An Integrated Model for Access Control and Information Flow Requirements

Samiha Ayed, Nora Cuppens-Boulahia, and Frédéric Cuppens

ENST-Bretagne, Cesson Sevigne 35576, France

Abstract. Current information systems are more and more complex. They require more interactions between different components and users. So, ensuring system security must not be limited to using an access control model but also, it is primordial to deal with information flows in a system. Thus, an important function of a security policy is to enforce access to different system elements and supervise information flows simultaneously. Several works have been undertaken to join together models of access control and information flow. Unfortunately, beyond the fact that the reference model they use is BLP which is quite rigid, these research works suggest a non integrated models which do nothing but juxtapose access control and information flow controls or are based on a misuse of a mapping between MLS and RBAC models. In this paper, we suggest to formalize DTE model in order to use it as a solution for a flexible information flow control. Then, we integrate it into an unique access control model expressive enough to handle access and flow control security rules. The expressivity of the OrBAC model makes this integration possible and quite natural.

Keywords: DTE, OrBAC, MLS, RBAC, Security Policy.

1 Introduction

With diversity of possible attacks on an information system and with the different security properties that we try to ensure, maintaining and guaranteing system security is being more and more complex task. On the one hand, to protect a system, we must control all actions done from the beginning until the end of a user's sessions. So, a user must be authenticated and must be controlled when accessing objects. All actions that this user performs in the system have to be authorized. On the other hand, information systems present multiuser aspects and they manage interactions between different system parts. These interactions must be also supervised since they can lead to a misuse of system's objects or to a compromising of the system's integrity. To address these different issues, many models were proposed to satisfy different security requirements of a system [1]. Thus, there are models which are interested in integrity and confidentiality properties. Other models are interested in usage control and others in flow control. To secure a system, more than one model must be used since security requirements are as various as the diversity of these models. But there is no model that gathers these different security concerns. Workflow Management Systems are a very eloquent example of systems which need more than one security control model. Indeed, they not only present a diversity of objects and users but also various dependencies between different tasks and so between users. Access to the same object can be needed simultaneously, also writing on documents and modifying them must conform to the execution order. Moreover, the information flow has to be checked. All these system constraints have to be managed with a global security policy. This policy must deal with access and flow control requirements. Many works were done in order to converge access and information flow control. Several authors have discussed the relationship between RBAC and MLS lattice based systems [2–8, 10–12]. All these works are treated in details later. The basic idea that we retain is that the majority of them are founded on a mapping of RBAC notions and MLS notions. In these models, subject clearances are used as security levels to be assigned to roles in a role-based system. Beyond the limitations of MLS models, we consider that a such correspondence is a misunderstanding of MLS notions. We go in further details in the following section. Thus, the contribution of this paper is twofold. First, we go beyond MLS models as an example of flow control models. For this purpose, we base our study of flow control on a more flexible model which is able to permit us defining a flow control policy far from MLS constraints, say DTE (Domain Type Enforcement) model [15–17], we take close interest in this model, we explain our proper vision of it and finally we formalize it in order to use the formalism in defining our integrated model. Second, using DTE, we are leaded to the obligation of using contextual rules to express a security policy which manages information flows and object accesses. So, we propose to base our model on OrBAC model [18, 19] In OrBAC model, security rules are contextual. This characteristic of OrBAC security rules is necessary and sufficient to take into account information flows control. Moreover, OrBAC is able to express confinement aspect, a key concept in security management of complex systems. The entity *organization* defined in OrBAC model allows us to handle this aspect. Combining OrBAC access rules and a formally stated DTE, we present a different model to integrate access control and information flow requirements.

The remainder of this paper is organized as follows. Section 2 details related works. Section 3 provides different motivations of this work. Section 4 clarifies DTE model and defines our formalism. Section 5 introduces an overview of the OrBAC model and its components. Our integrated model for access and information flow control is presented in section 6. Finally, section 7 concludes the paper and outlines future work.

2 Related Works

Many works have addressed the issue of converging access control and information flow requirements. Nyanchama and Osborn have initially addressed the issue of combining RBAC and Bell-Lapadula (BLP) models in [2]. They examined the application of information flow analysis to role-based systems. Thus, [2] defines a flow policy which describes the authorized flows in the system. It classifies flows in different categories. Then, during process execution we must derive the set of flows generated. The two sets are compared to deduce and ensure that a given role-based scheme is consistent with the specified policy defined with basic flow axioms. Also, to determine this consistency, [2] uses graph theory to deal with the issue. It considers the set of roles and the role-based protection scheme and it draws a graph G1 to represent actual potential flows. A second graph G2 represents relations between flows defined in relation to the flow policy. It defines categories as nodes and edges as permissible information flows between categories. A role-based scheme is consistent with the system security policy if and only if the former graph is a subgraph of the latter. After this first tentative, Nyanchama and Osborn have tackled the issue with a different approach in [3]. In fact they introduced the notion of context. A context is viewed as the set of information accessed via a role. Using this concept, [3] proposes a realization of mandatory access control in role-based protection. In their formulation, role contexts are treated as the equivalent of security levels. They consider two concerns. The first is an acyclic information flow among role contexts. The second is equivalent rules to the simple security property and *-property of traditional multilevel security. [3] proposes a number of access constraints that would realize the equivalent of BLP rules. Finally, it concludes that in MAC, information flows must be acyclic. So the approach proposed ensures that information flows, caused either by role execution or user-role assignment, will be acyclic. In [5] Osborn was based on Nyanchama model described in [3]. She considers details of a single role or node and a given edge. Then, the model determines under what conditions such structures violate MAC constraints. [5] defines a more detailed structure. The new graph contains assignable roles. So it is very restricted compared to general role graphs. In other words, it is more interesting to analyze every role and every edge in a general role graph to verify if roles are assignable, and at what levels they are assignable. Sandhu was also interested in the issue but he goes in the other direction [4]. His approach represents another vision of simulating MAC controls in RBAC models. It is based on configuring RBAC components. It considers similarities between MLS security levels and RBAC roles. So a role is identified to a level of a login session. Its basic idea is to suppose two hierarchies in RBAC model, one for read and another for write. Thus, to each user we associate two roles, one for read (RR) and one for write (RW). Consequently, permissions are divided into groups of read and write privileges and so they must be assigned separately to RR and RW. [4] examines different variations of lattice based access controls (LBAC) and translates each of them into a role hierarchy. It defines a construction using a single pair of roles to accommodate lattices with different variations of the *-property. An extension of this work is presented in [7] and [10] considering different DAC variations. [7] focus on the importance of the administrative aspect. An implementation of these ideas can be found in [8]. In [6], Kuhn uses a construction of a role hierarchy. He defines an assignment of object categories to privilege sets and an assignment of categories to roles. So, to each role it assigns the categories associated with its privilege set and categories associated with privilege sets of its ancestors.

The first limitation of this work is related to category mapping which must be regenerated if changes are made in the role structure. The second is that the hierarchy created by the algorithm must be a tree, rather than a lattice hierarchy. Atluri, huang and bertino have used a convergence between RBAC and MLS to apply it to WorkFlow Management Systems. They define in [11] and [12] models of WFMS based in petri nets and they define an RBAC security policy. They associate levels to different objects used in the system and so to tasks using these objects. Then, they apply the MLS approach to their model taking into account different task dependencies. But their approach is localized into a workflow execution fully secure and partially correct. In other words, the approach does not enforce all task dependencies. So it can affect functional workflow execution.

Most approaches aiming at integrating access control and information flow requirements actually combines the RBAC and BLP models. The RBAC model is used to specify access control requirements by assigning users to roles and permissions to roles. A user is permitted to perform an access if he has activated one role this user is assigned to and if this access is permitted by the role. BLP is the first and mostly used information flow control model. It is based on the Multilevel security (MLS) policy and is used to prevent a high malicious subject from creating information flow to lower security levels. For this purpose, BLP defines two requirements: the simple security property (a subject is only permitted to read lower classified objects) and the *-property (a subject is only permitted to modify higher classified objects).

3 Motivation

Various previous works are essentially based on the same idea. They defined similarities between RBAC and MLS systems. The choice of RBAC is done to handle access control. Then, an application of MLS levels is done on system roles. They investigate clearance notion present in MLS systems and they apply it to roles. Thus, using a mapping between RBAC roles and MLS clearances they associate a clearance to each role. There is actually nothing wrong with identifying a role with a clearance level and assigning users to these roles. What is wrong in these approaches is to apply the BLP principles to the role behavior, especially the *-property. To illustrate our claim, let us consider an MLS application that manages classified objects and provides means to declassify or encrypt these objects. In the RBAC policy, one may consider that users assigned to the role *R_secret* (corresponding to the secret security level) are permitted to declassify and encrypt secret objects. Now, let us consider three different scenarios:

1. A user logs in the application at the secret level and attempts to declassify a secret object.
2. A user logs in the application at the secret level, creates a digest of a secret object and attempts to declassify this digest.
3. A user logs in the application at the secret level, has an access to a secret object using a browser and attemps to declassify this object through this browser.

We can consider that the first scenario is secure if both the login and declassification functions are trusted (i.e. they do not contain a Trojan Horse) and the secret object to be declassified has high integrity (i.e. this object has not been previously manipulated by a malicious application which hid some secret data the user did not want to declassify). This scenario actually corresponds to a robust declassification as defined in [9]. Regarding the second scenario, it is also secure if the application used to create the digest is a trusted function. Finally, regarding the third scenario, it is not secure if we consider that a browser is not a trusted application since this browser may call the declassification function to illegally declassify other objects. Notice that the conclusion would be the same if we replace declassification by encryption in the third scenario because a malicious browser could use the fact that an object is encrypted to create an illegal covert channel. Now if we apply the BLP principles to the role *R_secret*, then these three scenarios will be considered insecure since they all violate the *-property principle. This is clearly unsatisfactory. This is why we claim that it is incorrect to identify roles with clearance levels and then apply the BLP principles to these roles. Actually, the BLP principles apply to processes acting on behalf of users to prevent these processes from creating illegal information flow when they contain a Trojan Horse. By contrast, roles should define permissions of user, not of processes. Therefore, all previous works are based on BLP model to ensure flow control. This model present some weaknesses. Although it is able to protect a system from trojan horse used to access a document, BLP is unable to prevent a Trojan horse from altering or destroying data as it permits processes to write into files they can not read. BLP can not detect covert channels and remove them. As an MLS model, it is unable to be used to define policies outside multilevel security, which is not very used in practise because of it restrictions. To go away from these different constraints and drawbacks, we propose in this paper to use another model more flexible to control information flows, say DTE. DTE is presented in the following section. On another side, information flow are generally conditioned by program executions or generated after a process execution. For this reason, flow control must be contextual and not exclusively role dependent. Thus, to specify our integrated model, we choose using OrBAC model instead of RBAC model since it permits us defining contextual and dynamic rules. All the same, OrBAC allows us to express the confinement aspect as it defines explicitly the authority (organization) who defines and manages the security policy. So, our security policy is expressed using OrBAC contextual security rules and integrated DTE concepts. In this way, with the integrated model we propose, we ensure a fine grained access control and manage a more flexible information flow which is constrained by strictness of security levels.

4 DTE: Domain Type Enforcement

4.1 DTE Principles

In a system execution, processes dependencies and users interactions include data exchange. Information flows can be either explicit (as in assignment statements) or implicit (as in conditional statements) [13]. It can be due to functional dependency (e.g. x depends on y, hence there is a flow from y to x), or deductive (if knowing the value of x implies knowing the value of y, hence there is a flow from y to x) [14]. These data may have different sensitivity levels. So, if no security mechanism is applied on such exchange, the data transfer between processes may lead to a leak of confidential information and to a misuse of some documents or information. To limit the damage that it can be caused, confinement mechanisms have been developed [22, 23]. They are based on the idea of restraining the privilege access of subjects on objects. So, confinement is to restrict actions of programs in execution. BLP was the first model which was developed to address this issue and to deal with flow control. But it has some weaknesses we had already explained. Away from multilevel security domain, DTE model has been developed to satisfy security requirements of a system. Since there is no works which detail all DTE functionalities as they were specified, we propose in this section to clarify this technique and give our proper vision concerning its concepts. Domain and Type Enforcement (DTE) [15–17] is a technique originally proposed to protect the integrity of military computer systems and was intended to be used in conjunction with other access control techniques. As with many access control schemes, DTE views a system as a collection of active entities (subjects) and a collection of passive entities (objects) and groups them into domains and types. This classification not only reduces the size of the access control matrix but also simplifies the management. DTE defines two tables to base its access control definition. The first table is a global table called Domain Definition Table (DDT). It governs the access of domains to types (domain-type controls). Each row of the DDT represents a domain and each column represents a type. When a subject attempts to access an object, we must verify the entry corresponding to the domain of the subject and the type of the object in the DDT. If the access needed is defined in the matrix then the access is allowed, if not, the access is denied. The second table is called Domain Interaction Table (DIT). It governs the interaction between domains (Inter-domain controls). Each row and each column of the DIT represents a domain. The intersection cell denotes the access privilege that the domain corresponding to column possesses on the domain corresponding to row. To be stronger, DTE has defined the manner to be used to pass from a domain to another. So, if a subject S belongs to a domain D_1 then it wants to pass to a domain D_2, it must refer the DIT. The intersection of D_1 and D_2 in DIT should contain an entry indicating the activity or the program that S must perform to access to D_2. This entry is called the entry point. Thus, each domain has one or many entry points which consist in programs or activities to invoke by a subject in order to enter this domain. Any subject belonging to another domain must execute an entry point of the destination domain to be able to access this domain. When passing from a domain to another, a subject looses all its privileges of the source domain and gets a privileges set of destination domain. This notion of entry point makes the inter-domain communication more strict and precise. Although DTE model seems simple and enough strong it was not very used as a flow control model.

[16] presents a DTE integration into a μ-kernel. It suggests to centralize all access control decisions in user mode. The μ-kernel uses just a domain abstraction. Later, a work done in [17] extends the integration of DTE introducing both domains and types into kernel. [24–26] are concerned in using DTE in Unix and they present examples of DTE policies expressed in DTEL (DTE Language).

To more explain a DTE policy let us consider figure 1. It presents a DTE policy defined for Unix system. In this specification, *Types* declares one or more object types to be available to other parts of a DTEL specification. *Domains* declares different domains. Then, a domain specification is expressed as a list of tuples. It defines a restricted execution environment composed of four parts:

(1) "entry point" programs, identified by pathname, that a process must execute in order to enter the domain (e.g., (/bin/bash)), (2) access rights to types of objects (e.g., (rwxcd $\rightarrow$ root_t)), (3) access rights to subjects in other domains (e.g., (exec $\rightarrow$ root_d)). A DTEL domain controls a process's access to files, a process's access via signals to processes running in other domains, and a process's ability to create processes in other domains by executing their entry point programs. If a domain A has auto access rights to another domain B, a subject in A automatically creates a subject in B when it executes, via exec(), an entry point program of B, and (4) signals exchange between processes of source and destination domains. *Assign* associates a type with one or more files.

Policy of figure 1 shows how to protect a system from the wu-ftpd vulnerability to prevent an attacker from obtaining a root shell. This policy example will be reused in section 6 to clarify our integrated model.

All Works done around DTE use DTEL to specify the security policy. No reflection has been undertaken until now to formally define and use DTE model. A such formalism can be powerful enough to provide expressive security policies. In this paper, we propose to define a formalism allowing us to define merely a security policy which takes into account the flow control between system entities. Afterwards, this formalism must be blended with an access control model in order to deal with flow and access control simultaneously. This twofold control could make our security policy more useful and increases assurance that it is correctly specified. The premise of this formalism is that access control is based on domains and types. It provides facilities to express relationships between system entities. Thus we do not define a new model but an integrated one.

4.2 Our DTE Formalism

DTE has not been very much used since it has just inspired the design of some OS like SELinux. To use DTE as an approach to flow control, we propose to formalize the model. For this purpose let us introduce the following formal definitions.

Definition 1: *(domain) S is a set of all system subjects (active entities).* S *is divided into equivalence classes. Each class represents a domain* D *including a set of subjects having the same role in the system.*

Definition 2: *(type) O is a set of all system objects (passive entities).* O *is divided into equivalence classes. Each class represents a type* T *including a set of objects having the same integrity properties in the system.*

Definition 3: *(Entry Point) An entry point is a program or an activity which must be executed to pass from a domain D_1 to a domain D_2, denoted EP(D_1, D_2) or $EP_{1,2}$. An entry point implies two rules:*

- *subjects passing from D_1 to D_2 obtain a set of privileges depending on the entry point they execute,*
- *subjects passing from D_1 to D_2 loose all their D_1 privileges.*

The first rule means that the execution of an entry point defines the set of privileges that subjects will obtain when transiting from a domain D_1 to a domain D_2. These privileges are included into or equal to the set of privileges that D_2 subjects have. Each domain can have more than one entry point. The execution of these different entry points implies different privilege sets. The second rule means that if a subject leaves a domain it can not return to it only by executing one of its entry points.

```
♯ ftpd protection policy
types root_t login_t user_t spool_t binary_t lib_t passwd_t shadow_t dev_t config_t ftpd_t ftpd_xt w_t
domains root_d login_d user_d ftpd_d
default_d root_d
default_et root_t
default_ut root_t
default_rt root_t
spec_domain root_d (/bin/bash sbin/init /bin/su) (rwxcd→root_t rwxcd→spool_t rwcdx→user_t rwdc→ftpd_t
  rxd→lib_t rxd→binary_t rwxcd→passwd_t rxwcd→shadow_t rwxcd→dev_t rwxcd→config_t rwxcd→w_t)
  (auto→login_d auto→ftpd_d) (0→0)
spec_domain login_d (/bin/login /bin/login.dte) (rxd→root_t rwxcd→spool_t rxd→lib_t rxd→binary_t
  rwxcd→passwd_t rxwcd→shadow_t rwxcd→dev_t rxwd→config_t rwxcd→w_t) (exec→root_d exec→user_d)
  (14→0 17→0)
spec_domain user_d (/bin/bash /bin/tcsh) (rwxcd→user_t rwxd→root_t rwxcd→spool_t rxd→lib_t
  rxd→binary_t rwxcd→passwd_t rxwcd→shadow_t rwxcd→dev_t rxd→config_t rwxcd→w_t) (exec→root_d)
  (14→0 17→0)
spec_domain ftpd_d (/usr/sbin/in.ftpd) (rwcd→ftpd_t rd→user_t rd→root_t rxd→lib_t r→passwd_t
  r→shadow_t rwcd→dev_t rd→config_t rdx→ftpd_xt rwcd→w_t d→sppol_t) () (14→root_d 17→root_d)
assign -u /home user_t
assign -u /tmp spool_t
assign -u /var spool_t
assign -u /dev dev_t
assign -u /scratch user_t
assign -r /usr/src/linux user_t
assign -u /usr/sbin binary_t
assign -e /usr/sbin/in.ftpd ftpd_xt
assign -r /home/ftp/bin ftpd_xt
assign -e /var/run/ftp.pids-all ftpd_t
```

Fig. 1. Sample DTE policy file

If we suppose the following DDT and DIT:

	T_1	T_2	T_3
D_1	true	false	true
D_2	false	false	true
D_3	false	true	false

	D_1	D_2	D_3
D_1	–	$EP_{1,2}$	–
D_2	$EP_{2,1}$	–	$EP_{2,3}$
D_3	–	–	–

DDT entries, true and false, indicate if the domain has an access to different types or not. DIT entries define the entry points must be executed to transit from a domain to another. If we consider that a subject s belonging to the domain D_1 want to accede an object o belonging to T_2, it will refer to the DDT. s has no access to T_2, but consulting DDT and DIT it can find a manner to accede

T_2. In fact, D_3 has access to T_2 and s has an entry point allowing it to accede D_2. Also, the DIT present an entry point from D_2 to D_3. So, to accede o, s must execute $EP_{1,2}$ to pass to D2 then it must execute $EP_{2,3}$ to pass to D_3. Being in D_3, s obtain privileges allowing it acceding o since DDT contains an entry from D_3 to T_2. This path that s construct to accede o is called "confidence path". The following definition gives a formal definition of "confidence path".

Definition 4: *(confidence path) is a set of entry points* $< EP_{i,k}, EP_{k,l}, \ldots, EP_{m,j} >$ *which must be executed by a subject s to pass from* D_i *to* D_j *in order to obtain access to object to which it has not initially the access. Privileges granted through this confidence path are restricted to minimum privileges required to perform the access needed.*

The DTE formalism is based on two kinds of rules expressing and substituting DDT and DIT. These two rules are formally defined in the following.

Definition 5: *(SR_DDT) is a security rule substituting a DDT entry. It is defined as a 4-uplet : SR_DDT = (rule_type, domain, type, privilege) where Rule_type belongs to {permission, prohibition}.*

An instance of this rule can be: SR_DDT = (Permission, professor, exam, change) meaning that only professors have privileges to change exams.

So, we express a DDT as a set of rules defined according to definition 5.

Definition 6:*(SR_DIT) is a security rule substituting a DIT entry. It is defined as a 4-uplet: SR_DIT = (rule_type, domain1, domain2, entry point) where Rule_type belongs to {permission, prohibition}.*

An instance of this rule can be: SR_DIT = (Permission, engineering, http_d, /usr/bin/httpd) meaning that engineers have permission to pass from engineering domain to http_d domain by executing the program /usr/bin/httpd. When transiting to http_d, engineers obtain a set of http_d privileges defined by the execution of the entry point and they loose all privileges of source domain.

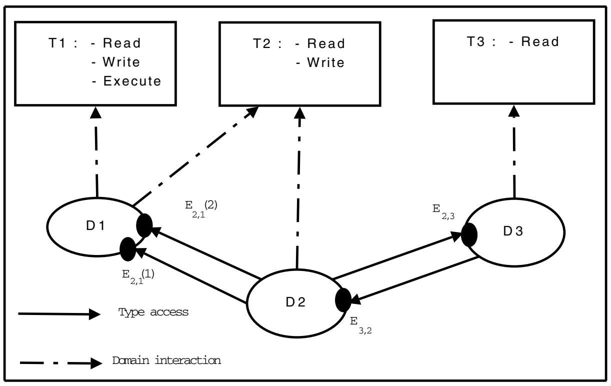

Fig. 2. Sample of a graphic system topology

As an example of a DTE policy, let us consider the figure 2. It presents a system consisting of three domains D_1, D_2 and D_3. These domains have different accesses to three object types T1, T2 and T3. Also, they have interactions between them defined through different entry points. We remark that D_1 has two entry points. In the system specification we can suppose that $EP_{2,1}(1)$ grants read and execute privileges to D_2 subjects and $EP_{2,1}(2)$ grants write privilege to D_2 subjects. Table 1 summarizes the DTE policy corresponding to figure 2.

Thus, we define our DTE control flow policy as a set of SR_DDT and SR_DIT rules. Such policy satisfies information flow requirements. Our integrated model uses this policy to deal with information flow control. We choose to base our model on OrBAC model as it is enough expressive and it allows us to integrate our flow control policy using access control rules. OrBAC offers *context* notion which permits us to define dynamic rules and to express our control flow policy using these rules. The sequel gives an overview of OrBAC and how we use it.

Table 1. DTE policy corresponding to figure 2

SR_DDT	SR_DIT
(Permission, D_1, T_1, Read)	(Permission, D_2, D_1, $E_{2,1}(1)$)
(Permission, D_1, T_1, Write)	(Permission, D_2, D_1, $E_{2,1}(2)$)
(Permission, D_1, T_1, Execute)	(Permission, D_2, D_3, $E_{2,3}$)
(Permission, D_1, T_2, Read)	(Permission, D_3, D_2, $E_{3,2}$)
(Permission, D_1, T_2, Write)	
(Permission, D_2, T_2, Read)	
(Permission, D_2, T_2, Write)	
(Permission, D_3, T_3, Read)	

5 OrBAC in Brief

In order to specify a security policy, the OrBAC model [18, 19] defines several entities and relations. It first introduces the concept of *organization* which is central in OrBAC. An organization is any active entity that is responsible for managing a security policy. Each organization can define its proper policy using OrBAC. Then, instead of modeling the policy by using the concrete implementation-related concepts of subject, action and object, the OrBAC model suggests reasoning with the roles that subjects, actions or objects are assigned to in an organization. The role of a subject is simply called a role as in the RBAC model. The role of an action is called activity and the role of an object is called view. Each organization can then define security rules which specify that some roles are permitted or prohibited to carry out some activities on some views. Particularly, an organization can be structured in many sub organizations, each one having its own policy. It is also possible to define a generic security policy in the root organization. Its sub organizations will inherit from its security policy. Also, they can add or delete some rules and so, define their proper policy.

The definition of an organization and the hierarchy of its sub organizations facilitate the administration [20]. The security rules do not apply statically but their activation may depend on contextual conditions [21]. For this purpose, the concept of *context* is explicitly included in OrBAC. Contexts are used to express different types of extra conditions or constraints that control activation of rules expressed in the access control policy. So, using formalism based on first order logic, security rules are modeled using a 6-places predicate.

Definition 7: *an OrBAC security rule is defined as: security_rule (type, organization, role, activity, view, context) where type* $\in$ *{permission, prohibition, obligation}.* An example of this security rule can be: security_rule (permission, a_hosp, nurse, consult, medical_record, urgency) meaning that, in organization a_hosp, a nurse is permitted to consult a medical record in the context of urgency.

6 Access and Information Flow Control Convergence

In this section, we present our integrated model for access control and information flow requirements. The model is based on OrBAC to express access control and it is enriched with a DTE approach to express information flow control. This convergence let us considering only one model (our proposed model) to specify security policy of a system. Finally, we exemplify the model.

6.1 Access Control Policy

OrBAC defines contextual rules which can depend on different contexts. These contexts can be related to conditions or circumstances under which a rule is valid or an activity is performed. If we observe SR_DDT and OrBAC rules we can deduce that the former are expressed by the latter using a default context. The default context is expressed in OrBAC as a context which is always true. So rules with such context are always valid. In fact, role, activity and view significance in OrBAC match respectively domain, privilege and type significance in DTE. So, an SR_DDT rule can be easily expressed using an OrBAC rule. This is a quite natural result as SR_DDT rules are specific access control rules. Further, OrBAC gives the possibility to define more fine access control rules than DDT in DTE can do since it presents different types of contexts. Indeed, with different OrBAC contexts we can express diverse conditions and temporal constraints. Thus, we can formulate dynamic and expressive rules which can not be expressed just by using DTE. That is why we have not choose DTE formalism to express both access an flow controls, otherwise we get a static integrated model with respect to access control aspects.

6.2 Information Flow Control Policy

As we have presented, OrBAC is an efficient model to express access control rules. So, it will be very useful if we succeed to express access and flow control using the same model. Our DTE formalism offers SR_DIT rules to define information flow control policy. But as we aforementioned, DTE is not very efficient in expressing access control, whereas OrBAC is. In the sequel, we present our approach to define our integrated model based on OrBAC and using a DTE approach. We suppose, due to space limitation, that this policy is closed meaning that all which is not permitted is denied and we do not deal with *obligations*.

Based on an OrBAC rule, SR_DIT [*SR_DIT = (rule_type, domain1, domain2, entry point)*] can be seen as a particular OrBAC rule. This rule must express the transition between different domains and must introduce entry point notion in order to preserve a secure information flows and to keep DTE aspects. Therefore, to consider this particular rule we suppose the following hypotheses: (1) the source domain is considered as the role in the OrBAC rule (we have already said that the two notions have equivalent significances), (2) the destination domain is considered as the view in OrBAC rule, (3) the transition between two domains can be expressed as an OrBAC activity since the basic role of this specific rule is to handle interactions between domains. So, we define the OrBAC activity as an `Enter` activity, (4) the entry point defines the manner to enter a domain. Thus, we can consider it as a condition of rule validation. Therefore, an entry point can be defined as a specific context in an OrBAC rule denoted `through(`$E_{i,j}$`)`. This context specifies that the rule is valid only through $E_{i,j}$ execution.

Thus, such a rule is expressed, in a specific organization org and handling transition between D_1 and D_2, as follows: *SR (permission, org,* D_1*, `Enter`,* D_2*, `through(`*$E_{1,2}$*`))`.* These flow control rules can be enriched with other OrBAC contexts. Indeed, `through(`$E_{i,j}$`)` context can be used in conjunction with different OrBAC contexts, for example temporal contexts, to express more restrictive or conditioned flow control. Also, We recall that a transition from a domain D_1 to a domain D_2 is possible only if there is the corresponding rule in the policy. In other words, handling domains interactions does not contain prohibitions. This interaction is allowed only if there is a corresponding permission. Such transitions between domains correspond, in our integrated formalism "access control/flow control", to a context change. Since transiting to another domain corresponds to the activation of a new context, new rules will be activated. These rules are those for which this context is valid. This dynamic management of the security policy and the closed policy hypothesis guarantee the loose of source domain privileges during transition. New granted privileges are defined according to the entry point executed. The entity *organization* is useful to control the flow in the inter-organizational environment. This will be developed in a forthcoming paper.

6.3 Example

To exemplify our proposed model, let us reconsider figure 1. Using our integrated model, line 9 is expressed as the following rules set. We suppose that we are in Unix system organization.

1. (permission, Unix, login_d, rxd, root_t ,*default*)
2. (permission, Unix, login_d, rwxcd, spool_t ,*default*)
3. (permission, Unix, login_d, rxd, lib_t ,*default*)

4. (permission, Unix, login_d, rxd, binary_t ,*default*)
5. (permission, Unix, login_d, rwxcd, passwd_t ,*default*)
6. (permission, Unix, login_d, rxwcd, shadow_t ,*default*)
7. (permission, Unix, login_d, rwxcd, dev_t ,*default*)
8. (permission, Unix, login_d, rxwd, config_t ,*default*)
9. (permission, Unix, login_d, rwxcd, w_t ,*default*)
10. (permission, Unix, login_d, *Enter*, root_d, *through(exec())*)
11. (permission, Unix, login_d, *Enter*, user_d, *through(exec())*)

The rules 1-9 express the access control policy. Rules 10 and 11 handle interactions between login_d and root_d, user_d. The whole set of these rules presents the security policy corresponding to line 9 of the first example in figure 1 of the section 4.1. This policy is based on OrBAC rules enriched with DTE approach. It is expressed using only one form of rules. If we consider for instance the first rule, "rxd" indicates the activity allowed by this rule. We use this notation just to simplify and reduce the list of security rules. In fact, "rxd" corresponds to three privilege activities: read (r), execute (x) and destroy (d). Thus, this rule corresponds to three rules in the policy. For the two last rules, the activity field contains *"Enter"* which specifies permitted transitions between domains since they are flow control rules. In this example we use the *"default"* context when expressing access control rules. This context implies no conditions on the activity performance. Such choice is done to simplify the example. Other contexts, such temporal contexts, can be used in conjunction to *"default"* context or to *"through"* context. *"through"* context is used in flow control rules to express the manner to enter corresponding domains. Transiting into root_d and user_d domains is released when executing exec() which activates one of the entry points defined for root_d and user_d respectively: (/bin/bash, /sbin/init, /bin/su) or (/bin/bash, /bin/tcsh). These entry points define privileges granted to login_d subjects migrating to root_d or user_d. Different domains used here are specified corresponding to our specification done in definition 1 of section 4.2. We remark that in this example login_d is used as a role in access control rules and root_d and user_d are used as views in information flow control rules. But, they can be used as domains in other access control rules. According to this policy, login_d has no access to the type user_t. If a login_d's subject requires an access to objects of this type, he must transit to root_d or user_d which have access to this type (see figure 1, lines 8 and 10). So, our integrated model ensure the aspect of *"confidence path"* defined in DTE formalism (see definition 4 of section 4.2).

7 Conclusion

In this paper, we have presented an integrated security model that is capable of taking into account information flow and access control. The security policy is based on OrBAC rules which integrate flow control using a DTE approach. OrBAC is an adequate choice since it permits us to define contextual and dynamic rules. Also, it is enough expressive to be able to integrate information flow control. The organization notion present in OrBAC allows to express confinement. Our approach remedies to weaknesses present in previous approaches based on MLS and RBAC models. In this paper, we have considered information flow control into the same organization. As part of future work, we will consider a more complex case where we supervise inter-organization flows. Indeed, organizations must exchange flows to have knowledge of what is happening globally in the system. These flows have to be managed in order to keep a secure execution environment of processes. Also, we intend to apply our integrated model to Workflow Management Systems (WFMS). This is a critical issue since such systems present a very important need of security. In fact, they present multi users aspects, inter-dependent execution and dynamic progression. The confinement aspect that the organization entity express will be very useful to define the inter-organization policy.

Acknowledgment

This work is partially supported by the RNRT project Polux.

References

1. Sandhu, R.S.: Lattice-Based Access Control Models. IEEE Computer 26(11), 9–19 (1993)
2. Nyanchama, M., Osborn, S.: Information Flow Analysis in Role-Based Security Systems. In: Proc. ICCI 1994. International Conference on Computing and Information, pp. 1368–1384 (1994)
3. Nyanchama, M., Osborn, S.: Modeling Mandatory Access Control in Role-Based Security Systems. In: IFIP Workshop on Database Security (1996)
4. Sandhu, R.: Role Hierarchies and Constraints for Lattice-Based Access Controls. In: Proc. Fourth European Symposium on Research in Computer Security, Rome, Italy (1996)
5. Osborn, S.: Mandatory Access Control and Role-Based Access Control Revisited. In: Proceedings of the second ACM workshop on Role-based access control, Fairfax, Virginia, United States, pp. 31–40 (1997)
6. Kuhn, D.R.: Role Based Access control on MLS Systems without Kernel changes. In: Proceedings of the third ACM Workshop on Role-Based Access Control, Fairfax, Virginia, United States, pp. 25–32 (1998)
7. Osborn, S., Sandhu, R., Munawer, Q.: Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access control Policies. ACM Transactions on Information and System Security 3(2), 85–106 (2000)
8. Demurjian, S.: Implementation of Mandatory Access control in Role-Based Security System. CSE367 Final Project report (2001)
9. Myers, A.C., Sabelfeld, A., Zdancewic, S.: Enforcing robust declassification. In: Proc. IEEE Computer Security Foundations Workshop, pp. 172–186 (June 2004)
10. Sandhu, R., Munawer, Q.: How to do discretionary access control using roles. In: Proc. of the 3rd ACM Workshop on Role Based Access Control (RBAC 1998), Fairfax, VA, USA (1998)

11. Atluri, V., Huang, W.-K.: Enforcing Mandatory and Discretionary security in Workflow Management Systems. Journal of Computer Security 5(4), 303–339 (1997)
12. Atluri, V., Huang, W.-K., Bertino, E.: A semantic Based Execution Model for Multilevel Secure Workflows. Journal of Computer Security 8(1) (2000)
13. Liu, L.: On secure Flow Analysis in Computer systems. In: Proc. IEEE Symposium on Research in Security and Privacy, pp. 22–33 (1980)
14. Millen, J.K.: Information Flow Analysis of Formal Specifications. In: Proc. IEEE Symposium on Research in Security and Privacy, pp. 3–8 (1981)
15. Badger, L., Sterne, D.F., Sherman, D.L., Walker, K.M., Haghighat, S.A.: Practical Domain and Type Enforcement for Unix. In: IEEE Symposium on Security and Privacy, Oakland, CA, USA (1995)
16. Tidswell, J., Potter, J.: Domain and Type Enforcement in a μ-Kemel. In: Proceedings of the 20th Australasian Computer Science Conference, Sydney, Australia (1997)
17. Kiszka, J., Wagner, B.: Domain and Type Enforcement for Real-Time Operating Systems. In: Proceedings ETFA 2003, Emerging Technologies and Factory Automation (2003)
18. Abou El Kalam, A., El Baida, R., Balbiani, P., Benferhat, S., Cuppens, F., Deswarte, Y., Miége, A., Saurel, C., Trouessin, G.: Organization Based Access Control. In: IEEE 4th International Workshop on Policies for Distributed Systems and Networks, Lake Come, Italy (2003)
19. Cuppens, F., Cuppens-Boulahia, N., Sans, T., Miége, A.: A formal approach to specify and deploy a network security policy. In: Second Workshop on Formal Aspects in Security and Trust (FAST), Toulouse, France (2004)
20. Cuppens, F., Cuppens-Boulahia, N., Miége, A.: Inheritance hierarchies in the Or-BAC model and application in a network environment. In: Second Foundations of Computer Security Workshop (FCS 2004), Turku, Finlande (2004)
21. Cuppens, F., Miége, A.: Modelling contexts in the Or-BAC model. In: 19th Annual Computer Security Applications Conference, Las Vegas (2003)
22. Boebert, W.E., Kain, R.Y.: A further Note on the Confinment Problem. In: Proceedings of the IEEE 1996 International Carnahan Conference on Security Technology, IEEE Computer Society, New York (1996)
23. Boebert, W.E., Kain, R.Y., Young, W.D.: The extended Access Matrix Model of Computer Security. ACM Sigsoft Software Engineering Notes 10(4) (1985)
24. Hallyn, S., Kearns, P.: Tools to Administer Domain and Type Enforcement. LISA XV. San Diego, CA (2001)
25. Oostendorp, K.A., Badger, L., Vance, C.D., Morrison, W.G., Petkac, M.J., Sherman, D.L., Sterne, D.F.: Domain and Type Enforcement Firewalls. In: Proceedings of the Thirteenth Annual Computer Security Applications Conference, San Diego, California, pp. 122–132 (1997)
26. Walker, K.M., Sterne, D.F., Lee Badger, M., Petkac, M.J., Shermann, D.L., Oostendorp, K.A.: Confining Root Programs with Domain and Type Enforcement (DTE). In: Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography, San Jose, California, vol. 6 (1996)