents the next generation in Microsoft's history of distributed architectures.COM+ called for multi-tier programming where the clients would run an application that would connect to a server with COM+ services running. This server would in turn connect to the back-end servers. This offered many benefits, such as sharing of resources on the COM+ server, and limited updating of clients. Various languages can use the COM+ architecture, such as C++, Visual Basic, Java, Delphi, and COBOL.

COM+ includes automatic security. This makes the code easier to write and maintain, it is easier to design security at a higher level and throughout an entire application, and it facilitates the configuration of a security policy.

COM+ includes the security features such as:

- Role-based security
- Impersonation and delegation
- Software restriction policies.

Role-Based Security

It is an automatic service of COM+ and can be extended in order to construct and enforce access policies. Role-based security can be implemented either as declarative or programmatically. In declarative security, permissions can be set on components much the same way that permissions are set on files within the Windows NT operating system. Therefore there is no need to recompile code during configuration of security. If the permissions need to be more granular than at the component level, then role-based security must be implemented programmatically.

Impersonation and Delegation

When a client accesses a resource, often the server that the client is connecting to needs to retrieve information with the client's credentials, ensuring that the client can only access information that it has been granted rights to. This is done in COM+ through impersonation. Delegation refers to the impersonation of a client over the network. If a client is running an application that makes a call to the COM+ middleware server, and it in turn needed to access a SQL database, delegation would be used.

Software Restriction Policies

Windows XP is the proactive framework of software restriction policies. The access is determined by setting one of two trust levels: unrestricted and disallowed. Unrestricted will allow the code to execute up to the limits given to the user executing the code, whereas disallowed is restricted to the sandbox. In role-based security, software restriction policies can either be set through a graphical user interface (GUI) or programmatically.

Issues

The marked absence of evaluation of the COM+ architecture and its current security problems is the main issue of COM+. It's very difficult to recognize the problems that may exist in the COM+ architecture. Also it does not appear that Microsoft's sandbox allows granular enough permissions. Only two levels of trust can be configured, which would appear to be inadequate for most implementations.

IV. Conclusion

In this paper four popular architectures for distributed systems security were discussed such as Globus, Java, CORBA, and COM+. Each of these architectures has strengths and weaknesses. There are several aspects that need to be considered while deciding which architecture to employ for the distributed systems. Firstly the level of security that the architecture provides must be considered. Java has several vulnerabilities but still it is most popular among the other three. Globus and CORBA do not have many, they have not been as widely implemented as java. Security for COM+ may be inconclusive as there is not a wealth of information on it. The difficulty of implementation must also be considered. If the system is overly complex, security problems may exist due to implementation problems. If the architecture is too simple however, there may not be enough flexibility to create the necessary security configurations. Finally, the environment needs to be taken into consideration. Java security architecture is recommended when the systems that need to communicate are primarily Java-based, Globus when there not so many users to use the system, COM if the environment is mostly Microsoft based in order to take advantage of close integration with the other Microsoft products, and CORBA as a general implementation.

References

1. Andrew S Tanenbaum and Maarten van Steen, Distributed Systems: Principles and Paradigms, 2nd ed. Upper Saddle River, NJ, USA: Pearson Higher Education, 2007.
2. George Coulouris, Jean Dollomore, and Tim Kindberg, Distributed Systems – Concepts and Design, 4th ed. London, England: Addison-Wesley, 2005.
3. April L. Moreno, “Distributed Systems Security: Java, CORBA, and COM+”.
4. Mohamed Firdhous, “Implementation of security in distributed systems”, International journal of computer information systems Vol. 2, No. 2, 2011.
5. Vijay Prakash, Manuj Darbari, “A review on security issues in distributed systems”, International journal of scientific and engineering research volume 3, issue 9, September-2012.
6. N. De Palma, D. Hagimont, F.Boyer, L. Broto, Self protection in a clustered distributed systems, IEEE Transactions on Parallel and Dis-tributed Systems, vol. 23, no. 2, 2012, pp. 330-336.
7. L. Qi, L. Yu, Mobile agent based security model for distributed sys-tem, 2001 IEEE International Conference on Systems, Man and Cy-bernetics, vol. 3, pp. 1754-1759, 2001.
8. Y. Zhao, N. Thomas, Computing methods for efficient analysis of PEPA models of non-repudiation protocols, 15th International Con-ference on Parallel and Distributed Systems (ICPADS), 2009, pp. 821-827.
9. Demissie B. Aredo, Sule Yildirim, “SECURITY ISSUES IN ADAPTIVE DISTRIBUTED SYSTEMS”.
10. Manu Agarwal, Gaurav Agarwal, “Accessing the Data Security Model in Distributed System”, International Journal of Applied Information Systems (IJAIS), Foundation of Computer Science FCS, New York, USA Volume 1– No.4, February 2012.
11. Morrie Gasser, Andy Goldstein, Charlie Kaufman, Butler Lampson, “The digital distributed security architecture”.
12. Chizmadia, D. (1998). A quick tour of the CORBA security service. Retrieved from http://www.itsecurity.com/papers/corbasec.
13. COM+ security programming part 1: declarative role-based security retrieved from http://www.itworld.com/nl/windows sec/06112001/
14. Object Management Group. CORBA basics. Retrieved from http://www.omg.org/gettingstarted/corbafaq

Author(s) Profile

Ranjit Kaur, is Assistant Professor in department of Computer Science and Applications, Kanya Maha Vidyalaya, Jalandhar, Punjab. She has received the M.Tech degree in Computer Science & Engineering from Lovely Professional University, Phagwara in 2014.