21

POLICAP - PROPOSAL, DEVELOPMENT AND EVALUATION OF A POLICY SERVICE AND CAPABILITIES FOR CORBA SECURITY

Carla M. Westphall, Joni da S. Fraga, Michelle S. Wangham, Rafael R. Obelheiro and Lau C. Lung
Network and Management Laboratory (UFSC-CTC-INE-LRG) &
Control and Microinformatic Laboratory (UFSC-CTC-DAS-LCMI)
carla@lrg.ufsc.br, {fraga, wangham, obelix, lau}@lcmi.ufsc.br

Abstract: This paper presents Policap - a Policy Service for distributed applications that use CORBA security model. Policap was proposed for insertion in the JaCoWeb Project context, which is developing an authorization scheme for large-scale networks based on CORBA security standards. The contribution of this paper is the combination of client-side and server-side access control, in a single domain. In this paper, operations of security management not currently included in the OMG standards are also proposed. The paper further presents the implementation results obtained and an evaluation of these results based on Common Criteria, ISO standard 15408.

Key words: Security Policies, Authorization Schemes, Security Evaluation, CORBA.

1. INTRODUCTION

Management of security policies in large-scale systems is a great concern today [1]. The specifications of the CORBA (*Common Object Request Broker Architecture*) Security Service (CORBASec) [2], if implemented and managed properly, can provide a high level of security for information and applications in large-scale environments. According to the CORBASec specifications, when an object is created in a distributed system, it automatically becomes a member of one or more security domains. However, a detailed analysis of the CORBASec specifications shows that

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*

they still lack procedures for managing domain members and security polices, which are essential for applications [2, 3, 4].

The *PoliCap* policy service, proposed in this paper, aims to provide a policy object management service centralized in a domain of distributed object applications. Our proposals meet the need identified in CORBAsec related to policy object management and were developed in order to act in the model of the *JaCoWeb* project [5].

Initially, the paper presents the security model of the CORBA standard in section 2. In section 3, the policy service proposal and the authorization scheme considered are described. Implementation results are shown in section 4 and the prototype evaluation developed is made in section 5 according to the Common Criteria of Security. Section 6 presents some conclusions and related work.

2. CORBASEC – CORBA SECURITY MODEL

The CORBA security model [2, 4] relates objects and components on four levels of a system: the *application level*; the *middleware level* formed by service objects (COSS: *Common Object Services Specification*), ORB services and the ORB core (*interceptors* implement ORB services and cause the transparent deviation of a method invocation, activating a corresponding COSS service); the *security technology level* composed of the underlying security services; and finally, the basic protection level formed by a combination of operating systems and hardware functionalities.

In CORBAsec, 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 2*), 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 1*), 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 2* grants a principal *bank_teller*, the required rights – *g* and *u* – to execute the operation *Deposit* of the *Checking_Account* interface.

Table 2. DomainAcessPolicy object.

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

Table 1. RequiredRights object.

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

3. JACOWEB SECURITY FRAMEWORK

The *JaCoWeb* (*http://www.lcmi.ufsc.br/jacoweb/*) aims to use CORBA security model integrated with Web and Java security models to compose an authorization scheme for distributed applications in large-scale networks.

The *authorization scheme* defines two security control levels: the *global level* and the *local level*. These two levels are actualized in COSS service objects and in security nodes and TCB's (*Trusted Computing Bases*), respectively. The service objects concentrate functions of identification and authentication of users and authorization controls in the access of visible objects on the global level. The security nodes and TCB's, present in each machine of the system, validate the ways of access to the local resources. *Figure 1* shows the main components that implement the *JaCoWeb* framework.

The *PoliCap* is a policy service for distributed objects whose invocations are ruled by the CORBAsec model. *PoliCap* was designed within the context of *JaCoWeb* project and corresponds to a first level of access control verification in the authorization scheme. The second access control level corresponds to a *capability* mechanism. The emphasized boxes are the main contribution of our work, considering the original CORBAsec model and are an evolution of our previous work (*www.lrg.ufsc.br/~carla/publica.html*).

The application applet, after its authentication, interacts with the CORBA name service, to obtain, from the name of the object, the reference or IOR (*Interoperable Object reference*) of the server application object. This must allow the binding with the server object. The calls executed by this application applet on a remote application server are subjected to two levels of access control. On the higher level, the verification occurs in *binding time*, and once the requisition is validated, the *PoliCap* policy service provides

versions of the *DomainAccessPolicy* policy objects and of the *RequiredRights* object that are used locally in the validation of access requests to the application objects. From this high level verification, capabilities are generated which are validated locally in the remote servers. To construct these two levels of access control, we use the two defined interception levels on CORBA security model and its service objects.

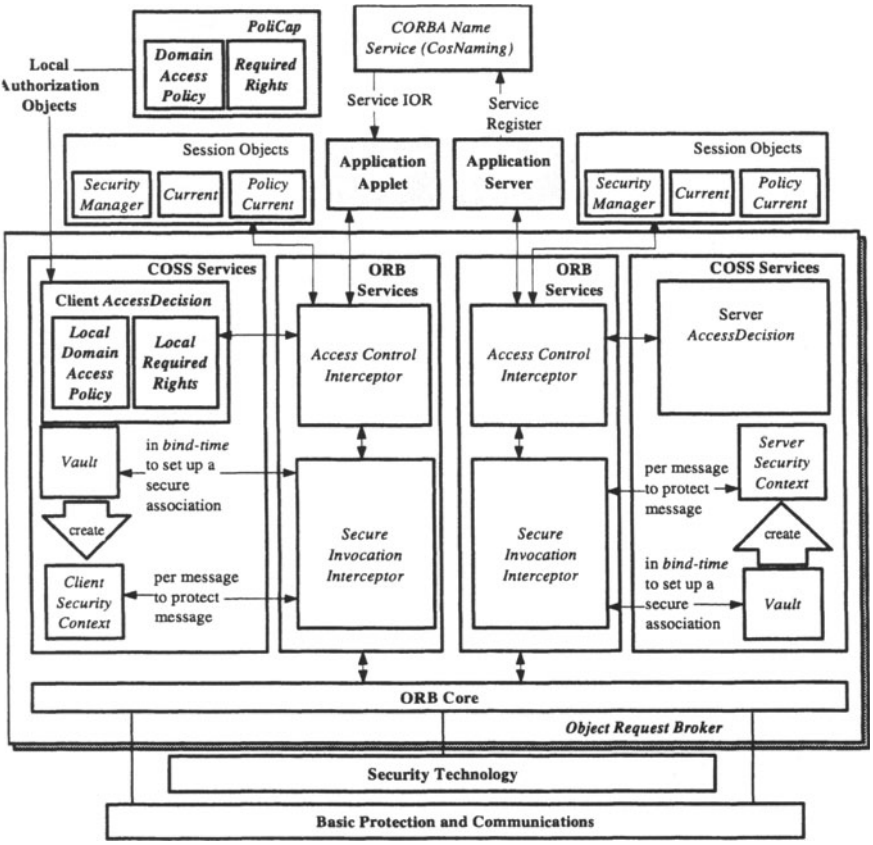

Figure 1. JaCoWeb Authorization Scheme.

3.1 PoliCap

The *PoliCap* policy service, proposed in this work, sets out to provide the central management of policy objects in a domain of distributed object applications, filling in the existing gap of policy object management in CORBAsec specifications.

The security policy domains are formed by a collection of object references that have a set of common security policies and are managed by a

domain manager object [3]. However, interfaces to add new policy objects to domains or to modify domain membership are still not standardized. The initial idea to fill in this gap is being proposed in the *Security Domain Membership Management Service* [3], where an extension of the *DomainManager* interface is proposed, with administrative functions that would serve to define and remove policy objects of a domain.

Even with these interfaces not yet standardized in CORBAsec, we felt the need to introduce this policy object management service in the authorization scheme proposed. The service developed – the *PoliCap* – is based on initial documents released by OMG [3] and surely will not be too far from the specifications that are to be standardized shortly. The *PoliCap* is a service that offers operations, both for *administrative* and *operational* functions concerning policy objects, playing the roles of *domain manager for policy objects* (*DomainAccessPolicy*) and of *rights manager for required rights objects* (*RequiredRights*) in our domain.

Administrative applications interact with *PoliCap* to manage policies and required rights and, *operational applications* or COSS objects, interact with the policy service to obtain, in *binding time*, policies and required rights necessary for the controls over a method invocation in execution time. The idea is that, in *binding time*, the policy service is to provide the *DomainAccessPolicy* and *RequiredRights* objects that act on an invocation. The PoliCap IDL module defines two interfaces. The *DomainAccessPolicyAdmin* interface has the following operations: *set_policy*, *delete_policy* and *get_local_domain_policy*. The *RequiredRightsAdmin* interface has the *get_local_required_rights* operation.

The operation *set_policy* associates the authorization policy defined and the corresponding policy object with the policy domain. The operation *delete_policy* removes the authorization policy from the domain.

The operation *get_local_domain_policy* sets up locally, *in binding time*, a version of the *DomainAccessPolicy*, essential for the validation (*in access decision time*) of several operations present in the same interface. For example, getting the *DomainAccessPolicy* object with the information of *Table 2* - the global object defined in the domain - to execute the operation: localDomainAccessPolicy=get_local_domain_policy(SecClientInvocationAccessDiscretionary, 'role,authority,bank_teller',initiator), the return of this operation is the *DomainAccessPolicy* object formed only by the third line of *Table 2*. This object has to act locally in the *AccessDecision* object of an invocation.

The operation *get_local_required_rights* also sets up a local version, with the required rights present in the global *RequiredRights* (centralized) object of the domain policy service. This version of the *RequiredRights* object, set up locally *in binding time*, contains all rows related to an interface, considering that a client object can execute several operations defined in

these application object interface, for example, having the *RequiredRights* object (global) in *Table 1*, executing the operation: localRequiredRights = get_local_required_rights (Savings_Account), the returned value of this operation is the *RequiredRights* object (local), formed only by the first and second lines of *Table 1*.

3.2 Capabilities Using CORBAsec

In *JaCoWeb*, on the first level, the verifications are made by the client object *AccessDecision* in partnership with the *PoliCap*. The local objects *DomainAccessPolicy* and *RequiredRights* are part of the structure necessary in the client object *AccessDecision* to generate *capabilities* to be verified in the server object *AccessDecision*. High-level verification is made only once, during the *binding* of the client and server objects, on the first request. The requests of subsequent operations on the server interface will be validated only by the capability mechanism.

In the *JaCoWeb* scheme, the *capabilities* are created dynamically at each client request and use the *Request* class of the CORBA specifications.

A *capability* in *JaCoWeb*, following [6], contains in its fields: the *IOR of the server object*, representing the identity of the entity over which a *capability* provides the access rights; *method requested*, representing the right; *identifier of the principal*, representing the identity of the entity requesting the operation; and a *nonce* that assures the *'freshness'* property of the request. In a *Request* for a usual invocation in CORBA, the *IOR* of the server object and the identification of the method requested are already present. The other two *capability* fields, identifier of sender of request and a *nonce* field, must be inserted in *Request* during the high level interception, in the object *AccessDecision* on the client side. The value of *nonce* is calculated as follows: *nonce* = SHA (identifier of sender, method requested, server IOR, Random). The value *Random* is calculated by means of the *java.security.SecureRandom* class of the Cryptographic API of Java. This random value guarantees the *freshness* property of the *Request*, providing protection against *replay* attacks. The initial *Random* value is obtained during the secure association establishment, from the *Policap*, and is known by client and server, acting as a session key between them. The posterior values of *Random* serve as a message counter. The identifier of the one who requested the operation is obtained from the client credential.

4. JACOWEB IMPLEMENTATION

A prototype – including the *PoliCap* policy service, the *capability* mechanism and SSL (**S**ecure **S**ocket **L**ayer) as the underlying technology - was developed in our laboratories (*Figure 2*).

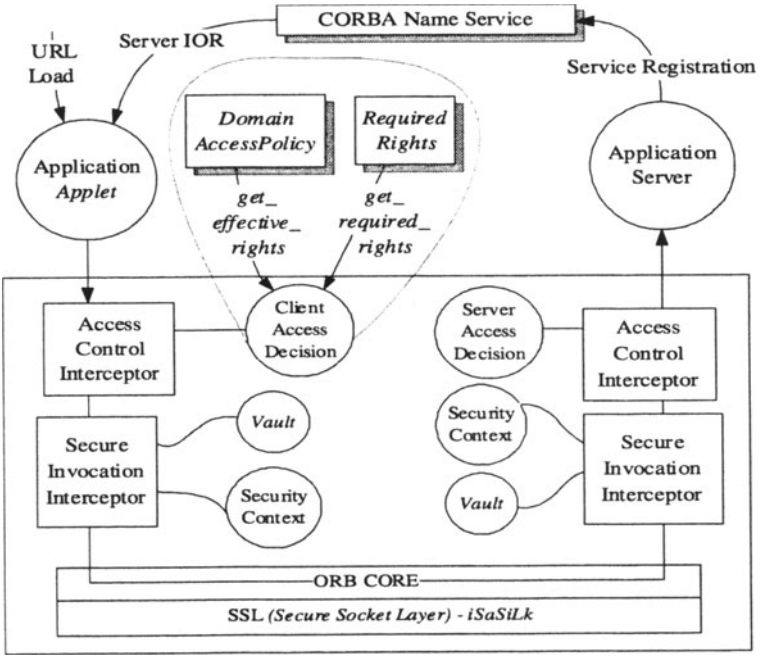

Figure 2. The Structure of Prototype Implemented.

An application example consisting of a bank system composed of a CORBA server object and a Java client *applet* was constructed to test the implementation of this prototype. The CORBA server object was developed with the tool JacORB 1.0 (www.jacorb.org), a *free* Java ORB, and the client *applet* was implemented with JDK 1.2.1. Netscape *browser* 4.5 was also used for the client and server interaction. The aim of this implementation was to outcome a discretionary policy based on CORBAsec structures [5].

This version of *PoliCap* was concerned with fully developing, first of all, the whole dynamic aspect of the CORBAsec service objects, which is no trivial task, inasmuch as the specification utilized is extremely broad [2]. For this reason, this initial version of *PoliCap* defines only the local objects *DomainAccessPolicy* and *RequiredRights* defined in a static form *in the binding time* that resides in the client machine.

Among the objects implemented in this prototype (besides the signed *applet* and application server) are the objects *ClientAccessDecision*,

ServerAccessDecision, *DomainAccessPolicy*, *RequiredRights*, *SecurityManager* and *PolicyCurrent*. These objects use other CORBA services, such as name service. The prototype is limited to a single name domain and the prototype credentials are created statically.

The implementation uses as deviation mechanisms the *interceptors* present in JacORB. In the prototype, the access control interceptor in the client machine invokes the object *ClientAccessDecision*, which is responsible for the validation of the access requests for the methods of server object, interacting with the local objects *DomainAccessPolicy* e *RequiredRights*. The method *access_allowed* of object *ClientAccessDecision* obtains the required rights invoking the method *get_required_rights* of the object *RequiredRights* and obtains the granted rights by the *DomainAccessPolicy* invoking the method *get_effective_rights*. It compares the required rights and the granted rights to the privilege attribute to decide whether or not the method to be invoked can be executed.

From this high level verification, the object *ClientAccessDecision* in cooperation with the access control interceptor (that has access to the *Request* CORBA structure) generates *capabilities*. Using API cryptographic methods of JDK 1.2, in the package IAIK-JCE, the values of *nonce* and *random* are generated. *The sender identifier* is obtained from the static credential. Once a *capability* has been formed, its fields are inserted in the *Request* CORBA. The object *ServerAccessDecision* verifies capabilities. In the prototype, the package iSaSiLk v.5.2 that implements the SSL v3 in Java was utilized (*http://jcewww.iaik.at/products/isasilk/*).

The implementation of objects defined in *PoliCap* will cause our model to take on its original format, as shown in *Figure 1*.

5. PROTOTYPE EVALUATION USING CC

The *Common Criteria* (*CC*) for the evaluation of security (*http://www.commoncriteria.org/*), standard ISO 15408, are single security evaluation criteria of information technologies *in distributed systems* [7].

Some important concepts of this criterion include *TOE*, PP and *ST*. The *TOE – Target of Evaluation* — is the part of the product or system that is subject to evaluation. In our case, the *TOE* to be considered is the prototype developed. The *protection profile* (*PP*) is a definition of sets of requisites and goals, regardless of the implementation, that allow the consumers and developers to create standardized sets of security requisites according to their needs. A registered *PP*, designated as *Controlled Access Protection Profile (CAPP)* [8], was utilized and accepted as standard in the evaluation

of this prototype. The *ST* can declare the conformity to one or more of the *PPs* and form *the basis for an evaluation.*

The *CC* provide seven levels of security guarantee of a *TOE*: EAL1 to EAL7 (EALs - *Evaluation Assurance Levels*).

There are two stages for the evaluation of a TOE: the ST evaluation and the corresponding TOE evaluation. To this end, an *ST* known as *JaCoWeb-ST*, was developed and its components are described in [5]. The evaluated *ST* operates in conformity with [8] reaching the level EAL3. *JaCoWeb* Security Group conducted an informal evaluation of the ST against the Security Target Evaluation class requirements presented in part 3 of the CC. Although a working draft, the ST was deemed to be in a reasonable state to allow the evaluation to proceed. NSA (*http://www.radium.ncsc.mil/tpep*) is responsible for formally evaluating the ST.

5.1 *JaCoWeb* Evaluation Report Results

The TOE evaluation was conducted by following the evaluator actions elements defined by the EAL3 requirements [9] using the evaluated *JaCoWeb-ST* [5] as the basis. As a result, a *Final Evaluation Report* was written in order to document the prototype evaluation. Performing each of the evaluator action elements, the *JaCoWeb Security* team, in conformity with all other requirements defined in [8], concluded that the prototype is considered a *TOE* level EAL3, meaning the prototype is methodically tested and checked and that it provides a moderate level of assurance.

Evaluation evidences performed during the *JaCoWeb* evaluation related to the tests results and to the vulnerability analysis, some of the key points of the CC evaluation methodology [10, 11], are described here and are also available at: *http://www.lrg.ufsc.br/~carla/FinalReport.html*.

5.1.1 Tests Results

The purpose of this activity is to determine whether the TOE behaves as specified in the design documentation and in accordance with the TOE security functional requirements specified in the ST.

In order to obtain tests results for the *JaCoWeb Security* prototype, a test plan was defined (*http://www.lrg.ufsc.br/~carla/TestPlan.html*). Two types of testing were performed: *unit* tests and *system* tests [12]. The units tested were the application *applet*, bank server, name server and service objects *AccessDecision*, *RequiredRights*, *DomainAccessPolicy* and *Current*.

Unit testing was based on branch coverage and the objective was to cover 95% of the prototype logic branches. Unit testing was applied for each CORBAsec service object listed above. An UML state diagram was made

for each one of them. A unit test report is available at *http://www.lrg.ufsc.br/~carla/UnitTesting.html*.

System testing used valid and invalid set of values, limit boundary values, special sets of values and cause-effect graphing to define *testcases*. System testing was applied for the application applet, the bank server and the CORBAsec service objects dealing with access control and secure invocation tasks. UML *usecases* diagrams were designed to describe application objects and UML class diagrams were used to describe CORBAsec service objects functionalities. A system test report is available at *http://www.lrg.ufsc.br/~carla/SystemTesting.html*.

5.1.2 Vulnerability Analysis

The purpose is to determine the existence and exploitability of flaws or weaknesses in the TOE [9]. This determination is based upon analysis performed by the developer and the evaluator, and is supported by evaluator testing [13]. Here we focus on the EAL3 *evaluation of vulnerability analysis*.

In order to find obvious vulnerabilities, *penetration tests* were used [14]. *NetXRay* tool was used for the penetration tests; which is a *basic sniffer*.

The configuration used consists of three computers Intel P133 32Mb of a local network, executing the *JaCoWebSecurity* package, the Netscape Communicator 4.5 browser, Microsoft Windows 95 and FreeBSD 2.2.8 operating systems and the Apache Web Server 1.3.9. The computer with 150.162.14.58 IP address executed the client application *applet*. The 150.162.14.41 IP address computer executed the CORBA name server and the 150.162.14.31 IP address computer run the bank server. Another machine executed *NetXRay* tool.

For the capture of the packages exchanged between client applet and name server, it is possible to observe in the *Figure 3*, in the fields of data, that the operation requested to the name server can be discovered *(resolve)*, and also, what the name of the server that will be accessed *(banco service)* is. That is possible because to establish the *first* communication with the name server, the information is exchanged in clear. The next communication exchanges use SSL for ciphering.

The *CosNaming* vulnerability could be solved if we use another name directory structure, as LDAP (*Lightweight Directory Access Protocol*). In this way, the interactions between clients and the LDAP server could use authentication methods before obtaining object references.

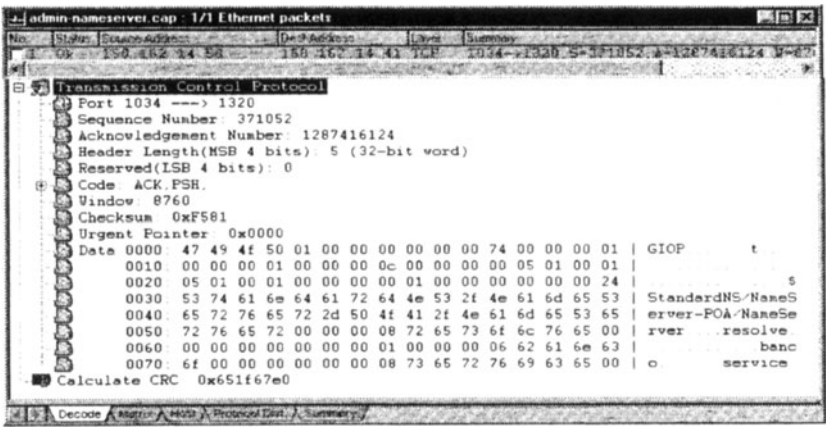

Figure 3. Data captured during Applet and CORBA Name Server communication.

6. CONCLUSIONS

The access control architecture of the project *Cherubim* [15] was developed using the concept of *capabilities*, which carry the access rights of the *principal* to the server machine where the authorization process takes place. The project does not utilize the CO*RBAsec COSS* objects, implementing its own policy objects instead.

Control [16] is an ORB that extends the *ORBAsec* with the *CORBAsec COSS* services that implement authentication services, secure transmission of messages and automatic access control. The discretionary policy is established with an access control language used for server objects. The interception of access control is executed on the server object side.

Comparing *PoliCap* with the experiences presented, we can verify that *PoliCap* is a service that performs access control *on the client side*, unlike the proposals described. *PoliCap* combines characteristics of *Cherubim*, such as that of *capabilities* and of *Control*, with the additional feature of restricted use of standardized objects.

PoliCap fills in an existing gap in security policy management in the CORBAsec, actualizing the first access control level of the project *JaCoWeb*. The second access control level is developed with the use of the *capabilities* proposed for the CORBAsec. *PoliCap* and the *capabilities* provide an important contribution to the managing of authorization policies in large-scale networks. The two access control levels reduce the network traffic in the case of denial of access, and at the same time the security of the system does not depend exclusively on the client's integrity. The capabilities

scheme proposed is a performance optimization when compared with traditional access control tasks performed on the server side only.

This prototype provided subsidies for the evaluation concerning standard ISO 15408, and we conclude that it meets the level EAL3. Some error situations were detected only during the definition of test cases.

Initially, the authorization scheme covers a single domain, but to make feasible its use in large-scale networks, the possible use of LDAP (*www.openldap.org*) is envisioned.

7. REFERENCES

[1] Bob Blakley, "The Emperor's Old Armor," In *Proc. of the ACM NSPW*, 1996, pp. 2-16.

[2] OMG, "Security Service:v1.5," *OMG Doc. Number 00-06-25,* June 2000.

[3] OMG, "Security Domain Membership Management Service," *orbos/01-07-20*, 2001.

[4] Bob Blakley, "CORBA Security: An Introduction to Safe Computing with Objects", *The Addison-Wesley OT Series*, 1999.

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

[6] Li Gong, "A Secure Identity-Based Capability Systems," *In Proc. of the 1989 IEEE Symp. on Security and Privacy*, pp. 56-63, Oakland, California, May 1989.

[7] ISO/IEC, "Common Criteria for Information Technology Security Evaluation," ISO/IEC 15408, December 1999 (http://www.commoncriteria.org/cc/cc.html).

[8] Information Security Systems Organization, "Controlled Access Protection Profile," NSA, Oct. 1999. (http://www.radium.ncsc.mil/tpep/library/protection_profiles/).

[9] CC Project, "Common Methodology for Information Technology Security Evaluation," *In Part 2: Evaluation Methodology*, August 1999.

[10] K. Jamer - CSE Canada, "Common Evaluation Methodology Special Topic: Testing," ppt slides. In: *ICCC First International Common Criteria*, May 2000, Baltimore, U.S.A. (http://niap.nist.gov/cc-scheme/iccc/trackd.html)

[11] J. Straw, "Common Evaluation Methodology Special Topic: Vulnerability Analysis," ppt slides. In: *Proceedings of the ICCC First International Common Criteria*, May 2000, Baltimore, Maryland, U.S.A (http://niap.nist.gov/cc-scheme/iccc/trackd.html).

[12] Pankaj Jalote, "An Integrated Approach to Software Engineering," *Springer-Verlag New York Inc.*, ISBN 3-540-97561-6, 1991.

[13] A. K. Ghosh et al., "An Automated Approach for Identifying Potential Vulnerabilities in Software," In: *Proc. of the IEEE Symp. on Security and Privacy*, 1998, pp. 104-114.

[14] T. J. Klevinsky, "Contemporary Hacking Tools and Their Use in Penetration Testing," Course. In: *FCSC99 - The Federal Computer Security Conference.* Course Day, May 1999, Baltimore, MD, U.S.A (http://www.sans.org/sf99/thursday.htm#thu-1).

[15] Campbell, Roy and Qian Tin, "Dynamic Agent-Based Security Architecture for Mobile Computers," *Proc. of the Second* PDCN '98, Australia, December 1998.

[16] Adiron Inc., "Control – Access Control for ORBAsec SL2 V 1.0 Alpha," *Adiron Center*, Syracuse University, Dec. 1999.