Global Information Assurance Certification Paper

Distributed Object Technology: Security Perspective

Subbu Cherukuwada
GIAC Security Essentials Certification (Version 1.3)

ABSTRACT

Distributed object technology makes object oriented programming even more powerful and efficient by making use of the objects that are available on different systems connected on a heterogeneous network, in addition to the locally defined objects. Wide range of hardware platforms and variety of operating systems can be inter-connected at the software level and deliver a more robust and comprehensive solution for today's Internet driven businesses. Internet itself plays an important role as a backbone for this technology. As the scope of the application code is not restricted to single source, platform or language, maximum attention must be paid to the security of the applications. The objective of this paper is to give a brief introduction to distributed object technology and an overview of security features available in Microsoft.NET and CORBA. The paper explains the architecture of .NET and covers some of its key security concepts like Security Policy, Code Access Security, Role Based Security, Verification and Stackwalk. It also explains CORBA and its security concepts like CORBA Security Services, Security Specifications, Security Policy, Domain Access Policy and Delegation. The paper concludes by explaining the way in which some key security concerns are addressed in .NET and CORBA.

1.0 INTRODUCTION

Distributed computing brings together the power of different hardware platforms and operating systems to deliver high performance applications. Object oriented programming modularizes the software development and improves the optimum usage of system resources. Distributed object technology is the combination of these two technologies that complements each other's features and provides the scalable platform to develop efficient and faster applications. Following are some of the most important objectives of this technology:

- **Providing framework of standard services and libraries**
  Most important methods that can be used in various kinds of development environments must be readily available for usage. A set of guidelines should be provided which describe the interfaces to these kinds of services and libraries. More importantly this kind framework should be easily extendable i.e. adding more services and libraries should be possible.

- **Platform independence**
  An application should be able to use the objects developed and available on different kinds of platforms. Code developed using this technology should work on all kinds of platforms.

- **Hardware independence**
  Application should work on different kinds of hardware platforms. Integration of objects available on different kinds of hardware platforms should be possible.

- **Language interoperability / independence**
  Developers should be able to use a programming language of their choice, selection based on the ability of the language to perform different tasks. Methods developed in one programming language should be exportable to other programming languages.

Thus, distributed object technology seamlessly integrates different modules of software developed in more than one programming language and residing on systems of various platforms and architecture.

Security Perspective

Security is an essential requirement at different stages of an application developed using this technology, because:

- Application is depending on the code from the systems connected over the network (could be Internet also). Authenticity and integrity of the code must be verified before allowing access to protected resources. Network transfer should not compromise the data sensitivity.

- Application is developed in different programming languages, so exposures like buffer overflows must be considered very carefully.

- Each module of the application may require varying degree of access to system resources and hence resource access should be granted in a very controlled manner.

Following are the examples of the distributed object technology platforms available today:

- Microsoft.NET
- CORBA
- Java RMI

2.0 Microsoft.NET

.NET is developed by Microsoft. .NET framework provides a platform to develop web services, distributed web based applications and windows applications using distributed object technology. Following concepts helps in understanding .NET architecture:

Managed code

Managed code [2] is the code developed in a programming language that is targeted for CLR (Common Language Runtime, which is explained in Section 2.1.1) at the compilation time i.e. it is made to run on CLR. Any other code is unmanaged code. At present, not all the programming languages are capable of producing managed code. Examples of the languages that can produce managed code: C++ with managed extensions, C#, Visual basic and Jscript. Many other compilers are being developed to produce managed code e.g. APL, CAML, Perl, OZ etc. Managed code can exploit complete functionality of the CLR i.e. it can efficiently manage the resources required for execution or make use of the latest CPU instruction set. More importantly,

while executing a managed code, CLR enforces the security policy. Executing an unmanaged code on CLR bypasses the entire security policy. Managed code is not targeted to specific hardware platform or operating system i.e. the executable will not be in x86 native language or any x-bit operating system native language. It is in Microsoft Intermediate Language (MSIL). MSIL plays an important role in security of .NET framework by making type safety verification easy.

Metadata
Metadata [5] is the data generated by the compiler and placed inside the managed code, that describes various methods implemented, object instances and types declared. CLR performs metadata validation before executing any code as part of its verification. Verification is a security management feature of CLR.

Assembly
Assembly [4] is a group of one or more files that can be executed (as managed code) together to perform certain task or execute an application itself. Assembly is the fundamental building block at which security is enforced i.e. permissions are requested and granted. Each assembly comes with a manifest that describes the contents of the assembly, controls what types and resources are exposed outside the assembly and what are the other assemblies that are required to execute this.

2.1 Microsoft.NET architecture
.NET is comprised of two main components.
- Common Language Runtime (CLR)
- NET framework class library

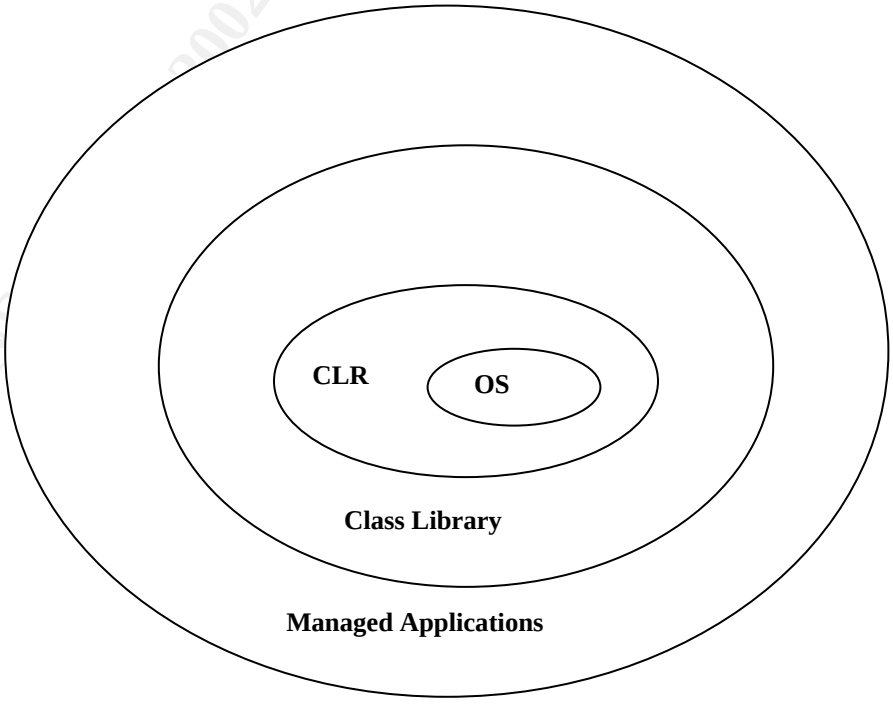

Figure 1: Microsoft.NET Architecture

2.1.1 Common Language Runtime

CLR [9] can be considered as the heart of .NET framework. This is the agent software running on the operating system. CLR isolates the managed code from the native operating system. While executing the code, CLR translates the managed code into native machine language. CLR is capable of executing both managed code and unmanaged code. However, CLR can not offer its complete functionality to unmanaged code. Besides executing the code that is targeted for CLR, it also offers different services like:

- Memory Management
- Thread execution
- Code safety verification
- Security Management
- Just In Time (JIT) Compilation

2.1.2 .NET framework class library

Class library [3] offers the most commonly used services to be readily used into the code built by developers. This library is object oriented and contains various types, classes and methods. Some of the example methods are file access, database operations and string manipulations. Security can be built into the code using the classes provided by this library. Classes that provide similar functionality are grouped into namespaces. Some of the namespaces related to security are

- System.Security
- System.Security.Permissions
- System.Security.Principal
- System.Security.Policy
- System.Security.Cryptography

2.2 Code execution

The fundamental building block of an application is assembly. An application may consist of many assemblies. An assembly will be loaded into an application domain before running an application. Each assembly is loaded into a separate application domain. CLR provides complete isolation among the application domains. Multiple application domains can safely run under the same process.

2.3 Security in Microsoft.NET

.NET security is governed by a security policy, which may be defined as a set of rules that can be configured. This policy decides whether a particular code or user can access a protected resource or not. Some of the resources that can be protected are files, environment variables, clipboard, user interface and registry settings. Access to resources is controlled based on the origin of the code, the user executing the code or both.

The security manager in the CLR maps the evidence to the security policy that is configured on the system. After successful mapping, the security manager determines the allowed permissions to that assembly. A class in the Class Library called Systems.Security.SecurityManager provides this mapping functionality.

Security policy is comprised of three elements:

- Code groups

- Named permission sets
- Policy assemblies

2.3.1 Code group

Code group [12] is the combination of a matching condition and set of permissions. If the code matches a condition, then the code is granted with the associated permission set. Security policy can be visualized as a tree of code groups. In other words, the matching condition is sub-divided into multiple layers. This granulates the matching condition and allows security administrators to set different permission sets at each level of the tree. The tree of code groups starts with a condition called All code with permission set Nothing. Security Manager (integral component of CLR that is responsible for security checks) starts checking from this root condition and traverses down. If a condition is not matched at any level, Security Manager will stop traversing down and moves to the next vertical branch. Figure 2 depicts a sample policy, where the root code group is divided into three branches, based on the zone. Based on the origin of the code, it may get FullTrust, Internet or Intranet permission sets. Following two examples explain the way in which this sample policy is enforced.

Example1: Assume that a user is executing an assembly that is located on the local system. So, the zone is My Computer. Security Manager checks the first level that matches all the code, grants the Nothing permission set and goes to the next level. At this level, the code matches with the condition of the first branch (zones are matched). So the permission set FullTrust is granted. As there are no levels down this code group, security manager returns to the last code group that matched and repeats the same process with the next subordinate group. In this case, it moves to the second condition i.e. Zone: Intranet. This condition is not matched. So the security manager will not traverse down and moves to the next code group. The matching condition for the next code group is Internet, which is also not matched. As it completed all the code groups, security manager calculates the union of all the permission sets that matched the condition, which are:
Nothing (from the condition All Code)
FullTrust(from the condition Zone:My Computer)

Example2: Assume that a user is trying to access an Intranet URL w3.research.com/proj1. Security Manager starts with root node and obtains the permission set Nothing. First code group in the next level has the condition Zone: My Computer, which is not matched, hence skipped. The next code group in this level is Zone:Intranet, which is matched, and it obtains the permission set Intranet and traverses down. In this level, the first code group has a matching condition of Site: w3.payroll.com which is not matched and it moves to the next code group in the same level that has the condition Site: w3.research.com which is matched. Security Manager obtains the permission set Research and traverses down to the next level. At this level, there are two code groups with matching conditions of different URLs. As the matching condition of the first code group is satisfied, security manager obtains Project1 permission set. As there are no subordinate code groups, the Security Manager moves to the second code group, which is not matched. Security Manager traverses up all the level up to the code group of matching condition Zone:Intranet and moves to the next code group in that level. This code group has a condition of Zone: Internet, which is not matched. That completes the security policy.

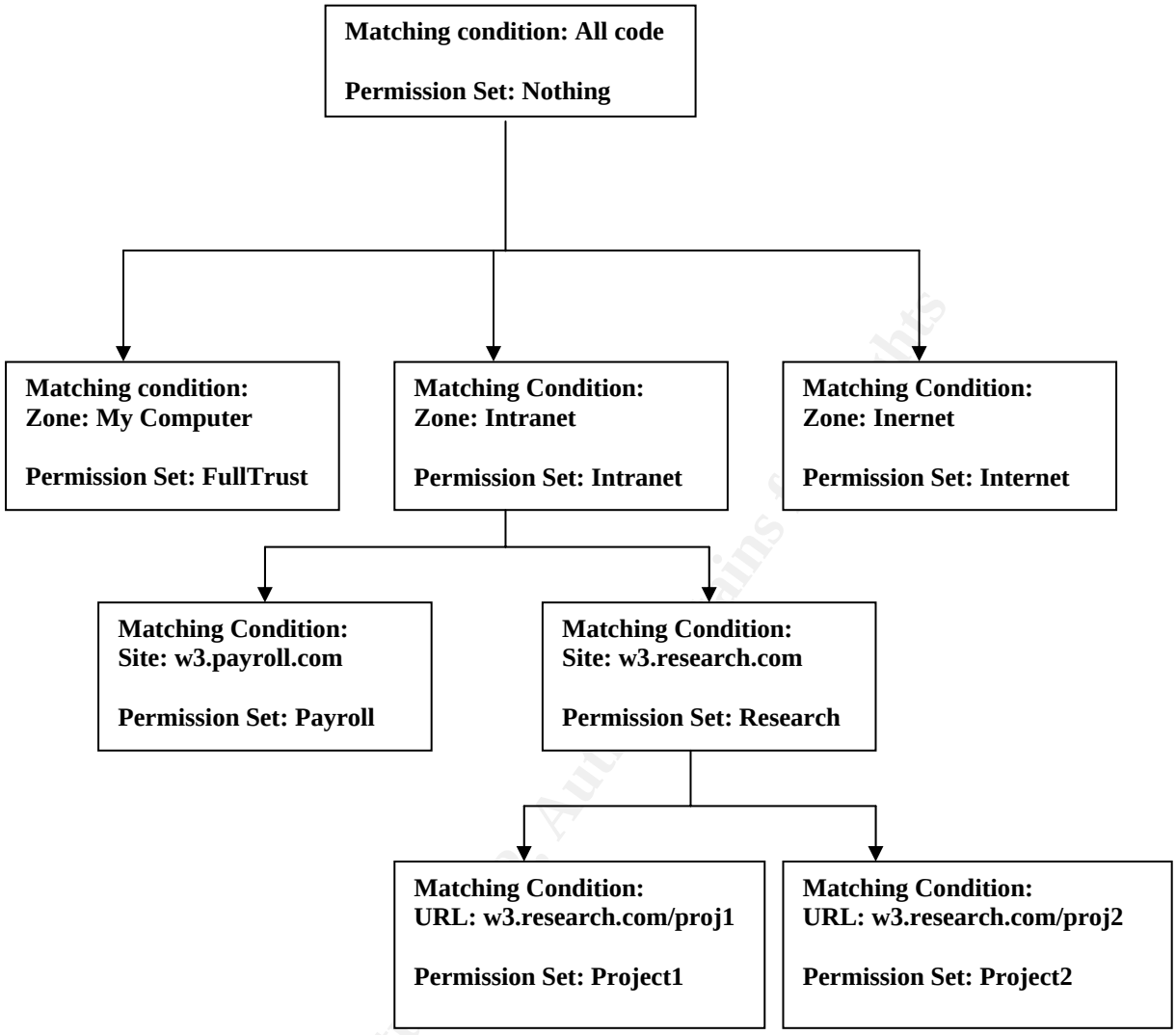

Figure 2: Code Access Policy

Finally, security manager makes the union of the following permission sets:
Nothing (from Zone: All Code)
Intranet(from Zone: Intranet)
Research (from Site: w3.research.com)
Project1 (from URL: w3.research.com/proj1)

Code groups can also control the traversal of the policy. This can be done by setting an *Exclusive* [15] attribute on a code group. If a code group condition is matched and it has *Exclusive* attribute set on it, then the Security Manager will grant only the permission set associated with that code group without traversing the entire policy tree.

2.3.2 Named Permission Sets

As the name implies, it is the set of permissions that are granted to an assembly. Code groups refer to different permission set by a name, hence the name Named Permission Sets [2]. In the above sample policy, Intranet is a named permission set that can have the following permissions:
SecurityPermission: Execute
FileIOPermission: Read, append, write files or directories
EnvironmentPermission: Read the environment variable PATH.

Following are some of the default named permission sets provided with CLR:

- Nothing: No permissions at all
- Execution: Execute only. No access to the resources.
- Internet: Recommended policy for code from internet
- LocalIntranet: Default permission set within an organization
- Everything: All permissions except skip verification
- FullTrust: Full access to all resources.

Permissions are objects that are part of the Class Assembly. There are various methods associated with these permissions like Demand, Assert, Deny, PermitOnly, Intersect and Union. Permissions [13] are divided into three types:

- Code access permissions
- Identity permissions
- Role based security permissions

Code access permissions protect broad range of resources like files, directories, environment variables, DNS access, user interface, registry, sockets, database access and services. Each resource type is associated with a set of code access permissions. Identity permissions are granted based on the evidence of the assembly. Role based security permissions can be used to determine whether a user has specified role to do a task. These roles can be completely specific to the application or Windows based.

2.3.3 Policy Assemblies

In addition to the permissions provided by the Class Library, custom permissions can be defined by the developers. While evaluating security policy, assemblies containing these custom permissions must be loaded. If any of these assemblies make use of the permissions defined in their own assembly, then that assembly can never be loaded as it goes into a loop of continuous check. In order to avoid these situations, these assemblies are defined as Policy Assemblies [2]. These assemblies will not be checked before loading.

2.4 Security Policy Levels

Security policy is organized into four levels [14]:

- Enterprise Policy
- Machine Policy
- User Policy
- Application domain Policy

Each of these policies have the same architecture as discussed above i.e. tree of code groups, permission sets and policy assemblies.

2.4.1 Enterprise Policy

This is the policy defined at enterprise level i.e. all the system in an organization or group of systems connected on a network. Changes made to the policy affect all the systems. This can be deployed using the tools provided by .NET SDK or SMS. Alternatively, caspol.exe command can be used to define the policy on each system.

2.4.2 Machine Policy

This is the policy defined on a specific system. Changes made to the policy affect only that system. This can be implemented using caspol.exe command, which is part SDK. By default, .NET is shipped with machine policy. Some of the highlights of this policy are:

- Unrestricted access to al resources if the code is originated from My Computer Zone or it has Microsoft strong name or ECMA strong name.
- No access for DNS, environment variables or event log and read access to files if the code is from internet.

2.4.3 User Policy

This is the policy defined for the user who is running the code. Changes made to the policy affect that particular user only. This can also be implemented using caspol.exe.

2.4.4 Application domain Policy

This is the policy defined at application domain level i.e. permissions to be granted to the assemblies loaded into the application domain. This can also be implemented using caspol.exe.

Microsoft Management Console snap-in [2] is available which provides a GUI to all the above policies. Each of the above levels can be configured, at any time. It is the responsibility of security manager in the CLR to enforce the security policy while executing any code. Security manager checks the policy in a hierarchical manner from top to bottom. Each policy level can only increase the security offered by the top level i.e. bottom level policies can not override the security policy set by the top level. At the end of the checking, security manager takes the union of all the permissions and makes the decision whether to grant access or not. End users can increase the security by reducing the permissions that are granted at the Enterprise Policy level. Similarly, machine administrators can make their system more secure by altering the Machine policy. This may make the application fail as it does not have enough permissions to run.
In order to avoid this, Enterprise Policy can be enforced to be the final policy by setting the *LevelFinal* attribute [15] in the desired code group in the Enterprise Policy. After setting the *LevelFinal* attribute, Machine Policy and User Policy will not be considered in the security permission evaluation.

2.5 Security Mechanisms in Microsoft.NET

Security in .NET is offered through various mechanisms. Some of the key mechanisms are,

- Verification

- Code Access Security
- Role based Security
- Stackwalk.

2.5.1 Verification

When an assembly is executed, CLR validates MSIL and verifies metadata that is part of the code. Verification [5] is the first security check performed by CLR. Before running the code, CLR checks that MSIL is type safe. A code is considered to be type safe only when it accesses its types in well defined and allowable ways according to MSIL grammar. In other words, interaction between types should always be through publicly exposed contracts (set of rules that define how to access a type). In addition to this, CLR also checks for correct usage of exception handling and stack overflow. If the code is found to be not type safe, then a security exception is raised and execution stops at once. CLR also verifies the metadata by examining the metadata tokens. All these tokens should index correctly into their type tables. If it is a string table, it checks that the tokens are not pointing at strings that are longer than their buffer size to avoid any possible buffer overflow. If the code passes verification, then MSIL is translated into native language of the system using a JIT compiler. If the assembly is completely loaded from local system, then verification happens as part of JIT compilation.

2.5.2 Code Access Security

In Code Access Security [2], permissions are granted based on the evidence. Evidence defines the identity of the code. Evidence is presented to the security manager whenever an assembly is loaded into CLR, by the hosting system. Evidence is typically composed of the following elements:

- Zone: Zones as defined in the Internet Explorer
- URL: URL name of the code
- Hash: Hash value of the assembly
- Strong name: Strong name signature of the assembly
- Site: The site name of the origin of the code
- Application Directory: Location of the assembly on the local system
- Publisher certificate: Digital signature of the assembly

.NET class library has a number of classes that can be used as standard forms of evidence. Using the code access security, even though the user in the Operating System is privileged, CLR restricts the access depending on the evidence of the assembly.

2.5.3 Role based Security

.NET allows developers to implement role based security in the application. These roles are not the same roles, which the operating system offers. These are very specific to the application and defined within the application only. Permissions are granted based on the Identity and Principal. Identity of a user can be established through authentication, which can be done using the native operating system or through an authentication service like Passport. Principal [16] contains the Identity and the authorization roles the user has. There are three types of Principals:

- Generic Principals: These are defined in the application only.

- Windows Principals: They represent the NT users and their roles.
- Custom Principals: These are special kind of principals that are custom made for the application.

Similar to code access security, role based security is also implemented using the objects and classes provided by the Class Library. It is also possible to make use of Windows Roles inside the application. Information obtained from code access security and role based security are used to grant permissions according to the security policy defined on the system.

2.5.4 Stack walk

An assembly may need to load a dependant assembly, during its runtime. Within the same assembly, a method may call another method. Every time a method is called, the current state of the execution along with the parameters passed to the method (if any) will be stored in the stack, as a new record. This ensures that the program returns to the correct point of execution. As the code is getting executed, the stack size changes depending on the methods called. Somewhere in this chain, if any method requires access to a protected resource, CLR implements a complete stack walk [2] to make sure that all the methods that are in the stack have access to that resource. A security exception will be thrown and execution fails, if any of the methods are not having the required permissions. Luring attacks can be avoided using stack walk. Luring attack [12] is initiated by remote code that tries to load a local assembly (located on the system). By default, the local security policy grants FullTrust to local assemblies. This will be exploited by the remote code in the luring attack. Because CLR performs a complete stack walk before granting access to the protected resource, remote code execution fails, as it does not have permissions to access the resource.

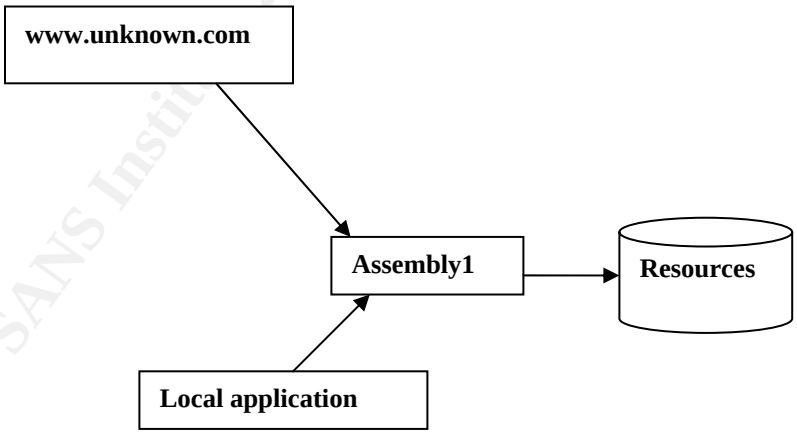

Figure 3: Luring Attack

In the above Figure 3, local application uses the Assembly1 located on the same system and can gain full access to all the resources. However, if someone is trying to execute a code from

www.unknown.com that also executes the Assembly1, then the code execution fails because CLR detects it in the stack walk.

3.0 CORBA

Common Object Request Broker Architecture is a very popular distributed object technology platform that is proposed and being controlled by Object Management Group (OMG). OMG is a non-profit organization supported by more than 800 hardware and software vendors. OMG specifies the standards for object oriented technologies. Once the standards are set by OMG, applications can be developed to conform to those standards.

3.1 Object Management Architecture (OMA)

OMA [21] is defined by OMG, which explains object oriented software environment. It consists of an Object Model (OM) and Reference Model (RM). Object model is an abstract model that defines basic requirements. RM is built using the OM.

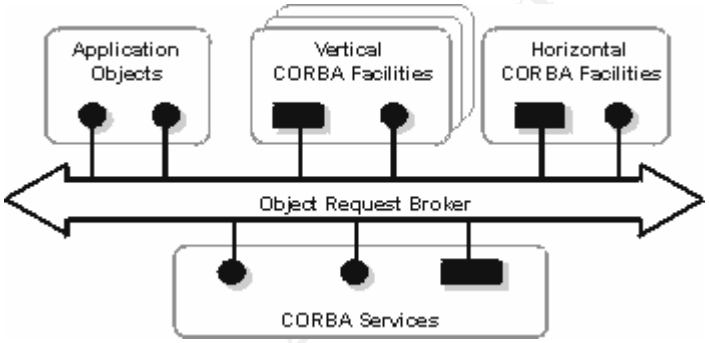

Figure 4: Object Management Architecture Reference Model [21]

As shown in Figure 4, OMA RM consists of five main elements

- **Object Request Broker (ORB)**
  ORB [20] is the heart of this architecture. It provides the communication channel between client objects and target objects in a transparent manner i.e. client objects need not be aware about the location of target objects.

- **CORBA services**
  These are the basic services that are required by most of the objects. OMG provides specifications for all these services. Two of these are services are naming service and security service. Naming service allows clients to find objects based on their names. Security services [17], which is the largest of all the CORBA services, provides security related services like:
  - Identification and authentication
  - Authorization and access control
  - Auditing
  - Confidentiality and Integrity
  - Non-repudiation

- Security Administration

Current version of the security services is 1.7. These services are also called CORBAsec.

- **Horizontal Facilities**
  These are services that are at a higher level than CORBA services. These services can also be used by most of the applications, like printing and email facilities.

- **Vertical Facilities**
  These services are useful for a specific business environment like health care, telecommunications or manufacturing.

- **Application Objects**
  These are the objects that are defined by the application developers. These are very specific to the application and are not defined by OMG.

3.2 Interface Definition Language

One of the main goals of the distributed object technology is to enable the use of different programming languages. CORBA offers this feature through Interface Definition Language (IDL) [17]. An interface should be written in IDL and compiled into the required programming language i.e. the language in which client program and server program is written. There are compilers available to map IDL into different languages like C++ or Java. Compiling an IDL interface generates stub and skeleton. Stub will be used in the client side program and skeleton will be used in the server side program.

3.3 Common ORB Architecture

CORBA defined by OMG is comprised of the following components:

- Object Request Broker
- **Client** This is the object that requires a particular operation or service.
- **Servant** This is the object that provides a predefined service.
- **IDL Stubs** This is the client side interface to the ORB.
- **IDL Skeletons** This is the servant side interface to the ORB.
- **Dynamic Invocation Interface** This allows invoking the target objects without the need of a stub.
- **Dynamic Skeleton Interface** This is the server sides equivalent of DII.
- **Object Adapter** OA is a server side component that manages the object references, method invocation, activation and deactivation of objects. More importantly, it is the responsibility of the OA to integrate with security services. A unique ID called ObjectID identifies each object within the scope of an OA.

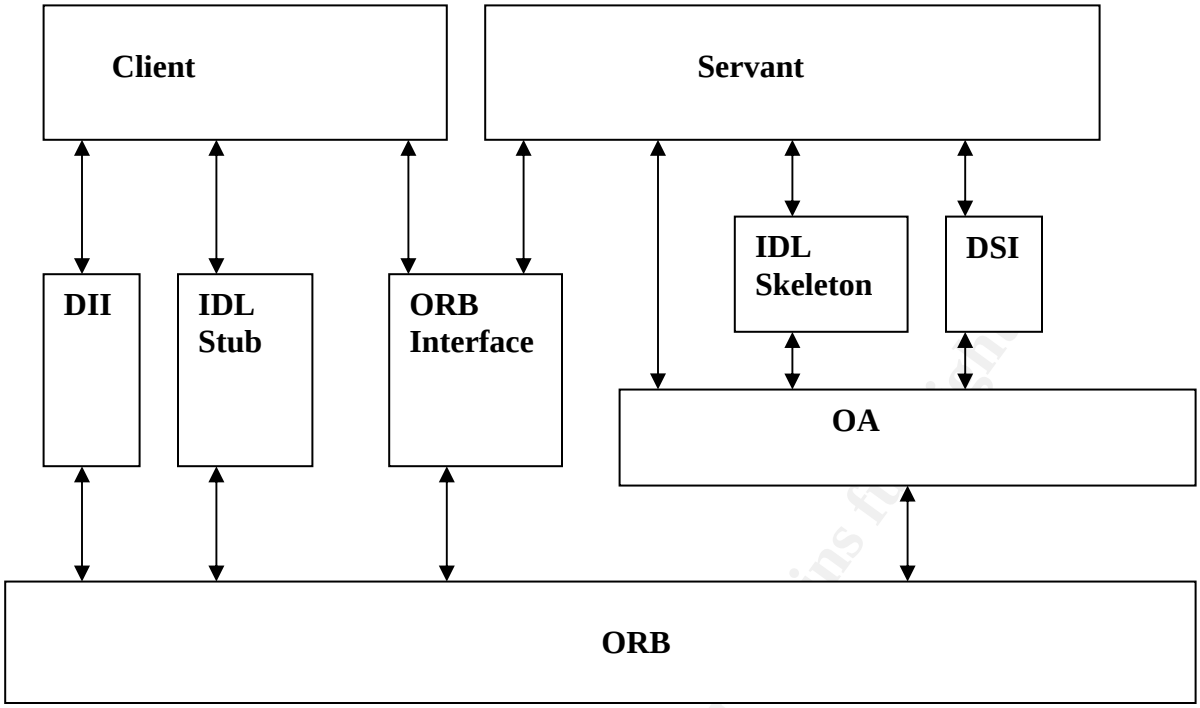

Figure 5: Common Object Request Broker Architecture [17]

Fundamentally, CORBA works on a client and servant concept. Client which itself is an object, requests for a service from another object called servant. Client and servant could be located on the same system or different systems connected across a network. An object can act as a client, servant or both. The originator of the object invocation is client and the recipient of the invocation is target. Sometimes, a target may need to invoke another object and hence it becomes a client also. In order to invoke an object, client needs the object reference of the target object. Client may already know this prior to the invocation or can obtain it from the naming services. Object reference consists of interface ID, implementation ID and optionally any data required for the target object. Once the object reference is obtained, client can communicate to the object reference through ORB. ORB receives requests from clients, marshals the parameters sent with the request, dispatches the request to the appropriate server, ensures the target object is activated and returns the output to the client. To make this entire transaction secure, it is required to ensure the following:

- Client is identifiable and authenticated
- Client is authorized to invoke the target object
- Target object is authenticated
- Confidentiality and integrity of the message transfer
- Protection against unauthorized access and modification

CORBA Security services offer the above functionality. In addition to meeting those basic requirements, following services are also offered:

- Auditing
- Security Management
- Non-repudiation

3.4 CORBA Security Services

OMG has defined the CORBA Security Services Specification that explains the ways to make a stand alone or distributed CORBA application secure. According to this specification, CORBA security is organized into several packages of different services.

3.4.1 Main Security Functionality Packages

It is divided into two levels, Level 1 and Level 2. In addition to those two levels, optional packages are also specified by OMG.

3.4.1.1 Security Functionality Packages, Level1

This level is targeted for applications, which are unaware of the security i.e. application objects are completely isolated from the security services. This level is suitable for applications, which do not have its' own security requirements. It offers the basic services:

- Authentication of users
- Establishing trusted relation between client and target
- Confidentiality and integrity of the messages
- Access control
- Delegation
- Auditing to some extent

3.4.1.2 Security Functionality Packages, Level2

This is targeted for the applications, which have their own security requirements. Applications can control the security provided at the object invocation. All the functionality offered at level 1 is also offered here along with some additional services. Some of the services are:

- Ability to authenticate inside or outside an object system
- Domain Access Policy
- More delegation options
- Security Policy Administration

A secure ORB implementation must provide at least one of the above levels to become a Secure ORB.

3.4.1.3 Optional Security Functionality Packages

As the name suggests, the services offered in this package are not mandatory. Non-repudiation is one such service that generates, stores and verifies the evidence of transactions in the application.

3.4.2 Security Replaceable Packages

This package specifies how replaceable the services offered in the ORB. ORB allows the use of interceptor interfaces to incorporate different services. Entire ORB services can be replaced or Security Services alone can be replaced as per the requirement.

3.4.3 Common Secure Interoperability (CSI) Packages

These packages offer the features required for inter-operating among different ORBs. Applications developed using distributed objects can make use of different hardware platforms and operating systems. Different ORBs are required to inter-operate without compromising the security. There are three levels of CSI.

- Identity based policies without delegation (CSI level 0): Only the principal identity is transmitted from client to target. Delegation (explained in section 3.5.3) is not supported.
- Identity based policies with unrestricted delegation (CSI level 1): Delegation is supported in this level i.e. intermediate client objects can impersonate the originator client.
- Identity and privilege based policies with controlled delegation (CSI level 2): Along with different kinds of identities like audit ID and access ID, privileges like groups and roles will also be transmitted from client to target. Initiating objects can control the delegation of credentials.

An ORB must offer one of the above packages to be a secure interoperable ORB.

3.4.4 Other interoperability packages

In addition to the above, following packages are also specified by OMG to provide secure interoperability:

- SECIOP Interoperability package
- Security Mechanism package
- SECIOP plus DCE-CIOP Interoperability

3.5 Concepts of CORBA security

3.5.1 Principal

Principal [23] is an identifiable active entity that is authenticated. For example a user logging into a system or application is a Principal. To access objects, every system entity must authenticate itself and establish its rights. Principal is comprised of two kinds of attributes, Identity attributes and Privilege attributes. Identity attributes serve the purpose of establishing identity to different security services. Several identities could be associated with the same principal, each serving its own purpose. For example, audit ID is used in the security auditing and access ID is used to determine the access rights. Privilege attributes are a list of groups and roles the entity is attached to and are used for access control. Identity and privilege attributes are obtained after successful authentication. In addition to that, Public attributes can be obtained without authentication. Authentication can be done by supplying a security name (user name), authentication data (password, challenge/response) and the required privilege attributes. After successful authentication, principal obtains the credentials, which is a list of security attributes. Once the principal obtained its credentials, it can invoke objects subjected to the security policy defined at the client and the target side.

3.5.2 Security Policy

Security policy is a set of rules that can be implemented at the client side or target side or both ends. This could be same or different at both ends. There are different types of security policies like audit policy, access policy or message protection, each focussing on a specific area. Broadly, security policy is used to meet the following requirements:

- Trusted relationship between client and target
- Access control
- Auditing
- Non-repudiation
- Confidentiality and Integration

3.5.2.1 Establishing a trusted relationship between client and target

Server may require the client to authenticate and client may require the server to authenticate. Objects can be accessed only after mutual authentication is successful. Alternatively, target may request the client to submit its' credentials.

3.5.2.2 Access control

This is implemented using an access policy. This consists of two layers, Object Invocation Policy and Application Access Policy. Object invocation policy decides whether the client has the rights to invoke the protected object. This is enforced by ORB and Security services regardless of the kind of application. Access decision is made based on the privilege attributes of the client, restrictions on these attributes like time of the day, type of operation requested and any control attributes on the target object. Application access policy enhances the security offered by ORB by implementing additional access control that is specific to the application like starting and stopping a service or access database. Access policy decides the access control for the protected objects.

Privilege attributes of the client and control attributes of the target are the key parameters based on which access decision is made. Privilege attributes grant special authority to do some operation. This can be done using roles or groups. Required permissions are granted to the role/group and individual users are attached to them. This makes the policy administration easy. Control attributes associated with the target object specify the permissions to different clients. These can be in the form of an Access Control List (ACL). ACLs consist of users, groups or roles and their permissions. Same ACL can be attached to multiple objects. It is possible to group a set of objects into a single entity called domain and apply an access policy to the domain. This kind of access policy is called domain access policy.

Domain is a logical grouping of objects based on some common characteristics. Security Policy Domain consists of one or more objects, which are controlled by a common security policy. Domain manager is the primary object to which the security policy is attached. All other objects in the domain are members of that domain. An object can be part of multiple domains, in which case the security policy is the combination of all those domain policies of that object. Security administrators set the domain policy. Permissions to do an operation on a secure object can be granted in the form of rights. Rights are grouped into Rights Family. “corba” is an example for rights family that contained most commonly used rights like s(set), g(get), m(manager) and

u(use). Additional rights can be defined as required. Rights Access policy can be defined for an object in terms of principals, rights to do an operation and right combinator. Right combinator defines the interpretation of multiple rights like any of the rights or all of the rights. Below example illustrates the domain access policy and required rights for a human resource department with two job functions Manager and Consultant who can execute different operations.

Example: The two job functions can be mapped into roles in the policy. Rights can be assigned to each of the roles and users can be given the appropriate role. There are two kinds of operations op1 and op2.

Domain access policy:

User name	Roles	Rights Family:Rights
paul	consultant	corba:gs
john	manager	corba:gsmu

Required Rights:

Operation	Rights Family: Rights	Rights combinator
op1	corba:gs	any
op2	corba:sm	all

User paul has consultant role and is allowed to do operation op1 but not allowed to do op2. User john is allowed to do both operations.

3.5.2.3 Audit

CORBA security services allow logging the access grants and revokes according to the audit policy. This is useful to analyse the security policy violations. Auditing is controlled by an audit policy. Audit policy can be defined based on the requirements like kind of operations or type of action i.e. success, failure or both. Audit policy can be set at the client or target or at both ends.

3.5.2.4 Non-repudiation

Creating, storing and verifying the evidence of transactions. Storing information like a client has invoked an object and a target has operated on the data sent by a particular client can be controlled by security policy. This evidence can be used to make the users accountable for their actions.

3.5.2.5 Confidentiality and Integrity

Messages transferred between client and target can be eavesdropped or even tampered. Security policy provides means of maintaining confidentiality and integrity of the messages in transit. Generally, establishing a trusted relationship between client and target ensures message protection. Symmetric and asymmetric key technologies are generally used to provide this.

3.5.3 Delegation

Client object can delegate some or all of its' attributes to another object as part of the invocation. When a client request a service from a target object, the target object may need to invoke another target object to complete a method. This chain could possibly grow bigger.

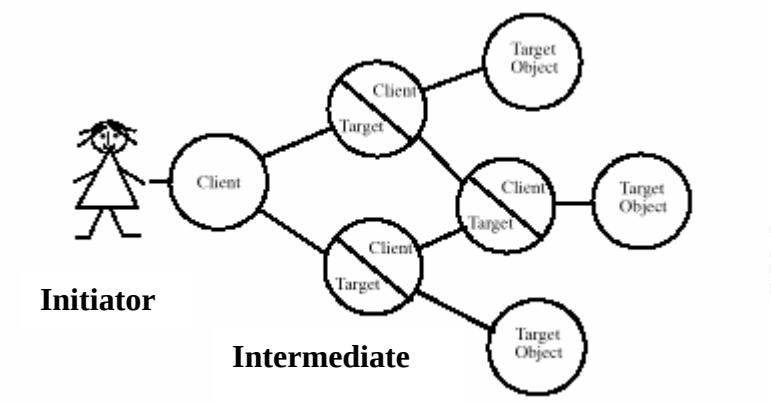

Figure 6: Delegation [23]

As shown in Figure 6, there will be an initiator, who initiates this chain and a final recipient who is the final target and a number of intermediate objects. Initiator can decide whether any of the attributes can be delegated to an intermediate object. Target object can restrict access based on the type of credentials i.e. initiator or delegated. There are different delegation schemes like

- **No delegation** Delegation is not enabled at all i.e. intermediate objects use their own credentials to access other objects.
- **Simple delegation** Client allows intermediate objects to use all of its attributes. It also enables the intermediates to delegate the attributes to other objects. Intermediate objects impersonates the client by presenting client credentials to the target.
- **Composite delegation** Credentials of client and intermediate objects are presented seperately. Targets can control the access using the two credentials.
- **Combined privileges delegation** Credentials of client and intermediate are presented to the final recipient as a single object.
- **Traced delegation** Credentials of each intermediate object is added and presented to the target.

4.0 Conclusion

.NET and CORBA offer various kinds of facilities to address the security concerns.

Identification and authentication

Distributed object technology allows clients to access objects or services available remotely. It is very important to establish the identity of clients through proper authentication mechanisms, before granting access.

.NET allows the usage of various kinds of user authentication mechanisms like native operating system or Passport service. Developers can implement their own authentication mechanisms within the application.
CORBA Security services provide interfaces which can be used to provide authentication. External security services can also be integrated with CORBA using interceptors.

Unauthorised Code
Application executing a malicious remote code may try to access the critical system or application resources.
Code access security policy in .NET restricts the access to the code based on the evidence, which may be composed of Zone, URL, Hash or Site.
In CORBA, security policy can be defined to establish a trusted relationship between client and target objects. Secure interoperability among different ORBs can be achieved using CSI packages.

Access Control
System resources and application resources are accessed differently by various objects and services. Control mechanisms that are used to protect critical resources should be scalable and flexible.
Four levels of security policy in .NET offers fine grained access control. A group of systems can be managed by defining enterprise policy. Access control can be further restricted on individual systems using Machine Policy or User Policy.
Domain access policy in CORBA can be used to define access control for similar type of objects. Any number of objects can be made domain members and an object can be made member of multiple domains.

Authorisation
Access to critical resources may be required for some legitimate users. Mechanisms to define authority and make access decisions based on the authority are required.
Role based security in .NET grants special permissions to access resources without allowing unauthorised access and disclosure. These roles can be completely specific to the application or operating system (Windows NT) roles.
Privilege attributes in CORBA in the form of roles or group membership can be used to define authority.

Confidentiality and Integrity
Messages transferred between the objects must be sufficiently protected as they are subjected to eavesdrop or alteration.
Cryptography namespace in .NET Class library can be used to ensure confidentiality and integrity of data transferred on the network.
Message protection in CORBA can be achieved as part of trusted relationship i.e. client and target can exchange a symmetric key encoded with their own private keys, before transferring the data. Symmetric key can be used in the subsequent communications between client and target objects.

Security Administration

Managing security in distributed object environment is very challenging. Security administration should be able to cater the growing needs of an organisation, allowing enough flexibility.
Security policy can be administered in .NET using the tools provided as part of the SDK. (caspol.exe) or MMC snap-ins.
CORBA security services offer numerous interfaces, which can be used to administer the security policy.

References

1. Damien Watkins, “Distributed Object Technologies”, Lecture Notes, June 2000
2. Damien Watkins, “An Overview of Security in the .NET Framework”, January 2002, http://msdn.microsoft.com/library/en-us/dnnetsec/html/netframesecover.asp
3. Microsoft Corporation, “ Overview of the .NET Framework”, .NET framework developer’s guide, http://msdn.microsoft.com/library/en-us/cpguide/html/cpovrintroductiontonetframeworksdk.asp
4. Jeffrey Richter, “Microsoft .NET Framework Delivers the Platform for an Integrated, Service-Oriented Web”, September 2000, http://msdn.microsoft.com/msdnmag/issues/0900/framework/framework.asp
5. Jeffrey Richter, “Part 2: Microsoft .NET Framework Delivers the Platform for an Integrated, Service-Oriented Web”, October 2000, http://msdn.microsoft.com/msdnmag/issues/1000/Framework2/Framework2.asp
6. Mary Kirtland, “The Programmable Web: Web Services Provides Building Blocks for the Microsoft .NET Framework”, September 2000, http://msdn.microsoft.com/msdnmag/issues/0900/webplatform/webplatform.asp
7. Microsoft Corporation, “Loading and Executing User Code”, .NET framework developer’s guide, http://msdn.microsoft.com/library/en-us/cpguide/html/cpconloadingexecutingusercode.asp
8. Keith Brown, “Managed Security Context in ASP.NET “, January 2002, http://msdn.microsoft.com/msdnmag/issues/02/01/Security/Security0201.asp
9. Microsoft Corporation, “Common Language Runtime Overview”, .NET framework developer’s guide, http://msdn.microsoft.com/library/en-us/cpguide/html/cpconcommonlanguageruntimeoverview.asp
10. Microsoft Corporation, “Code Access Security Basics”, .NET framework developer’s guide, http://msdn.microsoft.com/library/en-us/cpguide/html/cpconcodeaccesssecuritybasics.asp
11. Microsoft Corporation, “.NET Framework Enterprise Security Policy Administration and Deployment”, .NET framework developer’s guide, http://msdn.microsoft.com/library/en-us/dnnetsec/html/entsecpoladmin.asp
12. Keith Brown, “ Security in .NET: Enforce Code Access Rights with the Common Language Runtime”, February 2001, http://msdn.microsoft.com/msdnmag/issues/01/02/CAS/CAS.asp
13. Microsoft Corporation, "Permissions", .NET framework developer’s guide, http://www.msdn.microsoft.com/library/en-us/cpguide/html/cpconpermissions.asp
14. Microsoft Corporation, "Security Policy Levels", .NET framework developer’s guide, http://www.msdn.microsoft.com/library/en-us/cpguide/html/cpconsecuritypolicylevels.asp
15. Microsoft Corporation, "Administration with Code Group Attributes ", .NET framework developer’s guide, http://www.msdn.microsoft.com/library/en-us/cpguide/html/cpconadministrationwithcodegroupattributes.asp
16. Microsoft Corporation, "Principal", NET framework developer’s guide, http://www.msdn.microsoft.com/library/en-us/cpguide/html/cpconprincipal.asp

17. Damien Watkins, " The Common Object Request Broker Architecture : A Description", March 1999, Research Paper
18. Object Orientation Tips, " CORBA Basics", October 1997, http://ootips.org/corba-basics.html
19. OMG, "INTRODUCTION TO OMG'S SPECIFICATIONS", http://www.omg.org/gettingstarted/specintro.htm
20. Douglas C. Schmidt, "Overview of CORBA", November 2001, http://www.cs.wustl.edu/~schmidt/corba-overview.html
21. OMG, "Object Management Architecture", February 2002, http://www.omg.org/oma
22. Bob Blakley (IBM) and Belinda Fairthorne (ICL), "Introduction to CORBA Security", April 1997, http://www.infosys.tuwien.ac.at/Research/Corba/archive/special/secws/1-1.pdf
23. OMG, "Security Service, version 1.7", August 2001, http://www.omg.org/cgi-bin/doc?formal/2001-03-08