Verifiable Identifiers in Middleware Security

Ulrich Lang
University of Cambridge
Computer Laboratory
Cambridge, UK
ulrich.lang@cl.cam.ac.uk

Dieter Gollmann
Microsoft Research
Cambridge, UK
diego@microsoft.com

Rudolf Schreiner
ObjectSecurity Ltd.
Cambridge, UK
rudolf@objectsecurity.com

Abstract

This paper discusses the difficulties of describing an appropriate notion of the security attributes "caller" and "target" in object-oriented middleware systems such as CORBA. Middleware security needs such security attributes in order to be able to express middleware layer security policies. Our analysis points out that, whilst there is no information available on the ORB layer to describe the caller and target, it is possible in practice to use descriptors from other layers. In CORBA security, the mechanism-specific identifiers on the caller side and the information from the object reference on the target side turn out to be most appropriate and trustworthy for describing caller and target application objects at the right granularity. As a proof of concept we mention our MICOSec CORBA Security implementation which demonstrates the feasibility of our approach. Our paper shows that it is unrealistic to expect a security service layer to be able to abstract fully from the underlying security mechanisms without implications on granularity and semantic mismatches.

1. Introduction

One of the main purposes of abstraction in middleware architectures is the separation of the underlying layers from the application layer above to facilitate portability of application code, as well as enabling interoperability across differing underlying technologies. The CORBA security services specification is based on this idea and therefore tries to abstract the application logic from the underlying transport and security mechanisms. This should present the applications with a generic security service that facilitates portability, interoperability and flexibility.

Our analysis of the CORBASec architecture identifies a number of issues related to security information and their semantics on different layers of abstraction. In particular it is difficult to express caller and target accurately in the middleware security policy with the information available on the middleware layer. The problems discussed in this paper were encountered during the development of MICOSec [7], our CORBA Security Services implementation for the MICO ORB [3].

This section briefly reviews CORBA security and the terminology used. Section 2 describes the different layers of abstraction in a secure CORBA environment and identifies the boundaries of the middleware (ORB) layer. Section 3 evaluates the usefulness of a range of potential ways of describing the main ORB layer components: caller, message, and target. In section 4, our MICOSec CORBA Security implementation is presented. Finally, section 5 summarizes the observations of this paper, and a conclusion is given in section 6. Note that, although CORBA was chosen as an example for middleware, the observations of this paper apply to middleware in general.

1.1. CORBA

The Common Object Request Broker Architecture (CORBA) [8] enables client applications to transparently call other software objects across networks. This is achieved by mediating all remote method invocations through an Object Request Broker (ORB) which hides the complexities of large, dynamic, and heterogeneous distributed systems. This allows applications to invoke remote objects almost in the same way as in the (non-distributed) object-oriented programming paradigm. On a conceptual level the ORB is often referred to as a "software bus", analogous to a hardware bus which provides hardware devices with an abstract interface to the communications mechanism. The actual functionality of this conceptual entity is implemented by the ORB libraries on each node. CORBA

object interfaces are specified in a standardized Interface Definition Language (IDL), and objects can be located with Interoperable Object References (IORs). CORBA also specifies a number of additional object services such as naming, events, persistence, time, and security.

1.2. CORBA Security Services

This paper is focused on the CORBA Security Services specification (CORBASec) [9] which specifies the following security functionality components [2]:

1. Authentication: clients and targets can verify the identity of the other party;
2. Message Protection: data in transit can be protected from integrity and confidentiality attacks;
3. Authorization: access to objects can be controlled;
4. Audit: Audit logs can record which operations have been invoked by which clients;
5. Non-Repudiation (optional): irrefutable evidence of method invocations can be generated and verified (see [1]).

Some services are implemented by CORBA on the ORB layer (with the use of so-called interceptors), e.g. access control and audit. However, they rely strongly on the services provided by the underlying security technology, such as authentication and message protection. So instead of implementing all security functionality itself, CORBASec acts to some extent like an API which calls underlying security mechanisms such as Kerberos v5 [5], SESAME [11], and SPKM, through an interface modeled after GSS-API[1] [6]. Therefore the functionality offered by CORBASec is always limited by the functionality offered by the underlying security mechanisms.

The CORBA Security Services specification was first published in 1995 and consequently went through several updates to mitigate a number of discovered architectural problems, in particular regarding interoperability and portability. In version 1.5, SSLIOP, the SSL-Inter-ORB-Protocol was added to the specification to meet industry demand. The current draft version 1.8 comprises around 450 pages. There are also a number of additional security-related documents, most notably the Security Domain Management Membership Service [10] revised submission, and a final submission for Common Secure Interoperability v2 (CSIv2), which attempts to improve CORBASec interoperability, and an informal draft for an Authorization Token Layer Acquisition Service (ATLAS), which allows clients to pull CSIv2 authorization tokens in the correct format for the target-side from a token server.

[1] Secure Sockets Layer (SSL) is also widely used as a basic security mechanism for CORBA security, but it does not integrate well into the CORBA security architecture. This is because SSL works as a secure transport mechanisms, i.e. it establishes a network connection as part of the security context establishment. Therefore SSL has to be integrated as an alternative transport mechanism into the ORB. This way, the security context is set up automatically when the ORB opens a new network connection.

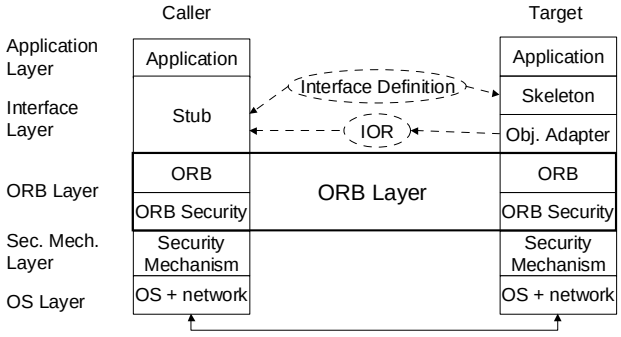

Figure 1. CORBA Layers

2. CORBA Abstraction

The CORBA architecture is based on a number of layers with interfaces between them that help achieve the basic CORBA design goals. Vertical interfaces provide abstraction, portability of applications, and flexibility regarding the underlying security mechanisms. There are also horizontal interfaces (i.e. standard protocols), which provide interoperability. For our discussion, only the vertical interfaces are of interest.

The purpose of this section is to identify the exact boundaries of the ORB layer and to show how it fits with the layers above and below. Figure 1 shows a generic secured CORBA application with a stack of layers above and below the ORB layer.

The bottom of the stack on either side consists of the underlying operating system and the network (OS layer). This layer handles all basic system calls and communications over the network. The next layer above (security mechanism layer) comprises what the CORBASec specification calls "underlying security technology". Note that CORBASec does not itself implement any authentication and message protection functionality, it merely provides the architectural abstraction to the underlying security mechanism. Examples for mechanisms on this layer would be Kerberos, SESAME, or SSL.

On the next layer (ORB layer), the CORBASec ar-

chitecture provides the abstraction from the underlying security mechanism layer. Despite the fact that some of the security policies and enforcement reside on this layer, in particular parts of access control, audit, and domain management, CORBASec relies on the underlying mechanism for most of its functionality. Typical abstractions on this layer would be the identities of caller and target.

The next layer above (interface layer) consists of client stubs and target skeletons, which contain the specific information about the target interface. Stubs and skeletons are generated as part of the IDL interface language mapping process [2]. In addition, this layer contains the target-side object adapter, which has several purposes: firstly it abstracts the target from the ORB and therefore enables portability of target objects across different ORBs, and secondly it generates Interoperable Object References (IORs) for the objects that are located above. These IORs are then used by the client to bind to the target. And thirdly, it ensures that each target object has an implementation of its interfaces.

Finally, on top of all this (application layer), the application logic is coded into the client stubs and the target skeleton. When the client application wants to invoke a method on the target, it first gets the target IOR, either from a naming or trading service, or by other out-of-band means. It then invokes the stub (giving the IOR as a parameter), and the request parameters are marshaled. Then CORBA does the rest: the information in the IOR is used by the ORB to locate the target application, and a connection is established. The request is transferred to the target side, where it gets unmarshaled and presented to the target application. After the target has completed its task, a reply goes back to the client on the same path.

One of the central concepts that characterize a layer is the description and granularity of the communicating parties and their associated (security) attributes. The granularity of the message content also plays a role. In this paper, the term identifier is used to denote a representation of the caller and target on various layers. On the middleware layer, such a representation is needed to express access control (and audit) policies. The caller is the initiator of an invocation, whereas the target is the entity that executes the method invoked.

For example, on the network layer the communicating parties are often represented by network sockets and messages are just opaque streams between these endpoints. On the security mechanism layer, an authentication process provides verified (mechanism-specific) identifiers for the participants. On the application layer, participants can be represented as objects or methods, and the message content is fully visible. The security attributes on the ORB layer are discussed in detail in section 3 below. It is important to ensure that the corresponding representations on all layers link together vertically in a semantically correct way – ideally, one would like one authenticated mechanism-specific identifier to correspond exactly to the identifier for each participant on the application layer, so that the functionality from the security mechanism layer could be propagated to the application layer. Of course this can only work to some extent, because the granularity of these descriptors tends to vary between the different layers. Figure 2 illustrates the fact that the ORB layer does not provide any useful representation for the identifiers available on the other layers.

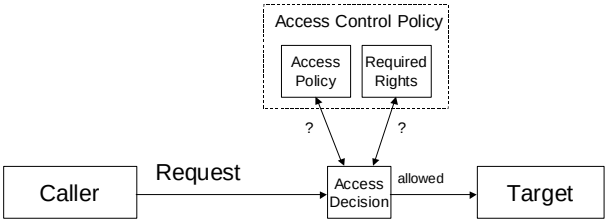

Figure 3. ORB layer Access Control

Any security system relies critically on correct and trustworthy representations of the participants. In CORBASec, the target identifier is used on the client side to authenticate the target to the client, but also on the target side to locate the security policies associated with the invoked target object when a request comes in. The caller identifier is necessary on the target side to authenticate the caller, and it is also used in the target's policies to locate the policy that should apply to a particular caller.

These authenticated identifiers are ultimately needed to express security policies, in particular for access control. Figure 3 shows how ORB layer access control and the related ORB layer policy objects are defined in the CORBASec specification.

Whenever a request arrives at the target ORB, the security service queries the *AccessPolicy* object for the granted rights associated with the request's mechanism-specific security context information[3]. Note that the implementation of *AccessPolicy*

[2] This paper only describes the case of static invocation. CORBA also supports dynamic invocation, but this feature is considered not to be important for our discussion.

[3] A security context is normally established as part of the authentication process. It represents (authenticated) security-related information about the remote participant.

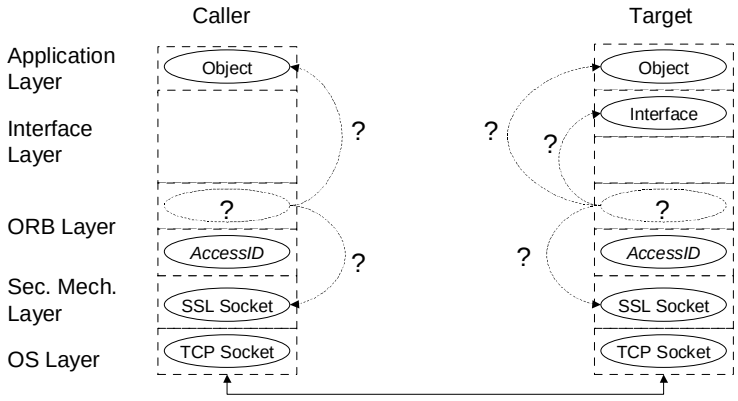

Figure 2. Example of vertical connections

Figure 4. Abstract ORB layer components

is mechanism-specific, although the object logically resides on the ORB layer. Also note that the *AccessPolicy* in CORBASec is the caller's capability. The security service also gets the required rights for the called target object type (this paper will show below why the object type is unsuitable for this purpose) and operation from the *RequiredRights* object. The *AccessDecision* object allows access only if the rights from *AccessPolicy* are sufficient to meet the *RequiredRights*. The *AccessDecision* object deals with access rights on a very coarse granularity: "get", "set", "use", and "manage" are the only specified rights that can be used to describe access to particular operations.

3. ORB Layer Security Attributes

In this section we examine the different parts of the middleware system from the ORB layer perspective. On a very abstract level, there is a calling instance (the "caller") that wants to send a request (the "message") to a target instance (the "target") , and a target that wants to send a reply message to the caller (see figure 4).

An appropriate description for these components is necessary in order to preserve the vertical link between the underlying technology and the application objects above, and at the same time provide a generic abstraction of the lower level representations. Note that both the object reference and the target interface do not appear on the ORB layer, they are situated on the layer above. Object location and authenticated identities of the caller and target are concepts of the layers below. Also note that CORBA does not provide unique naming of object instances as this is considered to be not scaleable. This is important for the following discussion on how these ORB layer entities are best described.

3.1. Caller Identifier

ORB layer caller identifiers are necessary in order to maintain the vertical link between the underlying technology and the application above, and at the same time preserve the ORB layer abstraction from the layers below to achieve portability, mechanism flexibility, and interoperability. We will now look at several ways to describe the caller.

There is no explicit representation of the caller on the ORB layer – the caller is a stub implementation that inputs an invocation (with an object reference that locates the target) into the ORB layer. Note that from an object-oriented programming perspective, the client is not even an object because one of the central definitions for an object is that it has an interface. CORBA clients do not have their own interface, they have a stub which contains the target interface. The ORB generates a message from the invocation and the object reference and forwards it to the target, and (by using a unique number) makes sure that the corresponding reply gets sent up into the same client application. The caller as such is therefore always anonymous on the ORB layer.

We will now look at other pieces of information, which could be made available on the ORB layer:

Interface Type, Object Reference

Firstly, object references and interface types are resident on the interface layer and not on the ORB layer. In addition, the interface type and the information from the object reference describe the target, not the caller. Therefore they are of no use to express the caller in the policy.

Mechanism-Specific Identifier

The authentication function of the underlying security mechanism normally refers to some mechanism-specific identifier for the communicating parties, for example cryptographic keys and X.509 certificates in the case of SSL. From a theoretical perspective, the mechanism-specific identifier belongs on the security mechanism layer and should not be propagated up into the ORB layer as that would break the abstraction and prevent interoperability and flexibility. Apart from that, there is also a semantic problem – the identifier used by the security mechanism does not always correspond exactly with a caller on the ORB layer. The remainder of this subsection will illustrate this semantic mismatch by comparing two different authentication mechanisms.

We first need to introduce CORBASec authentication, which (according to the specification) proceeds in the following steps (see [9], p. 15-58, 15-103):

1. The user normally logs onto some caller-side login GUI or uses specialized hardware (called the "User Sponsor") to authenticate to the *PrincipalAuthenticator*, which then generates so-called *Credentials* for the user. Although *Credentials* objects reside on the middleware layer, they contain (a reference to) mechanism-specific security information.

2. The target side does normally not have users to log on, therefore it generates its *Credentials* from another source, e.g. by loading a certificate from disk.

3. As part of security context establishment, CORBASec securely transfers the content of both *Credentials* to the other side where these peer credentials are stored. The security attributes from these peer credential objects can then be used for security enforcement, e.g. access control.

Figure 5 illustrates the steps involved in the CORBASec authentication from an application layer and ORB layer perspective.

Of course, successful authentication on the middleware layer of abstraction requires the involvement of the underlying security technology, which should be hidden from the application as it resides on lower levels of abstraction.

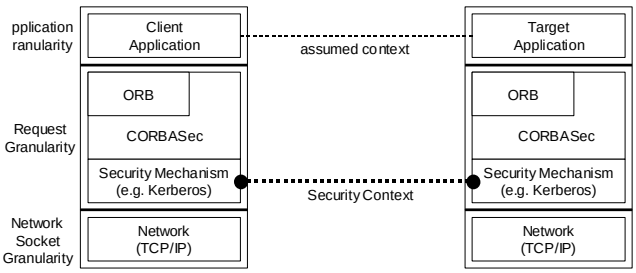

Figure 6. CORBA Security Context with Kerberos

When a secure CORBA client application tries to establish a security context with a target, the ORBs on both sides will therefore call on the underlying security mechanism to carry out unilateral or mutual authentication. The security attributes resulting from this mechanism-specific process will be stored in the *Credentials* objects on both sides. In theory, this mechanism-specific data is supposed to be invisible (often also called "opaque") to the ORB layer and to the application to facilitate interoperability and flexibility; however, in CORBASec it is not translated into mechanism-independent ORB layer security attributes because this would introduce semantic uncertainty. In other words, CORBASec uses mechanism-independent names for security attributes (e.g. AccessID or AuditID), while the content remains opaque mechanism-specific data. We will illustrate how the identity semantics change when the underlying security technology gets changed from Kerberos to SSL:

Figure 6 shows the position of authentication mechanisms like Kerberos in the layered structure from a granularity perspective. The Kerberos authentication library can be used with the security-enhanced CORBA protocol (SECIOP), which can support multiple security associations over the same transport connection. It could therefore be used by the ORB Security Service to transfer security attributes at a very fine granularity. In particular on the target side it would be possible to assign a separate Kerberos identity to each individual target object (provided the ORB implementation supports this).[4]

In practice, many CORBA applications use Secure Sockets Layer (SSL), a transport layer protocol, which

[4] However, in most implementations this multiplexing feature of SECIOP is just used to improve performance, i.e. there is only a single security context between the two endpoints of the TCP connection.

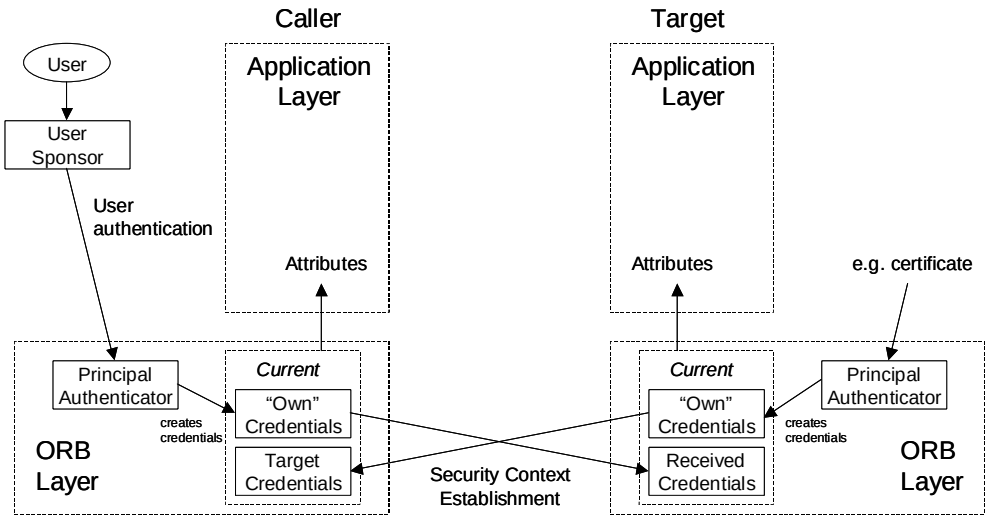

Figure 5. CORBASec Authentication

authenticates TCP connections between the network sockets of the participating hosts. This means that SSL cannot authenticate on an application object or operation granularity because SSL is tied to a connection on the network layer below the ORB layer – there could be a number of different application objects on the ORB that listen to the authenticated socket, and the caller has no way of authenticating the identity of the object it is invoking. Figure 7 illustrates how the situation changes: the security mechanism now only authenticates on a per [host, port] granularity, i.e. the security contexts end when they reach the socket on the destination hosts. This can cause problems if several application objects are running on top of one ORB which listens to a single port – the security mechanism is not able to identify the application layer objects and operations with the same granularity as in the previous example. For most real-world applications, this is more a problem on the target side than on the client side, as there is normally only one application object on the client-side, whereas there are often a number of objects with different security requirements on the target side.

These examples show that the meaning of "authenticated identity" changes when differing underlying security mechanisms are used which terminate the security context at different points on the path between caller and target. This clashes with the basic principle of abstraction, i.e. that the nature of the underlying technology is supposed to be hidden by the middleware layer[5]: the granularity and meaning of the security attribute "identity" can only be interpreted correctly with in-depth knowledge of the underlying authentication mechanism.

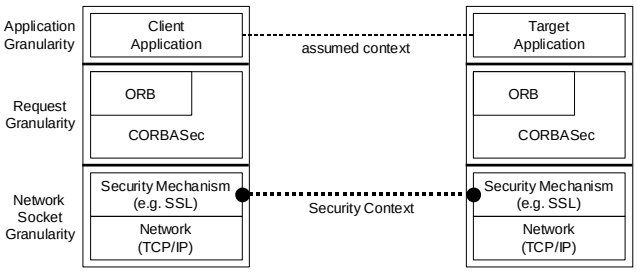

Figure 7. CORBA Security Context with Transport Layer Security

Conclusion - Caller Identifiers

In practice, using the identities provided by the security mechanisms is currently the only practical way to describe the caller in a trustworthy manner on the ORB layer – CORBASec therefore specifies the content of identities (e.g. AccessID or AuditID) to be mechanism-specific. This means that CORBASec callers can be described (with the granularity provided by the security mechanism), but not in an interoperable, mechanism-independent, and portable way. Our MICOSec implementation currently uses X.509 identities to describe

[5] Please note again that this is a theoretical point – CORBASec does currently not support mechanism-unspecific content for security attributes

callers in the security policy.

3.2. Target Identifiers

There are a number of concepts which could be used to describe the target:

1. the interface type of the target object
2. the object reference, which is generated by the target's object adapter once the target object is registered. It includes the server socket of the TCP connection and the target's object key. The object key contains the object adapter identifier and the ObjectID, which identifies the particular target object uniquely within the scope of each object adapter, but which are not necessarily unique across object adapter boundaries
3. the request header, which contains the target's object key and the invoked operation
4. the underlying identifier used by the security mechanism

In this section, we will analyze the problems associated with target descriptors, and present a possible way of expressing the target object in the security policy, and a way of locating security policies for their corresponding target objects.

Interface Type

As mentioned above, CORBA does not provide unique names for object instances. Instead the interface type could be used to describe objects based on their type. However, there are various drawbacks of this approach:

1. The interface type resides on the layer above the ORB and is therefore theoretically unavailable on the ORB layer. Despite that, making the interface type available on the ORB layer is only a minor violation of the layering of abstractions, in particular because a type name is a rather well-defined string.
2. However, due to CORBA's support for interface inheritance it is sometimes impossible to get the exact type, i.e. the Most Derived Interface (MDI) [4], of a target object. The object reference contains an attribute RepositoryID which contains a target object type, but it is not guaranteed that the object which carries out the operation is really of the type specified in RepositoryID, it could be possible that the parent base class of the object described by RepositoryID actually contains the called operation.
3. The target could have an interface which provides its type. However, for the same reasons as described above, this operation does not always give back the correct most derived type for the particular operation in question. In addition, the object would need to be activated before the policy can be applied to the invocation, which results in a "chicken-and-egg" situation.
4. The CORBA method "get_interface" can be used to get the type of an object. However this operation uses an Interface Repository (IR) to acquire that information, which is normally not suitable for the use within CORBASec for the following reasons: the IR is often not local and therefore slow; it uses CORBA invocations itself, which results in a chicken-and-egg situation; it would need to be trusted and therefore requires special protection; IRs are not available in all application environments.
5. Security enforcement based on the object type is not rich enough for most application scenarios. Security policies instead normally express principals (i.e. users) or the data inside application objects (e.g. a particular user's bank account).

Object Reference, Request Header, Domain Name

The target's object reference contains all the information necessary to locate the target object instance: the server socket to connect to, the object adapter identifier relative to the host, an object identifier relative to the object adapter. The server socket is, however, not always the socket of the CORBA server object. It also could be the server socket of a CORBA activation daemon which, when it receives a request, activates the real CORBA servers at run time and then forwards the requests to these servers. It also could be the server socket of a firewall. The object adapter name is generated when the object adapter is instantiated by the target application, and the object id is generated by the object adapter when the object reference is produced. The request header is available on the ORB layer and contains the information to locate the target object within the context of the TCP connection, plus the name of the invoked operation. Request headers will be described in more detail in section 3.3 below.

While it is in theory possible to describe the target with the information given here, it is hard in practice to locate the policy associated with the target object instance based on descriptors that can change whenever the object gets newly instantiated. To solve this

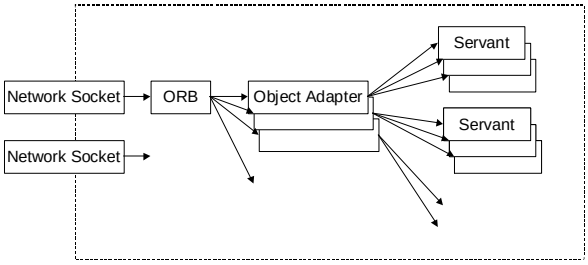

Figure 8. Target-side granularity

problem, our MICOSec implementation uses an object-to-domain mapper (as outlined in [10]), which maps several granularities of descriptors onto static domain names. Each object adapter has its own object-to-domain mapping service (it is therefore in the same trust domain). The actual policies can then be associated with its unchanging domain name, which accurately describes which target object a policy should apply to.

Mechanism-Specific Identifier

In principle, the problems of a security mechanism specific identifier are the same as on the caller side: loss of interoperability, mechanism flexibility, and portability. However, the granularity problem has a much higher impact on the target side than on the caller side, because each ORB can have several object adapters, which in turn have several target servants[6] (see figure 8).

If the authentication works on a per-socket basis, like in the SSL example above, then it might be impossible to draw any conclusions about the identity of the actual servant object. Therefore the identifier used by the security mechanism is often unsuitable to describe the target object in the target-side policy.

Conclusion - Target Identifiers

Authenticated mechanism-specific identities do often not describe the target accurately (i.e. at the right granularity) to the caller or inside the target-side policy. Thus the information from the object reference in conjunction with a domain mapping service (as outlined in [10]) seems to be the best option to accurately describe which policy belongs to each target application object.

[6] CORBA servants contain the target object interface implementations, whereas servers bootstrap the ORB, the object adapter, and the target application.

3.3. Messages

The ORB layer CORBA protocol is called General-Inter-ORB-Protocol (GIOP) and contains eight message formats, most notably the request message and the reply message. The interaction model seen by CORBA objects and GIOP is connectionless: a caller sends a request at any time, which causes a virtual function to be called on the target. Although connectionless, GIOP requests are dispatched over a connection-oriented transport, which means that the underlying operating system needs to take care of managing connections on behalf of callers and targets. Every GIOP request header contains a request_id, a unique number, which will also be included in the reply header to allow the caller to associate the response with the corresponding request. In addition, the header contains environment-specific endpoint information from the object reference, which is used by the underlying operating system to establish the connection to the target. GIOP is mapped onto the Internet-Inter-ORB-Protocol (IIOP) for environments where TCP/IP is used as the underlying protocol. In particular, this mapping specifies the encoding of IORs with three main components: the RepositoryID, the endpoint information (i.e. TCP/IP addressing information), and the object key.

Due to their connectionless and stateless nature, ORB level message descriptors are no useful security attributes. In addition, the abstraction layering discussed in this paper is irrelevant here because the notion of messages does not even appear on the application level. In security policies, messages are generally not expressed without any reference to the caller and target[7].

In the following, we will explore the usefulness of the GIOP message header (the message content is opaque on the ORB layer) as caller and target descriptors:

Request id

The request_id is generally an unpredictable number which cannot be used effectively as a caller descriptor in security policies – two separate requests from the same client will have different and non-correlated request_ids.

Request header (object reference)

The request message header information (which comes from the object reference) accurately describes the target servant (as discussed in section 3.2), but it uses

[7] Firewall rules could be an exception to that, but are considered beyond the scope of our discussion

some location attributes which are resident on lower layers in the abstraction stack.

Operation

The message header contains the name of the invoked operation, which can be used for more fine-grained security enforcement.

Underlying transport

The underlying transport layer is connection-oriented and therefore contains some endpoint information which can be used to describe the client and target applications. However, as illustrated in figure 8, the granularity on the transport layer can be too coarse to effectively describe the application objects.

Conclusion - Messages

In addition to the descriptors already discussed in 3.1 and 3.2, ORB level messages provide the name of the invoked operation. This is useful for fine-grained association of target-side policies with individual operations on target-side objects.

4. MICOSec: CORBA Security in Practice

As a proof of concept, ObjectSecurity Ltd. has developed MICOSec [7], an OpenSource CORBA security services implementation. MICOSec is based on the MICO ORB [3], a C++ implementation of the CORBA specifications. MICO was chosen for its transparent structure and its high degree of conformance to the CORBA standard. The free availability of the MICO source code was also an important facilitator for the integration of the security services into the ORB.

In the current version, MICOSec uses MICO's built-in SSL support for its authentication and message protection services. Both client-side user identities and target-side object identities are based on X.509 certificates, which can be managed by a public key infrastructure. In addition to the basic security functionality, we have implemented access control and security auditing on the middleware layer, which use domain names (as described in section 3.2.) to represent target objects in the security policy.

The mapping of target objects to domains is not done inside the ORB, but by a separate Object Domain Mapper (ODM), which is described as part of the upcoming OMG Security Domain Membership Management Service (SDMM) [10]. MICOSec's ODM is based on an earlier SDMM draft (published in 2000), which uses the following mapping information to support several levels of granularity: the server's X.509 identity, a pointer to the target object's POA, and the target object's ObjectId. In MICOSec's ODM, it was decided to use POA names instead of POA pointers, which has the following advantages:

1. POA names can be manually chosen, which allows the definition of a static mapping.
2. POA names are normally known before the application server starts, which allows the definition of the mapping in a configuration file before the application is launched. This way, the mapping can be defined easily for security-unaware applications.
3. POAs are arranged hierarchically, which can be reused well to represent the hierarchical domain name structure.

As MICOSec is based on SSL, clients are represented by their corrrsponding X.509 distinguished name (as described in section 3.1). However, SSL does not provide groups or roles, which makes it necessary to administer granted rights individually for each user. For scalability reasons, it is preferable to cluster users into groups, and then grant rights to these user groups. As a simple workaround, in MICOSec the OU (organizational unit) attribute of the X.509 certificate is mapped to the CORBASec "group" attribute.

MICOSec can be used both for security-aware and security-unaware applications. For security-aware applications, the X.509 certificates of client and server, the policies for setting up the security association between client and server, the access control and audit policies, as well as the ODM, are all defined in the application source code. For security-unaware applications, everything is set using command line arguments and configuration files, so that the application code can remain unmodified. This way, the security administration is segregated from the application development process, which is one of the main advantages of the CORBASec architecture.

5. Summary

This paper analyses the difficulties of describing callers and target applications accurately for expressing middleware layer security policies. For the discussion, CORBA and CORBASec were used as an exemplary architecture, but the basic problem applies to object-oriented middleware in general.

The first part of this paper defines the boundaries of the middleware layer, which in CORBA is called the

ORB layer, and showed how it fits between the layers below and above. It also defines an ORB layer model with the components caller, message, and target. We then look at different ways of describing caller and target in an accurate, trustworthy manner and at the right granularity. It turns out that there is no information available on the ORB layer of abstraction to describe the caller and target "identity", therefore we explore the use of information from underlying layers, in particular mechanism-specific identities and transport layer endpoint information from the object reference, and from layers above, in particular the interface type and operation name.

The analysis presented in this paper suggests that, whilst there is no information available on the ORB layer to describe the caller and target, it is possible in practice to use descriptors from other layers. In CORBASec, the mechanism-specific identity on the caller side and the information from the object reference on the target side turn out to be most appropriate and trustworthy for describing client and target application objects at the right granularity in the target-side security policy.

The paper also gives a brief overview of MICOSec, our OpenSource CORBA security services implementation, to demonstrate how the observations of this paper can be applied in practice. In particular, MICOSec illustrates how the target information from the object reference can be mapped to domain names, and how client identities can be mapped to groups.

6. Conclusion

CORBASec tries to solve the middleware security problem on several layers. Both authentication and message protection are implemented by widely used security mechanisms, which reside below the ORB layer, e.g. SSL and Kerberos, and their exact nature is abstracted by CORBASec. The two other main functional components, namely access control and audit, are implemented on the ORB layer and – because there is no suitable notion available on the middleware layer – rely on the security attributes established by the underlying security mechanisms, in particular the authenticated identifiers for the caller and the target. Often the granularity and semantics of these identities do not match with the representations required to express effective ORB layer access control and audit policies. Therefore the whole concept of ORB level separation from the underlying security technology breaks: introducing the middleware layer not only separates the application from the underlying network, it also separates the security problem from the security solution.

References

[1] ISO/IEC 10181-4. *Information Technology – Open Systems Interconnection – Security Frameworks for Open Systems: Non-Repudiation Framework – Part 4.* ISO/IEC, 1997.

[2] ISO 7498/2. *Basic Reference Model, Security Architecture.* ISO, 1989.

[3] MICO Group. *MICO User's Guide.* 2000.

[4] P. Humenn. *Summary of MDI Discussions.* OMG SecSIG Mailinglist, 1999.

[5] J. Kohl and C. Neumann. *RFC 1510: The Kerberos Network Authentication Service V5.* 1993.

[6] J. Linn. *RFC 2078: Generic Security Service Application Program Interface, Version 2.* 1997.

[7] ObjectSecurity Ltd. *MICOSec User's Guide.* 2001.

[8] OMG. *CORBA Architecture and Specification.* OMG, 1998.

[9] OMG. *CORBA Security Services Specification, v1.8.* OMG, 2000.

[10] OMG. *Security Domain Membership Management Service, Final Submission.* OMG, 2001.

[11] T. Parker and D. Pinkas. *SESAME V4 – Overview.* 1995.