An Access Control model for CORBA

Mirel Muresan [1,2]
BCD Romextera
P-ta Consiliul Europei 32C
2500 Alba – Iulia
ROMANIA
m.muresan@mailcity.com

Iosif Ignat [1]
Faculty of Automation and Computer Science
Technical University of Cluj – Napoca
Gh. Baritiu 26-28
3400 Cluj-Napoca
ROMANIA
iosif.ignat@cs.utcluj.ro

Abstract

Integration of security and object-oriented techniques is critical for the successful deployment of distributed object systems. Object Management Group published a security service specification called CORBA Security to handle security in object systems that conform to the Object Management Architecture[2]. This paper provides a definition of the authorization part of CORBA Security using a Typed Access Matrix (TAM) model. The dependencies among the authorization elements are analyzed and possible interpretations for access control decision and audit function are given.

Key words: authorization, CORBA, distributed object systems, types, roles, access control

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. In addition, the middleware abstraction layer should make it possible to replace the underlying technology without affecting the application code. 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. The CORBA Security Services specification[1] 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. There are also a number of additional security-related documents, most notably the Security Domain Management Membership Service revised submission, and a final submission for Common Secure Interoperability v2(CSIv2), which is supposed to fix interoperability problems of CORBASec, and an informal draft for an Authentication Token Layer (ATLAS), which is supposed to fix interoperability problems of CSIv2. The design, specification, and implementation of security policies and the management of the corresponding access rights at runtime are both error-prone and security - critical. There are few methods or tools that provide adequate support for application designers and security administrators in distributed object systems. The main problems are ensuring scalability while at the same time allowing the description of fine-grained accesses, which requires appropriate grouping constructs. A related problem is manageability. To make large numbers of fine-grained access rights manageable, it is necessary not just to group these rights but also to provide abstractions that represent the underlying policies.

In this paper, we are concerned with the support for specifying access control policies, using a TAM matrix model[3] which introduces the notion of ***type***. In general terms, an access control policy is a description of which accesses are allowed and which are denied. In a more technical, but still abstract sense, an access control policy is a set of rules that, when parameterized with access control information, is evaluated by an access decision function to yield a boolean result, i.e. an access is either allowed or denied[4]. We take a lot of function which have the role of specifying and managing access control policies and that allow

administrators to deal with abstractions that are adequate for their tasks.

Because we cannot rely on the actual identity of users, as they are not known in advance, we have ascribed them the ***role***, where roles represent a combination of user groups with sets of authorizations, or just a set of authorizations. In other words, roles are sets of users and group principals based on their common aspects in different interaction contexts.

Role names for sets of users are declared in role clauses and assertions which express requirements on the authentication service used to authenticate users in roles.

The contributions of this paper are to type the elements of CORBA Security Service for access rights. The rest of this paper is structured as follows. Our access model is presented and discussed in section 2. Section 3 contains a realistic example for a policy with dynamic right changes and section 4 represents the conclusions of this work.

Roles

To support development, deployment, and management of policies in potentially diverse environments, we cannot rely on the actual identity of users because they are not known in advance. With regard to object invocations, the most suitable abstraction for users is that of a role as it allows us to concentrate on the specific, logical function in which a principal is operating on application or system objects. Our notion of role is different from the widely-used role concept in role-based access control (RBAC) where roles represent a combination of user groups with sets of authorizations[12] .

Roles are sets of users and group principals based on their common aspects in different interaction contexts. We assume a public-key based service like SSL[6,13,11] that issues privilege attribute certificates that represent role membership to principals upon request and whose signature is trusted by the access decision function. A security service like OMG can then manage access sessions between callers and objects so that objects always see "users in roles" rather than individual users. A deployer of a policy must check that role membership is certified in accordance with the assertions expressed in the policy. If existing roles do not map well, new roles need to be set up in the authentication service.

2. The Authorization Specification

In this section, we develop a specification that defines these semantics of access control in Corba.

2.1 Structure

Let O denote the set of objects, let D denote the set of domains and I denote the set of object interface which require that an object has at least one interface . We assume a relation ***domain*** $:O \times D[5,1]$ that assigns objects to domains and relation ***interface:*** $O \times I$ [5,1]gives the interfaces to which an object belongs, in both case is verify relation $[D,O] \subset \{T, F\}$, $[d,o]= T$, $o \in d$ and $[O,I] \subset \{T,F\}$, $[o, i]= T$, $o \in i$.

But, an object instance may have a lot of names. We denote IO the set of names of objects and assume the relation ***instance_ name*** $:O \times IO \rightarrow I.[5,1]$

Let M denote the set of methods, well defined by an interface and its name, in which case we may say that a tuple <m,i> unic identify an operation. In conclusion we denote a relation ***operation*** $: M \times O[5,1]$ that assigns methods to objects and IM denote the set of operation names and we assume a relation ***operation_name*** $: I \times IM \rightarrow M.[5,1]$

Let A denote the set of privilege attributes and A' the set of privilege attributes in different delegation states used in state delegate. Finally, the authorization units is the set $AS = A \cup A'$.

We consider $\Re$ the set of rights, its elements may be noted with *r. RightFamilies* subdivided into sets of rights types.

Let ***Mode*** $=\{$***allow, deny***$\}$ and we observe that a right is well defined by an operation, a permission and a family of rights, in conclusion we may define $\Re =$ ***IM x Mode x RF*** where ***RF*** is the set of rights contained by ***RightFamilies*** .

U is the set of principals or users, the active entities in the system. The relationship between users and privilege attribute types is defined outside of the CORBA specification. Let PA(u) denote the set of attribute types assigned to the user u, which implicitly determines that the state is of initiator or delegate.

Let us define a function for the users of the system named ***Subject :U x ROLE x D → {T,F},*** where **ROLE** is the set of roles of the subjects in the system.

In the model the are two types of semantics, like: *Disjunctive semantics(Any Right)[1], Conjunctive semantics(All Rights)*[1]. In conclusion the operations requested by an object are typed by the semantics of rights.
The semantics have two types *AND* and *OR*, when an operation is well-defined by a triplet like {s, im, i} where S={ AND, OR} , s∈S, im∈IM and i∈I.

On the other side an object may by transient or persistent. An persistent object contains an indefinite repetition of request on it and a transient object does not contain repetition on it. The history of a persistent object may be length, for example an Account object may have a hundred of credit and debit operations. In this case the dynamic separation of duties gives to the subject the ability not to perform two conflicting operations on the same object and optimizes the audit operations. Separation of duties can be enforced by keeping the following history information: the entire history of transient objects, a partially fixed long history of persistent objects for non-repetitive portions of the transaction.

In conclusion, let ***IT*** be the set of object types which has elements *{persistent, tranzitiv}* and we may define an interface instance by a triplet like { it, io, i}, where it ∈IT, io∈IO and i∈I. With the above descriptions of the elements of the CORBA Security Model we can give a formal definition of its structure.

A CORBA Security state is a four-tuple (SUB,OBJ,T,AM) where SUB=AS∪O∪M and OBJ=I∪U∪D are analyzed as follows:

- SUB is the set of subjects
- OBJ is the set of objects, SUB⊆ OBJ
- T : OBJ → T is the type function which gives the type of every objects. AM is the access matrix, with a row for every subject in SUB and a column for every object in OBJ. We denote the contents of the (S,O) cell of AM by [S,O]. We have [S,O]⊂ℜ

Where :

$\Re$={s, g, m, u, parent, cread, etc}
and

$\Im = \Im_{obj} \cup \Im_{sub}$ where :

$\Im_{obj}$ = I∪U∪D like { Account, Bank, U_1, U_2, U_3, OSS_TAM }
and

$\Im_{sub}$=AS∪O∪M like {Access_id:U_1,Access_id:U_2,Access_id:U_3,U3,Bank, Debit,Credit,Balance,etc}

Also we may say like that well-formed CORBA Security state[5] is a state(S,O,T,M) in which the access matrix satisfies the following condition:

$$\forall o \in O,\ d \in D : d \in domain(o)$$

We can give a concrete definition of relations as follows:

domain :O x D→ {T,F}, defines the relation between objects and domains.

interface: O x I → {T,F} define the relation object interface

operation: M x O x S → IM define the relation methodes, objects and semantics into the set of operations unic identified

rights_in_domains: D x 2^{AS}→ 2^R define a function mapping a set as_1,as_2..as_l of privilege attributes (where ∀i, 1<i<I ; as_i ∈AS) in a domain dj ∈ D (where 1<j<n) to a set of rights r1,r2,..rp where ∀i, 1<i<p ;r_i ∈R) that are effects for the given set of attributes and roles [9].

object_rights:(2^D x 2^R x $\Im$obj x ROLE) → 2^R define a function mapping sets of rights returned from ***rights_in_domains*** for every domain in D, of the objects ***type*** from domain in function of role ro_k∈ ROLE (where 1<k<s), to a set of effective rights.

By this functions, we identify the partition RR=M[M,I] that defines the rights required to invoke an operation, and partition GR = M[AS,D] that defines the set of rights granted to privilege attributes. Another advantage is the possibility of using the policy for control of information by interface, ***ORCON*** (originator controlled). The ORCON policy[3] require that the creator (i.e. originator) of the object retains control over granting access to the information to the object.

We assume this relations ***ORCON:*** $(2^D x ASx \Im_{obj} x 2^R \ x \ ROLE) \rightarrow 2^R \ x \ 2^{AS} x \ ROLE$ and a control function of types of domain level by roles ***Role_Control :*** $\Im_{obj} \ x \ 2^R \ x \ ROLE \rightarrow \Im_{obj}$.

Another function is ***Audit_Control:*** $\Im_{obj} \ x \ M \ x \ O \rightarrow \{T,F\}$ which realizes an efficient management of audit types of system, like in [7,8].

System state

A system protection state is usually not constant. Objects and subjects are added or deleted from a system, and rights may be granted and revoked for the purposes of delegation of responsibility or as part of an application - specific security policy. We distinguish the following case:

- discretionary granting or revocation, using the access matrix which contain the rights modified explicitly by the security service.
- dynamic granting or revocation is performed implicitly by the service, using the function ORCON
- delegation occurs implicitly during the course of an operation invocation when the target object delegates the call to another object.

3. EXAMPLE

We presents an application- specific policy for a system that supports banking operation on a bank. This system is a simple workflow application and supports the following design:

Application design

We consider two interfaces of type **Bank** and type **Account** in the domain typed ***OSS_TAM*** where the first manages the accounts with its operations and the second manages the account of the client.

```
interface Bank{
        Account create( in string name, in float
balance ) raises ( AlreadyExist );
        void remove( in string name ) raises
        ( NotFound );
        Account get( in string name ) raises
( NotFound );
};
```

```
interface Account{
        readonly attribute string name;
        void credit( in float value );
                void debit( in float value ) raises
( UnauthorizedBalance );
        float balance();
};
```

The object of type *Bank* has the right to create objects of type *Account* from the strings of input by calling *create()*. After it has created the object it may give it the reference by calling *get()* or remove the object by calling *remove()*.

The object type *Account* allows credit, debit and list of the balance of bank client.

Policy design

The roles present are: ***BankManager, BankUser, BankClient*** with the assertion :

- $BankClient \subseteq BankUser \subseteq BankManager$
- $BankClient \cap BankUser = \varnothing$; $BankManager \cap BankUser = \varnothing$
- cardinality(*BankManager*) = 1

We see that a membership in *BankManager* implies membership in role *BankUser* and a membership in *BankUser* implies membership in role *BankClient.*

Static Policy

The role ***BankManager*** controls objects of type *Bank* and *Account* , ***BankUser*** controls objects of type *Account* and ***BankClient*** partially controls objects of type *Account.* (see the access matrix)

Dynamic policy

The most interesting of this policy are the changes in the protection state when the user in role *BankClient* of type U3 may be calling only *balance()* operation and *credit()* no; from then on, they may. Thus, the access permitted be depends on earlier accesses, similar to the Chinese Wall[10]. To describe transitions like these that are directly connected to changes in the application state, we use the ***ORCON*** function.

OBJ SUB	Access_id:U 1	Access_id:U 2	Access_id:U 3	Account	Bank	OSS_TAM	U1	U2	U3
Access_id :U1						CORBA:g,u,s ,m ORB:own	T		
Access_id: U2	ORB: cread		ORB: parent			CORBA:g,u,s ,m		T	
Access_id:U3				Get from U2 CORBA:g,m		CORBA:u			T
Create: AND					CORBA:g,u,s,m ORB:own				
Remove:AND					CORBA:g,u,s,m ORB:own				
Get :AND					CORBA:g,u,s,m				
Credit:AND				CORBA:g,u,m					
Debit:AND				CORBA:g,s,m					
Balance :AND				CORBA:u					
BankManager				T	T		T		
BankUser				T				T	
BankClient				T					T

This function permits the user in role *BankUser* of type U2 to change the rights of a user in role *BankClient* for a while, because the user of type U1 in role *BankManager*, who is the owner of the objects type Account, allows to the user of type U2 in role *BankUser* the right created.

4. CONCLUSIONS

Existing midlleware technologies are necessary but not sufficient for the effective protection of the resources of distributed enterprise applications. In this paper we suggested an authorization specification using Typed Matrix in CORBA Security.

We have tested this model using a ***Visibroker*** product with the SSL authentification protocol and has been implemented in Java.

The authorization model does not limit the types, it can be different and have multiple types like IDL interface, domains, users, privilege attributes. They may be combined, added or erased as wish you want and confaire an dynamic model in the side of rights.

Because it is a discretionary model, using the matrix we may modify the rights in an explicit mode. The dynamic rights are implicit (by **ORCON** function) and the other use the delegate.

The information protections it not part of the implementation and does not depend on the environment, the model separates the specific policies from the development of application.

The ***types*** are used for protection, the model is fine-grained and offers the possibility to group and create a framework as the person pleases.

This model leaves open all design decisions about how implicit authorizations are derived, how rights propagate in groups, which conflict resolution strategies are used and how priorities are employed.

The separation of authorization and application logic simplifies the development of both distributed systems and their security functions and therefore makes it easier to enchange their quality.

Equally important, it paves the way for the uniform use of authorization mechanisms across (heterogenous)system boundaries, as well as for centralizing enterprise security administration and management, traditionally time consuming, costly.

5. REFERENCES

1. OMG. ***CORBA Security Service Revision*** 1.7. November 2001
2. OMG.***The Common Object Request Broker : Architecture and Specification,*** Revision 2.3 June 1999
3. *Ravi S. Sandhu.* ***The typed access matrix model.*** In Proc IEEE Symposium on security and Privacy, pages 122-136, 1992
4. *Ulrich Lang,* ***Access Policies for Middleware,*** Technical Report, University of Cambridge Computer Laboratory, ISSN 1476-2986, May 2003

5. *Gunter Karjoth, **Autorization in Corba Security*** In Proc ESPORICS'98 pages 143-158,1998
6. *Key Tools SSL*, Http://www.baltimore.-com/keytools_ssl.pdf
7. *Ravi S. Sandhu, Pieragela Samarati, **Authentication, Authentication, Access Control and Intrusion Detection,*** The Computer Science and Engineering Handbook, CRC Press 1997
8. *Ravi S. Sandhu, Pieragela Samarati Authentication, **Access Control and Audit,*** ACM Computing Surveys, Vol. 28, No 1, March 1996
9. *Konstantin Beznosov, **Engineering Access Control for distributed Enterprise Applications***, Florida International University, Miami, Florida, 2000
10. *Sushil Jajodia, **Access Control***, INFS 762, Fall 1999
11. *B.Lampson,M.Abadi,M.Burrows, şi E.-Wobber, "**Authentication in Distributed Systems: Theory and Practics***", In Proceedings of ACM Symposium on Operating Systems Principles, Asionar Conference Center, Pacific Grove,California,1991
12. Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein and Charles E. Youman ***Role-based access control models***. IEEE Computer, 29(2):38-47,1996
13. *Roland L.Rivest ,**Cryptology**, MIT Laboratory for Computer Science, Cabrige, Massachuttes 02139USA.*