OMG White Paper on Security

OMG Security Working Group

editor: Belinda Fairthorne, ICL
email: s.b.fairthorne@bra0511.wins.icl.co.uk
Telephone: +44 (0)344 711082
Facsimile: +44 (0)344 711776

Summary

This White Paper describes the requirements for security in Object Systems conforming to the Object Management Architecture. It will be an appendix to Request for Product 3.

Issue: 1.0 April 1994

CONTENTS

Contents **2**

1 Introduction to the document **3**

2 Security in Distributed Object Systems **5**
2.1 Introduction to Security 5
2.2 Security Issues for Distributed Object Systems 5
2.2.1 What makes distributed object systems different? 6
2.2.2 What does CORBA say about security? 7

3 Requirements **8**
3.1 General Requirements 8
3.1.1 Consistency 8
3.1.2 Scalability 8
3.1.3 Availability 9
3.1.4 Business Requirements for Security 9
3.1.5 Interoperability 11
3.2 Functional Security Requirements 11
3.2.1 Identification and Authentication 12
3.2.2 Authorisation and Access Control 12
3.2.3 Security Auditing 13
3.2.4 Propagation of Attributes about Principals between Objects 14
3.2.5 Secure Communications 14
3.2.6 Cryptography 15
3.2.7 Administration 15

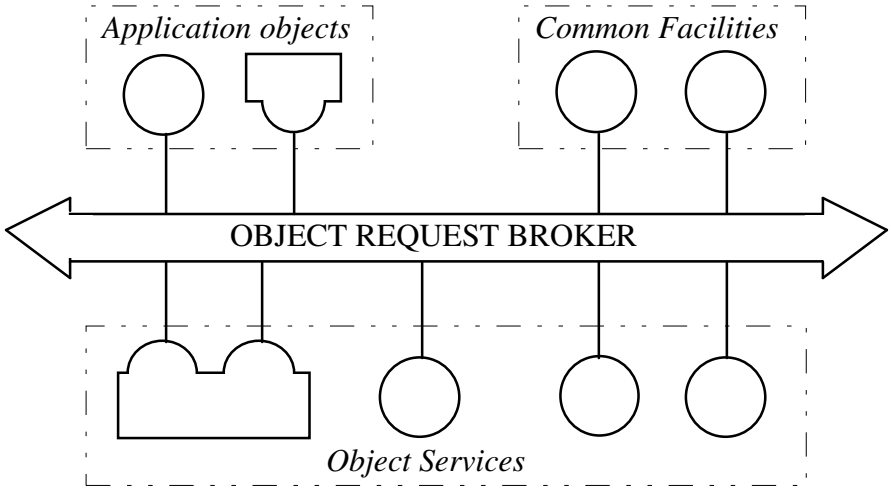
4 Architectural Implications **16**
4.1 Introduction 16
4.2 Architectural Implications of Normal Use 17
4.3 Implications on the Architecture for Interoperability 19
4.4 Trusted Computing Base 20
4.5 Interfaces 21

5 Additional Issues **22**
5.1 Security model 22
5.2 Requirements 23
5.3 Implications on ORBS and CORBA Architecture 24
5.4 Implications on Other Object Services 24

6 References **25**

1 INTRODUCTION TO THE DOCUMENT

This white paper describes requirements for security in distributed Object Systems conforming to the Object Management Architecture (OMA). The major parts of the OMA Reference Model are shown in the following diagram.



- The Object Request Broker (ORB) enables objects to make and receive requests and submissions transparently in a distributed environment. The Common Object Request Broker Architecture (CORBA) defines the programming interfaces to the ORB.

- Object Services is a collection of services (interfaces and objects) that support basic functions for using and implementing objects.

- Common Facilities is a collection of services that provide general purpose capabilities useful in many applications.

- Application Objects are objects specific to particular end-user applications.

Security affects the whole distributed system, so introducing security has implications on the Object Request Broker, most Object Services, Common Facilities and object implementations, as well as requiring provision of security services.

The goals of this paper were:

- to provide significant input to the Request for Product (RFP) on security in OMA compliant object systems

- to educate OMG members on how security is expected to affect such object systems

- as the result of the above, to involve appropriate people in completing these security requirements so they do not have unacceptable affects on other OMG requirements such as usability and performance

Section 2 of the white paper provides an introduction to security and the particular issues of providing it in a distributed object system. Note that current security standards do not fully address all the requirements for distributed object security as described in this paper.

Section 3 gives the requirements for security. It includes general requirements such as consistency and portability and also the requirements for specific security functionality.

Section 4 gives architectural implications of putting the system together in a secure way so that the required security is enforced. It also outlines the sort of interfaces that will be needed.

Section 5 looks at what responding to the requirements will mean, and some of the questions responses will need to answer.

Security Terminology

In achieving the above goals, this white paper will be read by a broad technical and business audience. Certain common terms such as function, policy, service and mechanism used in this document have specific meanings in security terminology. Definitions of security terms are available in the reference documents listed in section 6. In this paper, most security terms are defined when first used in the text; however, a few key terms and the relationship of these to each other are provided here as background for the reader.

Security function - a specific security-relevant function utilised in providing one or more security services

Security mechanism - a specific method which implements one or more security functions

Security policy - a set of rules and criteria that constrain one or more sets of security-relevant activities for the provision of security service

Security service - a combination of functional and data elements that are exercised through a well defined interface, provided by a security infrastructure, which ensures adequate security of computing information or resources

Threat - potential violation of security

Relationship between Terms

In general, organisations concerned with security *threats* specify a security *policy* to address these risks. The security policy is specified in terms of rules and criteria. These policies are invoked through a set of security *services* which provide the necessary security *functions* for the necessary protection to counter the threats. A variety of *mechanisms* may be used to achieve the necessary security functions. In some cases, there may be a choice of security mechanisms to implement security functions. Multiple security mechanisms may be utilised to achieve an organisation's security policy.

2 SECURITY IN DISTRIBUTED OBJECT SYSTEMS

2.1 Introduction to Security

Security protects an information system from unauthorised attempts to access information or interfere with its operation. It is concerned with:

- **Confidentiality**: Information is disclosed only to users authorised to access it.
- **Integrity**: Information is modified only by users who have the right to do so, and only in authorised ways. It is transferred only between intended users and in intended ways.
- **Accountability**: Users are accountable for their security relevant actions.
- **Availability**: Use of the system cannot be maliciously denied to authorised users.

  [Availability can only be discussed relative to what is required to be available (e.g. communications or the service provided by an object). The way of fulfilling this requirement depends on the particular functionality/service. This diversity makes general consideration inappropriate so is not considered further in this document.]

Security is enforced using security functionality such as authentication, access control, auditing, encryption and associated administration. In addition, there are constraints on how the system is constructed, for example, to ensure adequate separation of data and functions so objects don't interfere with each other and separation of user's duties so the damage an individual user can do is limited.

Security is pervasive, affecting many components of a system, including some that are not directly security related. Additional components - an authentication service, for instance - provide services that are specific to security.

The assets of an enterprise need to be protected against perceived threats. The amount of protection the enterprise is prepared to pay for depends on the value of the assets and the types and strengths of threats which have to be countered. The security policy needed to protect against these threats may also depend on the environment and the how vulnerable the assets are in this environment. The OMA security architecture must support a variety of such security policies to meet different needs.

2.2 Security Issues for Distributed Object Systems

Security is an important issue in any distributed system, as information in transit is more vulnerable, and use of several machines introduces issues of trust and consistency between them. Distributed object systems are generally more complex than some traditional client-server systems, and the security issues can be more subtle. In this section we discuss some of the special properties of distributed object systems, and their implications for security.

2.2.1 What makes distributed object systems different?

In general, distributed objects systems can be unlimited in scale and may be continually evolving in a distributed and often heterogeneous environment. This means systems which:

- have many components,
- have rich interactions between components, and
- can introduce intricate boundaries of trust.

Distributed object systems have many components

One of the key goals of OMG is to make it possible (and easy) to build complex software systems from a set of components (i.e. CORBA objects). The components that a user (and/or application programmer) uses directly when interacting with a software system may merely be the tip of the iceberg. The user may be aware of only a small percentage of the underlying objects in the distributed system. Also, because of subclassing, inheritance or customisation, the implementations of objects may change over time, often without the user/programmer knowing or caring.

There are rich interactions between components

Simple client-server architecture have straight forward interactions between components which are usually well-understood by users and programmers. In distributed object systems, the distinction between clients and servers is no longer a clear one - a server that implements a CORBA object might also act as a client of another object. Some objects may be implemented by delegating part of their implementation to another object, or (more generally) by using a set of objects connected in some arbitrary graph.

In general, users and programmers do not understand all of the interactions that take place between objects that they invoke. Encapsulation hides the details of an object's implementation from programmers, including which method implementation is selected in response to a particular invocation. So the proper use and understanding of a security architecture should not have to depend upon a full understanding of how objects interact "behind the scenes". A security architecture is fundamentally user-oriented. Its role is not only to provide security mechanisms, but also to help users gain confidence in the integrity of the system, and clearly understand what is, and is not, being protected against. Users should not be excessively "surprised" by the introduction of security policies (e.g. getting mysterious "no permission" errors with no additional information), nor should they be lulled into over-confidence.

Boundaries of trust are more intricate

In a simple client-server system with relatively few components connected in straightforward ways, it is usually clear which of these components are trusted and by

whom. For example, clients of a network file system will typically trust the server, but not vice-versa. In particular, a user will typically trust the server not to disclose the contents of his personal files, and will trust the client file I/O code to execute with his full privileges (because it is trusted systems software running on his own machine). Simple models of trust like this are usually well-understood by users and programmers.

With distributed object systems, however, things can be more complicated. The "natural" boundaries between objects provided by encapsulation may not always be the right ones to respect, as object boundaries are not always supported by underlying security protection mechanisms. So a user may trust an object with confidential data, for example, when its implementation runs on his local machine, but not if it is running on some unknown machine. Even if the implementation is running on a trusted machine, it may uses other objects which run elsewhere or are not known to keep data confidential. CORBA-compliant systems have the flexibility to introduce and/or replace large numbers of components, and this very property risks leaving them vulnerable to "trojan horses".

In an environment where users and programmers may not be aware of all the objects with which they are interacting, what objects (or collections of objects) can they trust? Similarly, where object implementations (or the CORBA implementation supporting them) make decisions about security, what other objects should they trust? An OMA security architecture should allow for environments where mistrust between objects is ubiquitous.

The trust which users/programmers have in objects, and object implementations have in each other also depends on the trust in the CORBA implementation and the boundary of trust between object implementations and CORBA implementation. Moving some security into the CORBA implementation could mean little trust is needed in most object implementations. Also, use of trusted tools to generate objects and/or moving some security into run-time libraries can reduce the trust needed in the part of the object implementation written by the application developer.

2.2.2 What does CORBA say about security?

Finally, let us review what the CORBA specification already has to say about security. The CORBA specification discusses security briefly, in the context of the Basic Object Adapter (BOA). Section 9.2.4 identifies an interface "Principal", and states that the BOA "guarantees that for every method invocation (or object activation), it will identify the principal on whose behalf the request is performed". An object implementation can use the "get_principal" operation to obtain this principal. This section also notes that the meaning of this principal "depends on the security environment", and that it is left to the object implementation to decide whether or not to permit the operation.

This gives little help in identifying what security is needed in the system, and where and how to provide it. Also, in some security models, object implementations are not generally trusted to enforce (some of) the security policy, so should not be left to decide whether to permit the operation. Other security models may trust object implementations to do this.

3 REQUIREMENTS

The sensitivity and value of the information being secured, and the threats to which it is exposed may vary from system to system, but there are common security requirements which apply to many commercial and government organisations, though some government systems will have more stringent needs. These requirements can be grouped into General Security Requirements and Functional Security Requirements.

3.1 General Requirements

The following define this set of general requirements for managing a secure, distributed and heterogeneous environment in an organisation and are common to every security system. Security services, as an important element of a trusted distributed system, must adhere to certain principles which underly distributed systems.

3.1.1 Consistency

The security model must offer a consistent and standard approach that facilitates portability across all platforms and encompasses both legacy and new system applications. This includes:

- support the use of common policies for who should be able to access what sort of information within a domain that includes heterogeneous machines
- compatibility with existing permission mechanisms. The model should function as a means of enhancing these mechanisms.
- fit with existing environments, for example, the ability to provide end to end security even when using communication services which are inherently insecure
- fit with existing logons (so extra logons are not needed) and with existing user databases (so extra unnecessary user administration is not needed).

3.1.2 Scalability

It must be possible to provide security for a range of systems from small, local systems to large intra- and inter-enterprise ones. For larger systems, it should be possible to:

- base access controls on the privilege attributes of users such as roles or groups (rather than individual identities) to reduce administrative costs
- have a number of security domains which enforce different security policy details, but support interworking between them consistent with their mutual trust
- manage the distribution of cryptographic keys across large networks securely and without undue administrative overheads, for example, by using public key technology

3.1.3 Availability

Security should be available as an end-to-end facility as transparently as possible to users and applications, based on sensible, configurable defaults.

3.1.4 Business Requirements for Security

Security is often seen as commercially prohibitive by customers. There is a limit to the amount which customers are willing to invest in security services and mechanisms or willing to allow for additional computing resources dedicated to security. Therefore the business requirements of security are extremely important to ensure commercial viability.

Regulatory Requirements

The security model must conform to government regulations on the use of security mechanisms (cryptography, for example). There are several types of controls:

- controls on what can be exported
- deployment and use controls such as limitations on confidentiality encryption in France
- privacy controls.

Details vary between countries; examples of requirements to satisfy a number of these are:

- allow use of different cryptographic algorithms
- keep the amount of information encrypted for confidentiality to a minimum
- use identities for auditing which are anonymous, except to the auditor (for Germany)

Enforceability

The OMA security architecture must allow security policies to be enforceable. It must not be possible to bypass the security controls.

Portability

Where an object is not itself responsible for enforcing part of the security policy, the object should be portable to environments that support different security policies. (Objects should not be required by the architecture to enforce their own security - either authentication or authorisation.) When an object enforces security itself (for example, uses particular privilege attributes for access control), portability depends on the consistency of the security policies. Interfaces to security services (such as authentication) should be independent of particular security services and protocols, so the same objects can be used with different security services and mechanisms.

Usability

If security is difficult to administer and use, it will not be set up correctly and users will try to bypass it, so the resultant system will not be secure. Requirements here include:

- usability for end users. For example, users should need to logon to the distributed system only once to access object systems and other IT services.

- usability for administrators. The administrative model should be based on a simple and unambiguous model which is represented intuitively at the user interface. Also, it should not be necessary for an administrator to specify controls for individual objects or individual users of an object (expect where security policy demands this).

- usability for programmers. Object developers should not need to understand security to fit with the security policy (unless the object needs particular protection of its own functions and data).

Evaluation Criteria

A secure product must also meet appropriate assurance criteria, to give the required level of confidence in the correctness and effectiveness of the security functionality. There are international criteria for assurance (as well as security functionality) such as the US TCSEC and the European ITSEC criteria, though Common Criteria are now being produced which should replace these (see section 6). Formal evaluation may not be needed for many commercial systems (if current initiatives happen), though an object system vendor may have to show sound development and quality assurance processes.

Performance

Security should not impose an unacceptable performance overhead, particularly for normal commercial levels of security, although a greater performance overhead may occur as higher levels of security are implemented.

Flexibility

As the security policy required varies from enterprise to enterprise, choices of security functionality should be allowed. This should allow an enterprise to pay for and administer only the level of protection needed, reducing the level (and therefore costs) for less sensitive information or when the system is less vulnerable to threats. The costs of providing security, include the resources required to implement, administer and run the system, should not exceed the perceived cost of losses from security breaches.

Standards

The OMA security architecture should conform to the appropriate distributed security framework and interfaces, for example, the emerging POSIX and X/Open standards.

Security Mechanism Independence

The OMA security model should be independent of the particular security mechanisms used, so the most appropriate ones can be chosen to meet other requirements. Therefore OMA security interfaces should hide details of these from object implementations, ORBs etc. Emerging industry standard interfaces, such as GSS-API should be used. It should allow, for example, use of both existing security mechanisms such as Kerberos and also mechanisms using public key technology for scalability such as Sesame, Netware 4 and NIS+. Similarly, the model should allow use of different cryptographic algorithms.

3.1.5 Interoperability

The security model should support interoperability between objects. This may include:

- across a heterogeneous system where different vendor's may supply different ORBs, and these may use different communications protocols and different security services
- between domains of the same distributed system where different domains may support different security policies, for example, different access control attributes
- with non secure systems

The architectural implications of this are considered in section 4.3.

3.2 Functional Security Requirements

The security requirements described above require implementation of security functionality such as authentication, access control and audit. This section describes the types of functionality expected to be required by OMA compliant secure systems. Note that any implementation is likely support only a profile of this functionality. This may conform to one of the TCSEC, ITSEC etc criteria for security functionality classes or profiles.

Security functionality includes:

- **identification** and **authentication** of principals (human users and objects which need to operate under their own rights) to verify they are who they claim to be.
- **authorisation** and **access control -** deciding whether a principal can access an object, normally using the identity and/or other privilege attributes of the principal (such as role, groups, security clearance) and the control attributes of the target object (saying which principals, or principals with which attributes) can access it
- **security auditing** to make users accountable for their security related actions. Note: it is normally the human user who should be made accountable. Auditing mechanisms should be able to identify him correctly, even after a chain of calls between objects.

- **secure communication** between objects to protect both systems data (e.g. security attributes of principals) and business data in transit between objects, which is often over insecure communications. This needs security mechanisms which utilise security functionality such as authentication and cryptography

- **cryptography** is used in support of many of the other functions and is rarely visible directly outside the security services, except to provide encryption of data under direct object control

- **administration** of security information about principals, server objects and security policy configuration options is also needed

The need to segregate objects, prevent object re-use and execute objects securely is described in section 4.

3.2.1 Identification and Authentication

Each user of the system must be properly authenticated. This may be done using a security service, but it must also be possible to acquire credentials of users who have been authenticated outside the object system so the number of user logons is not increased. User authentication should be possible using different methods such as passwords, smart cards or thumb print readers. Authentication of other objects may also be needed when these act as principals with their own identities and rights.

Authentication should result in a unique, certified identity which cannot be changed by the object itself, only by the authentication service. More than one identity will sometimes be needed as the identities for access control, audit and charging may not be the same.

3.2.2 Authorisation and Access Control

The authority to carry out functions may be based on the certified identity of the user (established on authentication). However, many security policies also require use of other privilege attributes of the principal such as role, group or security clearance. Finer granularity of privileges can reduce the amount of damage any one principal can do. It should be possible to assign such access privileges to any user or object on the system or network. As for identity, it should not be possible to use or change such attributes without calling a security service to check that this principal is allowed these attributes in this environment. The location of the user and the time can also affect what the user is allowed to access. It must be possible to support different levels of authorisation for application objects. That is, it must be possible to manage authorisation for each application object separately and at varying levels.

Access control governs operations on objects. The OMA security model should allow different types of access control schemes to be used to support different security policies (though no one implementation must necessarily support all of these).

For example, it should be possible to use:

1) **Access Control Lists (ACLs)** which list who can do what operations (or groups of operations) on what objects. The security model should allow:

   - access control to be specified for individual objects or some collection of objects
   - ACLs to specify the identities and/or privileges (e.g. role, group) of users

2) **Label schemes** where principals and the objects they access have labels, and there are rules specifying which principals can perform which operations on an object.

   These are most commonly used with mandatory access control schemes in higher security systems conforming to the TCSEC B level. In these, the principal label contains information such as the principal's security clearance and the object label the classification of the object. The rules may specify, for example, that a user cleared to CONFIDENTIAL cannot read SECRET information.

   A key feature of mandatory access control is that the controls are enforced by the system - the owner does not have the discretion to say who can access objects. Information of different classifications may need to be separated.

3) **Capability schemes** where object information (name and operations allowed) are used as principal privileges.

3.2.3 Security Auditing

Auditing is necessary to ensure that users are accountable for their security related actions. As for authentication and access control, the audit functionality required may be dictated by criteria such as the TCSEC and ITSEC ones.

User must have unique identities if they are to be accountable for their security related actions throughout the distributed system. It should be possible for the identity used for auditing to be different from that used for access, as the access identity is sometimes shared or loaned (for example, to allow a deputy to use facilities) and the audit identity may need to be anonymous (except to the audit administrator).

The object system (including security services) should be able to audit all significant security related events. These events may include authentication, activating objects and security administration. Object implementations should also be able to audit their own events.

Audit trails should be protected so that unauthorised users cannot modify or delete audit information, for example, remove records of their actions from the audit trail. In a distributed object system, audit information from a number of machines is likely to need collecting more centrally. Any such collection software must keep the audit trail protected.

For ease of administration, it may be desirable to classify events to be audited into types, the recording of which can be selectively switched on and off and routed, for example, as an alarm to a security administrator as well as an audit record to an audit trail. Tools are needed to analyse the audit trails, but these are not expected to be subject to standardisation by OMG.

3.2.4 Propagation of Attributes about Principals between Objects

The user (or other principal)'s credentials identify the user and his security attributes as required for access control and also accountability. In an object system, a call on an object to perform an operation is likely to require calls on many other objects. So the principal's security attributes often need to be delegated to these objects so they can use the privileges by proxy for access control or audit the security related actions, identifying on whose behalf they were done.

Some policies will also require controls of when and where these attributes can be used. On delegation, the user, or other principal may want to downgrade the privileges passed to the delegate for use in access control decisions on his behalf. (He may not allow certain privileges to be delegated at all.). He may also want to reduce the time for which they are valid and constrain which (collections of) objects can use them. The object may want to know which objects have handled them.

Large systems will often require several security domains and the principal's security attributes will need to be passed between them. In some cases, these security domains may have different security policies, where not only does trust have to be established between domains, but the attributes from one domain may need to be mapped to authorised ones in the other domain to provide credentials valid for access control and audit there.

3.2.5 Secure Communications

Distributed systems are vulnerable to attack when information is in transit. Information must be sent to the correct place and protected against corruption and replay; it may also need to be confidential. Systems data as well as application data needs protecting. Security of communications should be end to end, not relying on (though able to exploit) security in the communications services, and able to work between domains in large systems.

The actual security mechanism and services used to protect the communications between objects in a distributed system should normally be hidden from the object and generally also from the ORB, Object Adapter and other object services. However, these should be able to specify the quality of protection needed for a dialogue and/or particular messages, where this is consistent with the overall security policy.

The communication facility utilised by the security model must be capable of supporting a variety of authentication protocols or algorithms and be capable of extending the set of protocols supported as new protocols are developed.

Whether security services are provided at the network or higher layer, they should protect the data on every distributed computing channel and be provided transparently to the application users (unless these specify the quality of protection) and with minimum impact on network performance and interoperability.

3.2.6 Cryptography

Cryptographic algorithms are used in the protection of information. In general, the particular algorithm(s) used and even their types (symmetric or asymmetric) should be hidden from most users of the system.

The cryptographic keys used by these algorithms need managing securely - creating, distributing, archiving and destroying. The way this is done should be hidden, where possible, from the user and application developer, as it depends on the type of algorithms and security mechanisms used.

3.2.7 Administration

Where principals (users and some other active objects) are to be known within the object system, operations are needed to register them with the security services and maintain their identities and (particularly for users) security privileges such as permitted roles, groups, security clearances. Operations are also needed to maintain security properties about objects (and/or collections of objects) for access control e.g. ACLs, labels. It should be possible to restrict some of these operations so they can be used only by appropriate security administrators, though it should also be possible for a parent object to register a child object and set those security properties allowed by the security policy.

Security administration also includes management of encryption keys and public key certificates, where used, and may include revocation of a user's current credentials.

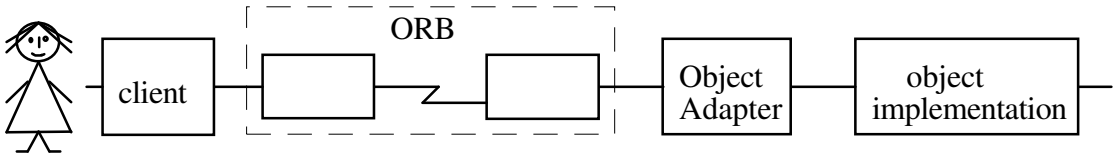
Facilities should also be provided to establish and maintain security policy options such as:

- the event or event classes which should be audited
- default life time for credentials
- the standard level of integrity protection when communicating between objects
- access control details such as privilege types supported, classes supported for labelled based access control schemes

It should be possible to split administration so different administrators can have separate duties and each has the least privilege needed to do his job. For example, the audit administrator is separate from the security administrator who is different from the systems administrator and their may be separate administrators for different domains.

4 ARCHITECTURAL IMPLICATIONS

4.1 Introduction



In a OMA system, the user calls a client which uses a (distributed) ORB to locate an object and this calls an object adapter to activate the associated object implementation. This object may then use the ORB to call another object.

To make such a OMA compliant system secure:

- The user must first be authenticated, resulting in an associated security context which includes his certified identity and security attributes (credentials).
- A (security relevant) action should be performed only after checking the security context in which the action is initiated to determine whether the action is authorised. The (attempt to) perform the action may also need to be audited.
- When a client invokes an object, the user/client's rights to use this method on this object may need to be checked and the invocation may need to be audited. Also,
  - the request and the principal's security attributes must be transmitted securely to the right object. Subsequent data may need to be protected in transit between them
  - the object should be activated in an environment with the required security context
- If the object now calls on other objects, this must be done securely as before. Also, the principal's security attributes may be delegated so the new object can act on its behalf.

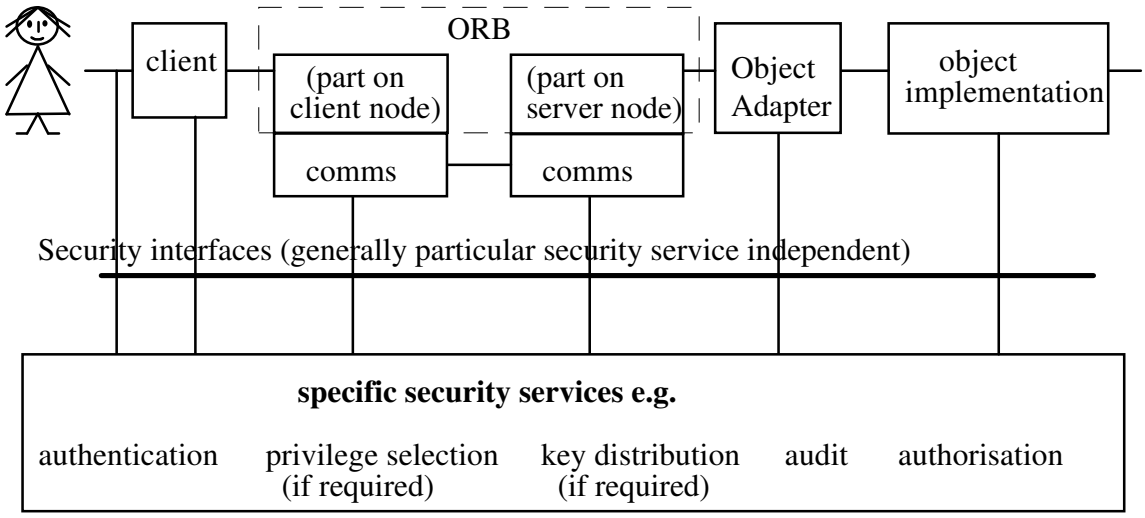
The difficult part of making the system secure is guaranteeing that:

- users are **always** authenticated and get the proper security context
- security context information is **always** tamperproof and is **always** properly propagated when other objects are called or new objects are created
- security relevant actions are **always** checked against security policy data and the appropriate security context to determine if the action is authorised
- security policy data is **always** tamperproof
- auditing is **always** done when called for by the auditing policy

4.2 Architectural Implications of Normal Use

The following is a walk through of a user logging on and then invoking objects to show how object systems components use the security facilities to meet the requirements in section 3 above. At relevant points, it identifies possible options and highlights some of the implications of requirements such as enforceability, portability and security mechanism independence on the architecture. All systems are not expected to support all options.

The requirements make it very desirable for most clients and object implementations to know little if anything about security, relying on the underlying object system to enforce the security policy. However, some objects will want to enforce security themselves - that is, control access to their own functions and data, control propagation of privileges, and audit security relevant actions. This walk-through covers both types of objects.



- A user logging on is authenticated via security services which may also select rights such as role and security clearance. An object system may need to support logon both:
    - outside the object system (for example, by the operating system) to give a single sign-on to both the object systems and applications accessed independently of it. The user's credentials from log-on must be made available to the object system.
    - by security services available within the object system.
- The user starts a client. The user's credentials are made available to it.
- This client calls another object. At this stage, a security aware client may specify:
    - the quality of protection of information wanted between it and the target object, possibly for the whole dialogue, sometimes for a particular message. This may specify the strength of integrity and/or confidentiality required. (There should be a

per-system or per-user default quality of protection for clients which are **not** security aware.)

- whether the user's or the client's security attributes (or a combination) should be transmitted to the object for access control and auditing there and specify any constraints on the attributes to be passed. For example, should they be available only for a limited time and can they be delegated when the object calls others?

- The client and server establish a security context between themselves. They (or software acting on their behalf) may need to establish mutual trust and the user and/or client's security attributes will be passed securely to the server (for example, in the form of a Privilege Attribute Certificate - PAC). They should utilise security interfaces such as the Generic Security Services API (GSS-API) which hide the particular security services used. Note this also means handling the security information passed (attributes and associated protection information) as an opaque "token" as details will depend on the security mechanism.

  The calls on the security interface may be done in the ORB or the supporting communications software (if, for example, a secure RPC is used). The caller should not know if satisfying the request needs calls on one or more security services.

  The OMA security model should allow a security context to be set up between a client and an individual object or a collection of objects. For example, it should be possible to set up a context between a user process (representing a client) and an object adapter (representing a collection of objects). However, finer granularity may also be needed. Some security policies may require all objects be protected from each other, while in other cases, particularly where light-weight objects are used, the cost of individual protection may be too high.

- When the request reaches the server system, an access control check may be needed to see if this user can use the requested method on this object. This check uses the principal's privileges and access control information about the object (or collection of objects). The OMA security model should not dictate a particular access control policy. For example, it should allow access control to be based on the principal's identity or other privilege attributes using ACLs but also allow label based access control. A general authorisation service may be used, if available.

  This access control decision could be made by the ORB, Object Adapter (OA) or the object implementation itself. If this access control is part of the security policy which must be enforced even when security unaware, untrusted objects are used, the check will need to be in the CORBA implementation, and in this case, the Object Adapter which dispatches the method call to the object is the most likely place. However, where the security model trusts object implementations for this, the access check can be left to the implementation. Also, in either case, some objects will want to control who can call them and therefore perform their own access checks.

- The invocation may also need auditing.

  Like the access control check, auditing the invocation could be done in the CORBA code or the object implementation, depending on the security model.

- The object adapter activates the object in a suitably secure environment. For some security policies, objects (or collections of objects) will need to be kept separate to prevent interference between them. The way the environment is set up is system dependent; however, the Object Adapter may need to obtain the principal's privileges when deciding what the security attributes of the environment need to be.

  Security policies conforming to TCSEC or ITSEC will require any remains of a previous object to be cleared out, to prevent object re-use i.e. so the new object cannot access sensitive data belonging to a previous object which used the store.

- Once the security context has been established between client and object, they will often transmit data between them. This data may be integrity and/or confidentiality protected using dialogue keys set up during the establishment of the security context.

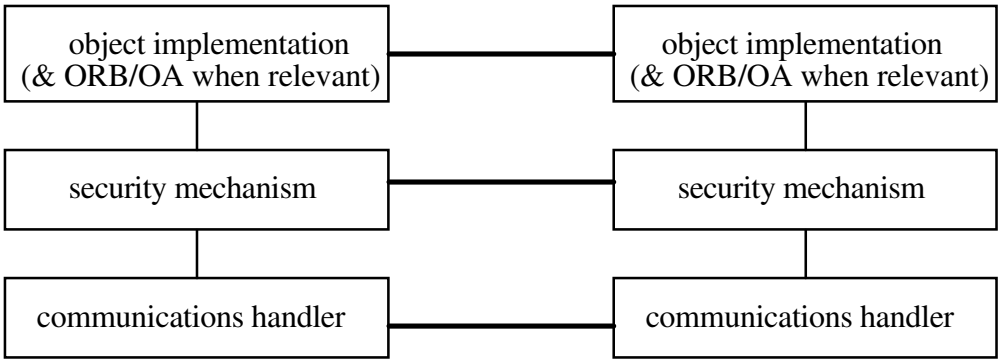
  The ORB (or supporting communications) may call encryption functions to do this in line with a quality of protection specified on invoking the object or a system default, to keep the object security unaware. Alternatively, the client and object may do this.

- A security aware object may get the user's identity and privilege attributes from the credentials and use these for access control and for auditing what the user has done.

An object implementation will often also act as a client and call other objects. When doing this, it may be acting on behalf of the principal which called it (so passes the security attributes of that principal) or on its own behalf (so passes information from its own credentials).

4.3 Implications on the Architecture for Interoperability

For interoperability of objects in different environments, three layers need to be considered:



- **Service Level Interoperability**: For objects (and the ORB/ OA which supports them) to interwork securely, the security policies must be sufficiently consistent. For example, they should use the same access control scheme with the same type of privilege attribute. (Otherwise, a gateway may be needed to translate the policy).

- **Interoperable Security Mechanisms.** For any particular dialogue, the same security mechanism (e.g. Kerberos - IEEE 802.10, Sesame, Netware 4) must be used for protecting the communications between systems (or a gateway will be needed). However, the authentication protocols and algorithms used may be transparently negotiated for this dialogue.

- I**nteroperable Communication Protocols.** The communication protocols used for secure dialogues must operate in a network in a transparent manner, i.e. existing protocols are able to co-exist and do not affect layers above or below.

4.4 Trusted Computing Base

Key security functionality should be confined to a trusted core known as a **Trusted Computing Base** (TCB) which cannot be by-passed. This is to:

- prevent (collections of) object implementations from interfering with each other and gaining unauthorised access to each others data.

- reduce the amount of software which is responsible for enforcing the security of the system and must therefore be trusted (and for more secure systems, evaluated)

- minimise the security functionality needed in clients and object implementations, and therefore the security skills needed by their developers

- enforce a consistent security policy across the system

The TCB should be kept to a minimum, but is likely to contain operating system(s), communications software (though note that integrity and confidentiality protection of data in transit is often above this layer), ORB, object adapters, security services and other object services called by any of the above during a security relevant operation.

Elements of the TCB will be distributed across the network and not all parts will be equally trusted. For example, an ORB on a machine which uses a B1 secure operating system may be more trusted than one which runs on a system which has not been evaluated. As discussed in section 2.2.1, the boundary of trust between objects is affected by the trust in the environments in which they run - CORBA as well as underlying system.

The establishment of trust between different parts of a distributed secure object system should normally be done between parts of the distributed TCB. However, it should also be possible for object implementations to establish their own level of mutual trust.

4.5 Interfaces

The requirements and architecture sections identify the need for security interfaces for:

- authentication of principals within the object system
- selecting a principal's privilege attributes for use in access control decisions. Note that not all OMA compliant secure systems need support privileges as well as identities. Also, a logon interface which allows specification of privilege attributes required at the same time as authentication may be desirable.
- acquiring credentials for a principal authenticated outside the object system
- recording audit events
- obtaining the principal's identities and privilege attributes and then using these for access control decisions (get_principal is relevant here)
- establishing and using security contexts, sometimes including controlling delegation. The (Extended) GSS-API standard is an obvious candidate for making secure associations, per message protection of data and handling attributes about principals.
- security administration

The OMA security model should include standard interfaces for security. There are potentially four levels of security relevant interfaces when a client invokes an object:

1) **Simple application level** - some interfaces could include security relevant parameters for applications which don't want to know much about security. For example, when the client calls the ORB to invoke an object, it may be desirable for the IDL (Interface Definition Language) to include a simple, optional "quality of protection" parameter.

2) **Security aware, but mechanism independent**. This is the level of interface used for setting up and managing security contexts. It is likely to be called mainly by the ORB or comms. This interface, like GSS-API, should be independent of particular security mechanisms so that ORBs, as well as objects, can use different mechanisms.

3) **Security Services Interfaces.** These are the interfaces to particular security services. In general, these should be hidden by the security context interfaces to prevent security mechanism dependence.

4) **Security service provider interfaces**. These are interfaces within the security infrastructure which supports the visible interfaces above. They are used to slot in different security mechanisms, including security services and different cryptographic algorithms. They are unlikely to be standardised in response to the first security RFP.

5 ADDITIONAL ISSUES

Responding to the security requirements will mean not only specifying interfaces to security services, but also specifying enough architectural and functional information to show that object systems which conform to it will be secure.

A response to the Security RFP should cover how it meets the requirements and:

- the overall security architectural model supported including its underlying philosophy
- the implications on the ORB and OMA architecture and interfaces
- the implications on other object services
- relationship with underlying operating systems and network security mechanisms

This section identifies the sort of issues which responses to the RFP will have to address in each of these areas. It is not intended to be a complete list of all issues.

5.1 Security model

- How are (human) users represented in the security model? What is the relationship between users, groups, clients, and principals? Can user information for the object subsystem be stored in existing, non-object user databases?
- How are "rights" or "privileges" represented in the system? Where and how are they visible? How do users perceive them - in particular, how do they relate to the ability to invoke specific interfaces or methods? What is their relationship to "principals"?
- Under what circumstances (and by what means) can rights/privileges be passed from one object to another? Is this explicit or implicit?
- What is the architecture's "trust model"? What parts of an OMA compliant system are assumed to be always trusted by everyone, and what parts have limited trust for which functions? Under what circumstances can an untrusted software component or hardware node be introduced into the system, and what threats does this introduce? (A discussion of trust should not be confined to object invocations.)
- Similarly, what is the trust model for users? How much power must any one user be given in order to administer the system?
- What range of security policies does an object implementation have? How much control does an object implementation have over whether an incoming invocation is allowed to proceed, or is denied? Where specifically can the security policy be implemented, and how does this relate to the trust model?

- What aspects of an object's security policy should be visible to OMG clients? How can an object's security policy be queried and possibly modified - what is the programming interface?

- Does the model support security policies for composites or collections of objects? If so, can this be instead of or as well as individual objects and what are the rules for handling these both when creating objects and when using them?

- What are the security implications of creating objects - statically and dynamically? Can the caller specify the security properties needed? Can these be inherited, and if so, from where? What are the implications of creating object classes, of overriding methods or customising existing objects?

- How are security mechanisms and cryptographic algorithms replaced?

- What level of security can be provided for security unaware objects which call no security interfaces? What default levels are provided?

5.2 Requirements

- What level of portability is supported for security aware applications? For example,

    - is a standard access control interface supported? If so, does it hide the actual access control information and mechanism used?

    - are standard attributes supported in a general authorisation service so administration of security information about objects can be done portably?

    - what portability to different security mechanisms does the interfaces provide?

    - can objects use cryptography for integrity and/or confidentiality protection of data (in addition to that for client/server data protection as described above)? If so, can they be ported to use different algorithms?

    - can audit analysis tools be ported? (are there audit administrative interfaces?)

- Is interoperability between objects in different domains or accessed via different ORBs addressed? If so, what assumptions are made about:

    - the security mechanism (e.g. Kerberos, SESAME, Netware 4) and associated protocols, PACs, etc. used? Is a choice or negotiation of mechanisms supported?

    - the security policies? Must be they be the same, for example, use the same type of privilege attributes (unless differences are handled by a gateway)?

- What TCSEC, ITSEC or other security criteria levels/classes will the response allow?

5.3 Implications on ORBS and CORBA Architecture

- What changes, if any, are needed to the OMA, and particularly CORBA architecture? For example, if the security model requires some security to be enforced in the ORB and/or OA, (for control of access to objects and auditing) how would the architecture need updating for this?

- What changes are needed to the interfaces e.g. the IDL concerned with invoking and activating objects etc e.g. can clients can ask for a particular "quality of protection"?

- Are security policies supported where some callers should not even know that certain objects exist, so browsing which objects are available is subject to access controls?

5.4 Implications on Other Object Services

- Is the response dependent on any other object services? If so, which are they, and what are the security implications? For example,

    - is a time service used to ensure that a set of credentials isn't outside its permitted time? If so, how is integrity of the time assured?

    - is auditing done using the Event Notification Service? If so, how does this meet the requirements for security of the audit trail in 3.2.3 above?

    - what is the relationship between the Authentication and Name Services in establishing the certified identity of a principal?

- Where object services are not used to perform such functions, how is security achieved (e.g. reliable time, secure audit trails)? How does distribution and installation of objects ensure the right software is installed at the right places without trojan horses?

- If other object services are to be included in a OMA compliant secure object system, what do they need to do to fit? For any object service, there may be general requirements on security functionality and architecture, for example:

    - how should it control access to its own functions and data to conform? Should it be auditing some of its actions?

    - does it keep its data secure enough? e.g. is optional encryption of stored data needed, does the store need to be cleared to prevent object re-use?

    - does it use ORB etc facilities to secure any communications as described above, or does it by-pass these and need to call on the appropriate security facilities directly?

    There may also be more specific requirements, for example, about consistency and integrity of information which is also used by the ORB and/or security services.

6 REFERENCES

The following documents are relevant:

For **security functionality and assurance criteria:**

- *Information Technology Security Evaluation Criteria (ITSEC)*
  The harmonised criteria of France, Germany, the Netherlands and the United Kingdom

- *Trusted Computer Systems Evaluation Criteria (TCSEC) and associated interpretations such as the Trusted Database Interpretation (TDI)*
  The Orange book gives the current US DoD security evaluation criteria and specifies levels. B levels are used in higher security systems, C in more standard ones.

  Draft US Federal Criteria have been produced. However, the Common Criteria now in preparation is expected to replace all of these, and will include information from them, and the Canadian Criteria. X/Open is also specifying commercial security profiles.

Distributed Security Frameworks:

ECMA TR/46 - *Security in Open Systems - A Security Framework*

ISO 7498-2 (CCITT X.800) - *OSI Security Architecture*

POSIX 1003.22/D0 *Draft Guide for POSIX Open Systems Environment - A Security Framework* Editor Dave Rogers, Data Logic; July 1993

X/Open Distributed Security Framework (XDSF)
Draft 2, February 1994

Interfaces

POSIX P1003.6 - expected to be the key standard for UNIX security, and already influencing interfaces though not yet a standard. Includes, among others, APIs for access control and auditing.

IETF RFC 1508 - *Generic Security Service Application Program Interface*
- John Linn, GZA, September 1993

X/Open Preliminary Specification P308 - *Generic Security Service API (GSS-API) Base* (P308 is a merged and reformatted version of IETF's RFC-1508 and RFC-1509, together with some additional material.)

X/Open Snapshot Specification S307 - *Generic Security Service API (GSS-API) Security Attribute and Delegation Extensions*

DCE Security AES - internal OSF document intended for X/Open