20

MANDATORY SECURITY POLICIES FOR CORBA SECURITY MODEL

Carla M. Westphall, Joni da S. Fraga, Carlos B. Westphall and Silvia C. S. Bianchi
Network and Management Laboratory (UFSC-CTC-INE-LRG) &
Control and Microinformatic Laboratory (UFSC-CTC-DAS-LCMI)
{carla, westphal, silvia}@lrg.ufsc.br, fraga@lcmi.ufsc.br

Abstract: This paper proposes extending the CORBA (*Common Object Request Broker Architecture*) security model to make possible the use of mandatory policies in distributed applications. The *Bell & Lapadula* model is adopted to define the mandatory controls in the authorization scheme *JaCoWeb*, through a policy service designated as *PoliCap*. Our mandatory control is carried out on the level of ORB (*Object Request Broker*), on the client side, preventing, in unauthorized accesses, the emission of the corresponding requisition, the associated processing on the server and also, the generation of new requests through this unauthorized processing.

Key words: Security, Mandatory security policies, Authorization scheme, CORBAsec.

1. INTRODUCTION

The authorization scheme *JaCoWeb* (*http://www.lcmi.ufsc.br/jacoweb/* and *http://www.lrg.ufsc.br/~carla/secpoli/*), uses the concepts and interfaces of CORBA security service – CORBAsec [1]. The *PoliCap* policy service constitutes the first level of verification and operates in binding time between the client and the server. The *PoliCap*, provides centralized management of policy objects (discretionary policies in the first versions [2]), in a domain of distributed objects applications. A second level of access control based on the *capability* mechanism is carried out in the execution ime of the application, reflecting the policies of *PoliCap* allowing, in this

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35586-3_46

M. A. Ghonaimy et al. (eds.), *Security in the Information Society*

way, verifications on both sides – the client and the server – to invocations of method.

The work presented in this paper describes our experience in extending the authorization scheme *JaCoWeb* to implement the controls of mandatory policies. We use as a base in the construction of mandatory policies the model *Bell & Lapadula (BLP)* [3, 4]. Even though BLP may have a large number of critics [5, 6], it still motivates a large number of researchers, in the sense of minimizing their limitations [7, 8, 9].

In order to provide a basis for discussion about our experiences, in section 2 of this text, the CORBA security authorization model is described. Section 3 presents the model Bell & Lapadula, used to incorporate mandatory functionalities in *CORBAsec*. In section 4 the authorization scheme *JaCoWeb-Mandatory* is presented. Results of implementation are shown in section 5 and some conclusions are described in section 6.

2. AUTHORIZATION IN CORBASEC

In CORBAsec [1], the security policies are described in the form of *security attributes* of the system resources (*control attributes*) and of the principals (*privilege attributes*). The *DomainAccessPolicy* object (*Table 1*), represents the access interface to a *discretionary* authorization policy, granting to a set of principals a specified set of rights to perform operations on all objects in the domain. To simplify administration, *DomainAccessPolicy* aggregates principals for access control by using their privilege attributes as subject entries. Some types of grouping are *group* and *role*. Just four types of rights: *g* (*get*), *s* (*set*), *m* (*manage*) and *u* (*use*) - that belong to the *corba* family - are defined in CORBAsec specification.

The *RequiredRights* object (*Table 2*), determines that for the invocation of each operation in the interface of a secure object, some rights are necessary or required (*control attributes*).

All access decisions of object invocations are made through a service object interface known as *AccessDecision*, which determines whether or not an operation to be executed by a specified target object is allowed. The access decisions rely on privilege and control attributes provided by *DomainAccessPolicy* and *RequiredRights* respectively. The access decision logic could be specified in different forms, but it is dependent on the context of the system and on the type of policy used. For example, the policy defined in *Table 1* grants a principal *bank_teller*, the required rights – *g* and *u* – to execute the operation *Deposit* of the *Checking_Account* interface.

A CORBAsec *interceptor* causes the transparent deviation of a method invocation, activating a corresponding COSS (*Common Object Services*

Specification) service. The *Access Control Interceptor*, on higher level, causes a deviation to carry through the access control in the call.

Table 1. DomainAcessPolicy object.

Privilege Attribute	Granted Rights
Role: bank_manager	corba: gs--
Role: bank_manager	corba: g---
Role: bank_teller	corba: g—u

Table 2. RequiredRights object.

Required Rights	Operation	Interface
Corba:g	See_Balance	Savings_Account
Corba:gs	Deposit	Savings_Account
Corba:g--u	Deposit	Checking_Account

3. THE BELL AND LAPADULA SECURITY MODEL

The *Bell & LaPadula model* adds *mandatory* access controls to discretionary access controls, impeding the information flow from the highest security levels to the lowest security levels [3, 4, 8, 10].

In *BLP*, the clearance of a subject (f_s) and the classification of an object (f_o) assume levels defined in the classification of *DoD* (Department of Defense): unclassified, confidential, secret or ultra-secret. The security level of a subject (f_s) represents the maximum security level of information that the subject may consult. In the dynamics of the BLP model, a second label is associated with a subject - the current security level, noted by f_c. The current level represents the highest information level consulted by the subject in the system; this last level fluctuates, therefore, with the evolution of the system.

The model Bell & LaPadula defines two basic properties [3]:

- *Simple Property*: known as *property-ss*, which separates the reading accesses of a subject only to objects whose levels may be *dominated* by its security level, that is, $f_o \leq f_s$.
- *Star Property*: also known as **-property*, determines that a subject may read only objects *dominated* by its current security level and may write in objects that *dominate* its current security level, that is, $f_c(s) \leq f_o(o)$.

The *overclassification of information* [10, 11], is the result of the restrictions imposed by the *-property, and represents one of the major problems of the BLP model. Some techniques are suggested in the literature to work with or to soften the overclassification problem [3, 5, 8, 11, 12].

3.1 Techniques to Avoid Overclassification

The concept of *trusted processes* was introduced as an implementation of the concept of trusted subjects [5]. In [8] *trusted objects* is defined - identified as *stateless objects* - which, following the example of *trusted*

subjects suggested in [12], have associated intervals of confidence $[L_{min}, L_{max}]$. L_{min} represent the *minimum*-security level of data that the stateless object can w*rite*. L_{max} corresponds to the *maximum*-security level, which can be *read* by the stateless subject.

A *stateless object* is an object that holds no datum of the application in the memory between two successive activations [8]. Its state is reinitiated on each of its invocations. This property of *stateless objects* is very important, since it means that two successive requisitions that access the same stateless object *cannot carry out an information flow* with this object.

Process servers of the system in general are *stateless objects*. An example is a file server object that responds to the requisitions for mounting file systems without holding any information about a previous session in the execution of the next requisition.

In the classic study of [11], the author suggests *initializing any recently created object with* f_O = *UNCLASSIFIED*. During the execution of any subsequent operations, the security level of the object is to "fluctuate" so as to represent the sensibility level of the information that is stored inside of it.

The modification of security labels, carried out in a dynamic way by the system itself, which seeks to avoid the overclassification of information using the aforementioned techniques, enables a system to comply with a variety of security requisites much greater than those models with static labels [8, 11]. However, this modification must be made with restrictions, so as to prevent illegal flow of information.

4. THE MANDATORY JACOWEB *PROPOSAL*

The *Mandatory JaCoWeb* objective is to extend the existing interfaces in CORBAsec, using the Bell & Lapadula model, to incorporate mandatory policies, non-existent in the current standard [1].

Our work, like that of [8], uses the *stateless objects* and the *intervals of confidence* associated with the requests in order to avoid overclassification. We also use the technique of initial sensibility to assume the value of UNCLASSIFIED and fluctuating labels suggested by [11]. However, our proposal of mandatory controls is inserted in a distributed objects environment. In our approach, the *activation of mandatory controls* is carried out o*n the client side*, before the requisition starts the communication with the server object.

The entities that comprise our model are: *subjects*, *objects* and *requests*. A subject represents an entity that provokes the occurrence of operations in information. An object is an entity destination of the operations that can be executed by the subjects, and may or may not have an associated state.

In the model a *security label* is attributed to every subject or object of the system, distinguishing between *stateful* and *stateless* objects. In order to control each interaction among objects, each request (representing the classification of information that is being transported) should also be labeled.

4.1 Subjects

Each subject receives a security label f_s, representing his/her *clearance* in the system. To incorporate the clearance of subjects in the structure of CORBAsec, a column of data called *Clearance* is included in the object *DomainAccessPolicy*. The content of *Clearance* may assume one of the following values: UNCLASSIFIED (= 1), CONFIDENTIAL (= 2), SECRET (= 3), ULTRA-SECRET (= 4). In this way, the object *DomainAccessPolicy* takes the representation of *Table 3*, where clearance (or security level) SECRET is associated with the subject *bank_manager*.

Table 3. DomainAccessPolicy object with the Clearance column.

Privilege Attribute	Granted Rights	*Clearance*
Role: bank_manager	corba: gs--	3

4.2 Objects

Security labels are attributed to *stateful* and *stateless* objects. A *single label* f_0 is associated with a stateful object *O*. This label represents the classification of the object. This security level is fixed and cannot be modified during lifetime of the object.

An *interval of confidence* $[fmin_0, fmax_0]$, where $fmin_0 \leq fmax_0$, is associated with the stateless objects *O*. The label $fmax_0$ represents the *maximum* security level of the information contained in the stateless object *O*. Likewise, the label $fmin_0$ represents the client's *minimum* security level to be able to write in this stateless object.

Our proposal is to represent an *interval of confidence* using two *digits*: the first digit represents the $fmin_0$ level and the second digit represents the $fmax_0$ level. Considering the security levels with numerical values 1, 2, 3 or 4 (UNCLASSIFIED=1, CONFIDENTIAL=2, SECRET=3, ULTRA-SECRET=4), the interval [CONFIDENTIAL, SECRET] is represented by the value 23.

Considering that the information about the objects application servers in CORBAsec are contained in the object *RequiredRights*, to associate security labels with these objects, our proposal includes a new element in the object *RequiredRights* called *Classification*. This extension consists of a column that contains numerical values containing three digits:

- first digit: represents the *mode of access* to the method considered (1 – reading, 2 – writing, 3 – reading/writing)
- second and third digits: represent the *classification* of stateful objects and *intervals of confidence* associated with stateless objects:
  - if the object is *stateful*, we have the following values: UNCLASSIFIED = 01, CONFIDENTIAL = 02, SECRET = 03, ULTRA-SECRET= 04
  - if the object is *stateless*, the values associated with the intervals of confidence are the following: 11, 12, 13, 14, 22, 23, 24, 33, 34, 44.

Table 4 presents the object *RequiredRights* with the insertion of the column *Classification*, where the operation *See_balance* of interface *Checking_Account* has as its access mode the value equal to 1 (reading), and it is a stateful object with the classification UNCLASSIFIED. The operation *Print_balance* of the interface *Printer_I1* is a reading/writing operation (value equal to 3) and it is a stateless object with an interval of confidence equal to 12 ([UNCLASSIFIED, CONFIDENTIAL]).

Table 4. RequiredRights object with the Classification column.

Required Rights	Operation	Interface	*Classification*
Corba:g---	See_balance	Checking_Account	101
Corba:gs--	Print_balance	Printer_I1	312

4.3 Requests

Each *request* in the system transports information that might be modified by the different objects accessed [13]. In our model, a security label is attributed to the request, characterizing an *interval of confidence* represented by [$fmin_r$, $fmax_r$]. The element $fmin_r$ represents the *classification of information* contained in the request and that can be written in an object. The classification of the first activation of a request created by a user is initialized with the security level = UNCLASSIFIED, does not contain any sensitive datum, following the suggestion of [11]. The element $fmax_r$ of a request represents its *clearance*. It is initialized with the current security level of the principal that activates the request and represents the maximum classification of information that can be read by this request. These two elements, then, compose the security label of the request: *[classification of the information, clearance of the request]*.

The concept of *fluctuating labels* [11] can be used during the execution of a request. The label associated with a request, depending on the server object of this request, will not necessarily be the same as the response or as new request generated from an associated processing.

Suppose there is a user with the clearance SECRET, interacting with a stateful object - *Object 1* (classification=02) and with a stateless object *Object 2* (classification=23). The user makes a request r_1, which accesses *Object 1* to execute one of its methods. This execution may result in sending a new message, the request r_2, and this new request is labeled, taking into consideration the information from r_1 and from *Object 1*. The original request label r_1 has the value = 13, that is, UNCLASSIFIED as its classification (first activation of request), and the value SECRET as its clearance, reflecting the user who invokes the operation on *Object 1*.The method invoked on *Object 1* activates another method to be performed by *Object 2*. The request *r2* is then labeled [CONFIDENTIAL, SECRET], since the classification of the object making the request is now CONFIDENTIAL and the request transports sensitive information from *Object 1* to *Object 2*.

The coding of the labels in the model under consideration uses the CORBA *Request* structure that contains the field of security label.

4.4 The Rules of the Mandatory Authorization Scheme

The rules of the authorization scheme in the *JaCoWeb* project define the evolution of the model and how mandatory controls are established. For our implementations six rules are considered that deal with the control of the authorization of a *request r* that calls for an *object O*.

In presenting rules of the authorization scheme, references will be created followed by modes of access to reading and writing. The *reading access* on a *stateless* object *O* is the execution of a method that released a flow of information about state of object *O* to the request. Likewise, the *writing access* represents the execution of a method of object *O* that releases a flow of information from the request that executes the method to the state of the object *O*. The *reading-writing access* represents the execution of the method that causes an exchange of information in both directions between the request that executes the method and the state of the object *O*.

The security labels used in the verifications are associated with subject (label present on object *DomainAccessPolicy*), to object (label present on object *RequiredRights*) and to request (label present in *Request* CORBA).

The information that is generated from the processing activated by a *request r* will be transported in the *reply* (return to client) or in a *new request*. Both the reply and the new request are represented as *request r'*. The label associated with this *request r' is* generated according to the label of *request r* and of the *object O* that was invoked.

We also have to consider the behavior of the two types of objects identified in our model: stateful objects and stateless objects.

4.4.1 If Object O is a Stateful Object:

- **Rule 1** (*derives from rule 1 of BLP model [3]*): If method *m* corresponds to a *reading* access and if $f_O \leq fmax_r$, the access is *authorized with restriction*. The $fmin_r$ that will represent the execution of a request will be adjusted for the value $max(fmin_r, f_O)$. This rule translates the fact that the current level of information on the output of the processing is adjusted to the level of the object read. When $f_O \leq fmin_r$, the access is *authorized without restriction because* $max(fmin_r, f_O) = fmin_r$.
- **Rule 2** (*derives from rule 4 of BLP model [3]*): If the method *m* corresponds to a *writing* access and if $fmin_r \leq f_O$, the access is authorized without restriction.
- **Rule 3** (*derives from rule 3 of BLP model [3]*): If method *m* corresponds to a *reading-writing* access and if f_O is situated in the interval [$fmin_r$, $fmax_r$], the access is *authorized with restriction* and $fmin_r$ must be adjusted to the $max(fmin_r, f_O)$ value. The restriction results from the reading operation; writing, however, is authorized with no restriction. In the case where $f_O = fmin_r$, access is *authorized with no restriction.*
- **Rule 4** (*derives from rule 8 of BLP model [3]*): *Creation of stateful objects*: A subject typically creates an object invoking a *request r* of the type "Create an instance" for a class of objects to be created. In order to avoid covered channels, and to ease the overclassification of information [11], the *object "son"* must have, as an initial label, a value equal to *at least the classification of the request r that* generated this creation (defined by the lowest element of the *request r label*). Thus, the stateful object is created with $f_O = fmin_r$.

4.4.2 If Object O is a Stateless Object:

- **Rule 5** (*derives from rule R2 of [8]*): If $fmin_O \leq fmax_r$ and if $fmin_r \leq fmax_O$ access is *authorized with restriction* (intersection of intervals of confidence is not empty). The information generated through activated processing will be transported in the *request r'* of output with label given by [max ($fmin_r$, $fmin_O$) , min ($fmax_r$, $fmax_O$)]. When the interval [$fmin_r$, $fmax_r$] is included in the interval [$fmin_O$, $fmax_O$], the access is authorized without the restriction of an adjustment of the *request r'*, since: $max(fmin_r, fmin_O)=fmin_r$ and $min(fmax_r, fmax_O)=fmax_r$.

One very important matter to be considered in the mandatory controls concerns the return messages from the invocations [13, 14].

– **Rule 6** (*rule proposed in Mandatory JaCoWeb*): *Return from requests executed on stateful objects.*

It is important to note that, in all the rules presented previously, the restrictions that are established are applied equally to the return messages or the *request r'*. The *requests r'* may be generated through processing executed by stateless and stateful objects.

- The *stateful object* does not conserve information about requests, and, if the invocation from a method of a stateless object *O* is authorized, the return from this *request r* will necessarily be authorized: the interval of confidence for the *request r'* that effects this return is automatically included in the confidence interval of the calling object.
- In the case of *stateful objects*, a *request r*, invoking a r*eading* or a r*eading-writing* operation, on an object *O*, may cause another *request r'*. This new *request r'* represents the return message and causes a w*riting* on the calling object. In this case, it is necessary to control this writing just as in a call of a writing method from any object. The *request r'* has unrestricted writing access to the calling object in case it meets one of the following conditions:

– If the calling object is a stateful object: $fmin_{r'} \leq f_{\text{CALLING OBJECT}}$;
– If the calling object is a stateless object: $fmin_{r'} \leq fmax_{\text{CALLING OBJECT}}$; and
– If none of these conditions are met by the *request r'*, writing access is denied.

5. IMPLEMENTATION RESULTS

An implementation prototype of the *Mandatory JaCoWeb* project was developed. An application consisting of a bank system composed of a CORBA server object and a Java client applet was constructed. The CORBA server object was developed with the tool JacORB 1.3 (*http://www.jacorb.org*), and the client applet was implemented with the tool JDK 1.3. The prototype is limited to a single name domain. *Figure 1* synthesizes the functionalities implemented in the prototype.

The experiment implements mandatory and discretionary policies, based on CORBAsec structures, by verifying access control *on the client side.*

The *DomainAccessPolicy* and *RequiredRights* are modified to contain the corresponding mandatory controls. To the *DomainAccessPolicy* object the following methods are added: *grant_clearance*, which inserts the subject's clearance level; *revoke_clearance*, which removes the subject's clearance level; *replace_clearance*, which substitutes the subject's clearance; *get_clearance*, which consults about the subject's clearance. To the object *RequiredRights*, are added the methods *set_classification*, which inserts the

classification of the object method and *get_classification*, which consults about the classification of the object method.

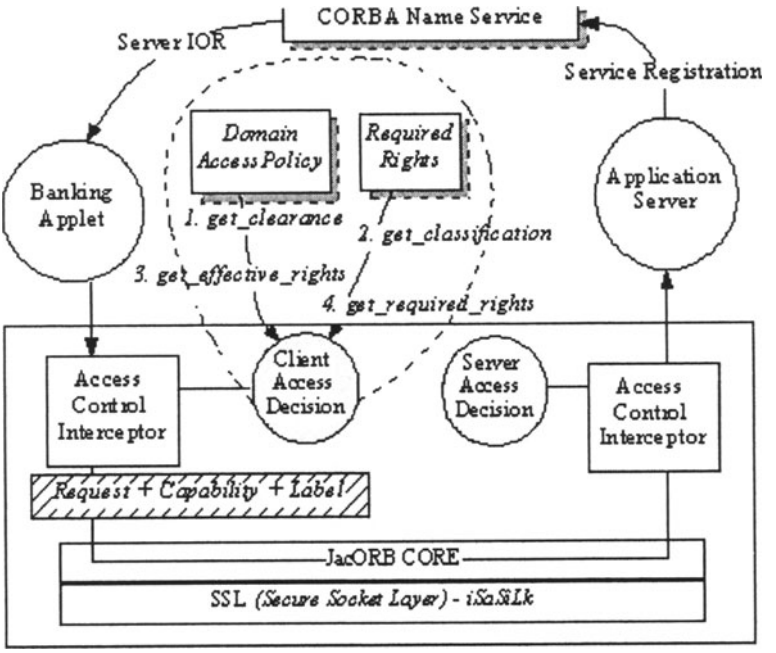

Figure 1 .The structure of prototype implemented.

The evolution of the labels of mandatory policy takes place as the binding process of objects, client and server, occurs. In *bind time* the mandatory policy, along with the discretionary one, is verified in global form, through interception on the client's side, on the level of access control, turning back to *PoliCap*. Once this policy has been verified, local objects are assembled on the client's side. On assembling the local objects, the mandatory policy comes to be represented in the local *DomainAccessPolicy* and *RequiredRights* objects.

In the prototype, the access control interceptor in the client machine invokes the *ClientAccessDecision* object. The method *access_allowed* of *ClientAccessDecision* object implements the rules that define the security labels evolution of the mandatory policy of the environment. Initially, this method performs the mandatory controls (*Figure 1* - steps 1 and 2), obtaining the clearance of the principal and the classification of the object, invoking respectively, *get_clearance* and *get_classification* methods. It also verifies which of the six rules of the scheme can be enforced for the request that is being made. In the sequence, the *access_allowed* method performs the normal discretionary controls according to CORBAsec model (*Figure 1* - steps 3 and 4). It obtains the required rights invoking the method

get_required_rights of the *RequiredRights* object and obtains the granted rights by the *DomainAccessPolicy* invoking the method *get_effective_rights*. It compares the required rights and the granted rights with the privilege attribute to decide whether or not the method to be invoked can be executed. On the server side, the *ServerAccessDecision* object verifies capabilities and labels to see if the message sent is correct.

One of the experiments made in our banking system prototype, was a *read* request of a stateful object named *Simple_Account*, with $f_O = 3$, on a stateful object named *Special_Account*, with $f_O = 4$. The label request was defined as $[fmin_r, fmax_r] = [3, 4]$, since the user who has invoked the request to the *Simple_Account* object had $f_s = 4$. Enforcing rule 1, the read access is allowed with restriction, that is, $fmin_{r'}$ must be changed to value 4. So, in this way, the *r' request* receives the following confidence interval values $[fmin_{r'}, fmax_{r'}] = [4, 4]$, giving rise to a blocking of the response since $fmin_{r'} > f_{Simple\_Account}$. As the adjustment of the labels uses rule 6 to verify the possibility of returns of requests performed on stateful objects, still in *ClientAccessDecision* object *on the client side*, this blocking does not occur in our project. Therefore, this request had its access denied since the response message could cause a system blocking.

6. CONCLUSIONS

The work developed by [7] proposes an initial, quite simplified idea, of the inclusion of mandatory access control policies in CORBAsec. The work of [8] presents a mandatory authorization scheme, using, to avoid the overclassification of information, the techniques of the stateless objects and of the confidence intervals associated with the trusted processes. Its approach does not have rules for the creation of the objects and for treating the return messages in a mandatory scheme. Other works relate the use of the Bell and Lapadula model with the general object-oriented programming model [13, 14].

The proposal of mandatory authorization policy for CORBA security model fills in a lack in that platform. Comparing the work proposed here with the existent literature, we could verify that the *Mandatory JaCoWeb* proposal treats the mandatory policies in CORBAsec, as in [7], but suggests and uses techniques to avoid the overclassification of information.

To implement the mandatory policies in our authorization scheme, we simply had the inclusion of a field in the *DomainAccessPolicy* object (clearance of the subject) and of a field in the *RequiredRights* object (classification of the method of the object).

One of the differences in our proposal, in contrast with other works of the mentioned literature [8, 13], is the *materialization of the mandatory controls accomplished on the client's side*, before the request begins the communication with the server object. This provided the existence of a clear definition of the rule that deals with return messages, a matter that is usually dealt in an informal way. The implementation of this rule in our scheme does not provoke blockages of returns of requests processed, as it can happen in [8, 13, 14]. Our work defines six rules that show the way as security labels develop in the system, using techniques to avoid the overclassification of information and considering an implementation environment.

7. REFERENCES

[1] OMG. Security Service:v1.5, OMG Document Number 00-06-25, June 2000.

[2] C. M. Westphall, An Authorization Scheme for Security in Large-Scale Distributed Systems, Doctoral Thesis, CPGEEL-UFSC, Brazil, December 2000.

[3] D. Elliot Bell e L. J. LaPadula. Security Computer Systems: Unified Exposition and Multics Interpretation. MITRE Tech. Report MTR-2297 Rev. 1, March 1976.

[4] L. J. LaPadula and D. E. Bell. Mitre Tech. Report 2547 – Vol. II. Journal of Computer Security, v. 4, issue 2/3, p. 299-323, 1996.

[5] C. E. Landwehr, C. L. Heitmeyer and J. McLean. A Security Model for Military Message Systems. ACM Trans. on Computer Systems, v. 9, n. 3, p. 198-222, August 1984.

[6] J. McLean. Reasoning About Security Models. In: Proceedings of the 1987 IEEE Symposium on Security and Privacy, p. 123-131, Oakland, California, April 1987.

[7] G. Karjoth. Authorization in CORBA Security. In: Proceedings of the Fifth ESORICS, LNCS, p. 143-158, Springer-Verlag, Berlin Germany, September 1998.

[8] V. Nicomette and Y. Deswarte. An Authorization Scheme for Distributed Object Systems. In: Proc. of the IEEE Symposium on Security and Privacy, p. 21-30, Oakland, California, 1997.

[9] S. Osborn, R. S. Sandhu and Q. Munawer. Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies. ACM TISSEC, v.3, n.2, May 2000.

[10] C.E. Landwehr. Formal models for computer security. ACM Comp. Surveys, v.13, n.3, p. 247-278, Sep. 1981.

[11] J. P. L. Woodward. Exploiting the Dual Nature of Sensitivity Labels. In: Proc. of the IEEE Symposium on Security and Privacy, p. 23-30, Oakland, California, 1987.

[12] D. Elliot Bell. Secure Computer Systems: A Network Interpretation. In: Proceedings of the 2nd Annual Computer Security Application Conference, p. 32-39, USA, 1986.

[13] J. K. Millen and T. F. Lunt. Security for object-oriented database systems. In: Proceedings of the 1992 IEEE Symposium on Security and Privacy, p. 260-272, Oakland, California, May 1992.

[14] S. Jajodia and B. Kogan. Integrating an object-oriented data model with multilevel security. In: Proceedings of the 1990 IEEE Symposium on Security and Privacy, p. 76-85, Oakland, California, May 1990.