On the Increasing Importance of Constraints

Trent Jaeger
IBM T. J. Watson Research Center
Hawthorne, NY 10532
jaegert@watson.ibm.com

Abstract

In this paper, we examine how the addition of role-based access control (RBAC) model features affect the complexity of the RBAC constraint models. Constraints are used in RBAC models to constrain the assignment of permissions and principals to roles (among other things). Historically, it was assumed that the role assignments would change rather infrequently, so only a few constraints were necessary. Given new RBAC features, such as context-sensitive roles, the complexity of the restrictions that can be required is increasing because the role definitions may depend on application state. As application state changes, so do the role assignments. We examine the RBAC constraint problem using an example of a virtual university. We propose RBAC model features for simplifying the representation of constraints given our experience with this example.

1 Introduction

In this paper, we identify the increasing importance of constraints in role-based access control (RBAC) models and examine the features of constraint models that can be brought to bear to handle this complexity. Constraints ensure that the role specifications that have been made actually enforce the access control requirements. A typical RBAC model consists of roles to which principals and permissions may be assigned [12]. The assignment of principals and permissions to roles is limited by constraints (e.g., user-role assignment and permission-role assignment constraints [10]). While the assignments limit the actions that can be performed, the constraints are the final line of defense. Any specified action can be taken as long as it does not violate some constraint.

In fact, the demand on constraints is becoming greater as new features are being added to RBAC models. Consider the addition of parameterized roles to RBAC models [4, 8]. In a parameterized role, the role and its permissions are indexed by parameters, so the same role definition may apply to multiple contexts. Of course, each of these contexts may involve constraints, and it is not true in general that the constraints can simply be parameterized as well. It is possible that some embodiments of the roles may be more or less constrained than others. For example, one role may allow certain objects to be part of the role's permissions that another would not allow. Thus, a means to control the complexity of constraints will be necessary.

Our investigation of constraints vacillates between two goals: (1) trying to enable proofs of various safety properties (i.e., can a principal obtain a particular permission) so that administrators can make statements about the effectiveness of the roles they have defined and (2) trying to define RBAC model primitives to simplify these proofs. Historically, constraints in RBAC models are repre-

RBAC '99 10/99 Fairfax, VA, USA

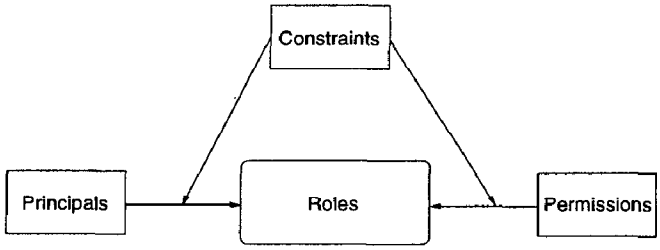

Figure 1: **Fundamental RBAC Concepts:** Roles associate the principals that can assume them with the permissions assigned to the role. Constraints may limit either the assignment of principals to roles or the assignment of permissions to roles.

sented as general constraint predicates. While this enables flexibility in constraint specification, consistent constraint specification using predicates is hard. Constraints are not fail-safe in general. Our hope is to simplify the safety problem through some careful decomposition where the compositions can be in the form of the union of permission sets. Due to tenuous nature of composing security properties, we ask more questions than we answer at present.

The remainder of the paper is structured as follows. In Section 2, we aim to demonstrate the importance of constraints to the development of an effective RBAC model specification. While Sandhu *et al* [12] have stated that constraints are "sometimes argued to be the principal motivation for RBAC," it was the functionality of constraints that they motivated. Here, we are also interested in the management of constraints themselves and the safety of a system controlled by constraints. In Section 4 we describe the constraint requirements of an example problem, a virtual university. In Section 3, we examine related work in the definition of constraint models for RBAC. In Section 5, we describe how the constraint requirements of our example could be satisfied. In Section 6, we try to generalize the approach specified in Section 5 while still enabling administrators to easily determine the permissions that are available to a principals and those that could be obtained by a principal. In Section 7, we conclude and state future work.

2 Security Constraints

The basic RBAC model consists of principals, permissions, roles, and constraints [10] as shown in Figure 1. Principals obtain access rights by being assigned to roles or having permissions added to the roles they belong to. The union of the permissions of a principal's roles describe its authorized access rights. In general, the access rights of a system need not be static, so it may be necessary to add principals to new roles and add permissions to existing roles. However, such additions must not enable a principal to obtain the access rights necessary to obtain unauthorized access rights. For example, if two principals are in a separation of duty, but an access right is added to a role that enables one principal to read data written by the other, then the that addition provides an unauthorized access right. That is, the addition of such a right would violate the access control policy associated with these two principals.

Constraints express restrictions, such as principal-role and permission-role assignment restrictions, to prevent such policy violations [1]. In general, if a principal can obtain a permission that would violate the system's access control policy, then the access control specification is said to be *unsafe*. In an RBAC model, it is the job of the constraints to prevent administrators from assigning principals and/or permissions to roles such that a principal's permissions may become unsafe. Toward this end, when any permission or principal is added to a role, the relevant constraints must be checked to determine if the addition is unsafe. In the case above, a separation of duty constraint between the two principals would prevent the addition of a right that enables the principals to communicate.

Historically, RBAC models were applied to environments in which assignments changed infrequently and there were few roles, so few constraints were necessary. However, advances in the development of RBAC model concepts are increasing the demand on constraints. First, the concept of a context-sensitive (or parameterized) role [4, 8] enables roles to depend on application state. The applicability of such an RBAC model has been demonstrated for database, medical, and general

[1] We will ignore other types of constraints, such as cardinality and activation, for the moment.

collaborative applications [7]. The main demand added to the constraint model is that since roles are indexed by application state, each time the application state changes a change in permissions may result. Since permissions can be controlled at a finer granularity, more constraints may be necessary.

Second, since parameterized roles enable fine-grained, the level of abstraction between constraints can vary, thus making analysis difficult. Instead of basic separation of duty, constraints may now focus on separation of a specific subset of the principal's access rights. Therefore, if there are two specifications for assigning permissions, the constraints under which these specifications are restricted must be consistent and complete.

Lastly, we examine the relationship between RBAC constraints and the *safety problem* [5]. The safety problem is to find an (reasonably efficient) algorithm to determine whether a particular right can be obtained given an initial protection state and operations for changing that state. Harrison *et al.* show that the safety problem is undecidable for general access control models. However, others have identified access control models in which safety is both decidable and reasonably efficient [14, 9]. In the RBAC approach, it is the conditions of the constraint that determine whether the addition is acceptable or not. In general, such conditions may attempt to verify the safety of the resulting protection state. No public statement on the decidability of safety in general RBAC models has been made, but given the complexity of the general model we expect safety to be undecidable.

Thus, given the importance of constraints to the RBAC model and the complexity of verifying that a set of constraints are sufficient to ensure the safety of the system, we conclude that, like previous work in access control models, simplifications of the general RBAC constraint model will be necessary to enable verification that the system is safe. However, it is unclear if this can be done in a way that preserves enough of the flexibility of the present model. An alternative is to simplify the use of constraints to enable the verification of safety for each particular system (e.g., because the number of constraints is small). In this paper, we investigate such simplifications.

3 Related Work

Constraints have been envisioned since some of the earliest RBAC models were proposed [15, 13, 11]. In these models, the concept of constraints was identified to restrict the assignment of users and permissions to roles in order to enforce common security policies, such as separation of duty. However, constraint models themselves were not included in these RBAC models.

The earliest RBAC constraint model that we have identified is that of Chen and Sandhu [2]. The constraint model consists of a number of functions for extracting role state and a language for expressing constraints using that role state. The language enables expressions of constraint predicates that may be tested as invariants or as preconditions to changes in role state. The functions and language primitives are policy-independent, so they may be applicable to a wide range of scenarios.

Later, Bertino *et al* defined a constraint model for workflow systems [1]. The difference between this model and that of Chen and Sandhu is mainly that the state functions act on the workflow tasks and role state, not just the state of the role system. The constraint language itself is again based on the expression of constraint predicates.

Constraint predicates are a general approach to expressing constraints, but as described in the previous section, even a system with only a few constraint predicates can be difficult to evaluate for safety. Thus, to date, practical application of constraint predicates has generally been limited to static and dynamic separation of duty constraints [3].

A goal is to find constraint representations that enable simpler analysis of safety. As has been typical of RBAC model development, researchers have re-applied RBAC concepts to solve new RBAC problems. An important example is the creation of administrative roles [10]. An administrative role is a kind of role that describes the ability of the principals that belong to that administrative role to administer regular roles. An administrative role describes the principal and permission assignments that administrators can make to the roles they can administer and the conditions under which those assignments can be made. Thus, administrative roles constrain the role assignments that can be made.

Another example of administrative roles are *transform limits* [7]. Each regular role is associated with a set of administrators for that role and the rights that those administrators can manage. The advantage of basic transform limits are that it is easy to compute various safety properties, such as the maximal set of rights a principal could obtain. Parameterized roles may be used for transform limits, so it is envisioned that some evolution may occur. Similarly to administrative roles, constraints restrict the evolution of transform limits. A problem with this approach is that one transform limit may preclude, by a constraint, the change in another transform limit even though the rights that embody the conflict have not been assigned yet. This is a trade-off between having complex safety analysis (administrative roles) and complex conflict resolution of safety constraints (transform limits). Since fail-safety is often a goal of secure systems, some form of conflict resolution may not be unreasonable, but the trade-off is not clear-cut.

4 Example

We now examine the requirements on constraints in a specific RBAC example. We have been investigating the application of RBAC to a virtual university setting [6]. The advantages of RBAC in this domain are that: (1) it enables the rights of principals to depend on the courses in which they participate and their role in these courses and (2) it enables a single instructor role and student role to be defined that confers the default permissions for any course. Using context-sensitive roles [4, 8], a student role and an instructor role can be defined whose rights depend on the course to which they are associated. All the default rights of students and instructors in a course can be defined using these roles. Thus, it is not necessary for instructors or students to administer the default rights of any course. However, a significant amount of ad hoc delegation is possible, so principal-role and permission-role assignment constraints are necessary to control these delegations. However, verifying that the specified constraints actually ensure the application's security policy is enforced is a concern.

In the virtual university application, all course information are represented electronically as course objects. The default operations in a course are as follows. First, instructors create activities [2], including homework and examinations and assign them to the course students. Students must only work on their own activities, although it is possible that a group of students may collaborate. The course students respond to the activities which are then graded by the course instructors.

The default security requirements of this application are as follows:

- **Instructors**:
  - Create (i.e., read and write) student activities
  - Create supporting materials for activities
  - Assign students in their courses to activities
  - Read committed student responses
  - Create grades for committed student responses
- **Students**:
  - Read assigned activities
  - Create responses to activities, either individually or in groups
  - Commit responses (which then become read only)
  - Read committed graded responses

Instructors create and assign student activities to groups of one or more students in the courses they teach. This results in the individual student groups obtaining a new response object in which they may respond to the activity. Students may only access a response for the student group to which they are assigned. Once a response is completed, the student group creates a committed response, and they may no longer modify it. Instructors may not modify the committed response either. Instructors create a graded response which they then commit before the students can read it.

However, courses are not quite as well-structured as we have portrayed in the default case.

[2] We use the term *activity* to refer to the descriptions of the tasks that instructors assign to students. We initially called them *assignments*, but since we also discuss the assignment of users and permissions to roles that term became ambiguous.

For example, a student may wish that an instructor review an uncommitted response. This requires that the student give the instructor read access to an uncommitted response. However, this is generally not considered to be dangerous because nature of the instructors' job requires that they help the student. This may even be made into a default right where it is assumed that the instructor will only use this privilege when requested by a student.

A problem would occur if an instructor can make an access control decision that could lead to an *unsafe* situation (i.e., a particular right may be leaked to a particular subject who should not obtain that right). For example, an instructor may choose to have the students devise different activities for the course. However, giving students write access to activities may lead to problems, such as other activities changing after their assignment (due to errors in administration).

Using the RBAC approach, we would hope to identify fundamental constraints that any principal-role and permission-role assignment must obey. For the system to work properly the constraints must meet two requirements: (1) the constraints are sufficient to enumerate all the necessary restrictions in order for the system to be safe and (2) the algorithms that evaluate the conditions of these constraints must be reasonably efficient (i.e., polynomial).

The informal statement of the fundamental constraints in this example are given below.

- **Permission-role constraints**
  - A student may only obtain write access to an object if an instructor does not have write access and vice versa
  - A response cannot be read by principals other than the students that submitted it and the instructor until a grade has been given to it
- **Principal-role constraints**
  - A student may not belong to multiple groups for a single activity
  - A principal cannot be both a student and an instructor of the same course
  - All students in a student group of a course must be students in the course

Fortunately, it appears that the conditions of these constraints can be computed in polynomial time. However, it is not clear that the specified constraints are sufficient to ensure a safe system, even in such a simple example. Unlike fail-safety in security, where only positive rights are specified so no rights are gained tacitly, it is not clear that a positive specification of constraints will ensure the right policy is enforced. For example, if the first constraint is stated such that a write permission is only granted to a student if the instructor does not possess a write permission to the same object, this is a necessary, but perhaps not sufficient, condition for expressing the full policy. That is, perhaps other conditions must also be tested before the permission is granted.

In particular, we can see that the default requirement that students cannot read activities until they are assigned is not captured in the fundamental constraints listed above. Of course, this constraint can be added, but clearly mixing constraints of different levels of abstraction makes it difficult to determine the overall safety of the system. It is difficult to tell if the combination of constraints always yields roles that obey the intended access policy. Specification of constraints is often complex, and forgetting a constraint can result in an error.

5 Constraint Design

In this section, we examine the expression of the security requirements of the VITC example. In particular, we look at three things: (1) how the basic rights of the instructors and students are established and maintained consistently with the course status; (2) how the instructors may manage the assignment of students to student groups to perform activities jointly; and (3) how exceptions to the basic rights are created and controlled. In the next section, we try to make some generalizations for constraint model design based on the experience of this example.

The RBAC model uses parameterized roles and controls role assignments using transform limits. Transform limits specify the administrators and a set of permission or principal specifications which define the permissions or principals that the administrator may assign to the role. Principals are

defined in terms of a set of principals. Permissions are defined in terms of a set of permissions. In addition, a single permission may refer to a set of objects if the target object of the permission is an object group. Roles are parameterized by parameterizing the principals and permissions in the role. Therefore, changes in transform limits are largely a result of changes in application state.

5.1 Default Course Permissions

We first define the default course permissions of the instructors and students. By default, we simply mean that we expect that all courses will use the same permission specification, so the permission definitions will apply to each course. Parameterized roles enable a single permission specification to apply to multiple courses because the object groups can be indexed by course.

The architecture of the VITC system consists of the following principals: a security server, a VITC server, and VITC clients. The security server maintains the permissions of the system's principals (i.e., it can support other applications besides the VITC). The VITC server administers the rights of the VITC clients. This job entails defining the roles available to VITC clients (e.g., instructors and students) and keeping the permissions of these roles consistent with the state of the VITC application. The VITC clients are students and instructors.

Next, we define the useful object groups for defining role permissions in Figure 2. The main groups of objects are activities, responses, and grades. Each of these objects goes through a phase of development where only the people creating them should have access, followed by a phase of availability where others can read the objects. In the case of responses, when the are committed (i.e., made available), they become immutable. Figure 2 shows the object groups and where transitions between groups occur. For example, a student creates a response object which is placed in the interim response group initially. Upon commitment, a new response object is created in the committed response object group.

We now examine the permission-role assignment policy for the VITC clients. Since the VITC application is the sole administrator in this example, it is the only principal that can define the membership of these object groups. However, multiple administrative principals are possible in general (see Section 5.2). The permission-role assignment for a role is defined by the permission transform limits for the role. The course instructor's permission transform limits are specified as follows [3]:

- **Role:** instructors (course)
- **Administrator:** VITC
- **Permission Transform Limits:**
  - {VITC, VITC obj, rw, activities(course)}
  - {VITC, VITC obj, rw, supplementals(course)}
  - {VITC, VITC obj, rw, grades(course)}
  - {VITC, VITC obj, r, committed responses(course)}

Thus, for a particular course, the instructors can use the VITC to create activities, supplemental materials, and grade sheets for student responses. Instructors cannot read the responses as they are being developed.

Note that the addition of a new grade object does not change the transform limits. The object group can specify all objects in a certain location in the name space, so simply creating the object makes it available without changing the transform limit. Only the VITC can create objects in these name space locations (not shown in the specification).

The role definition of course students is specified as follows:

- **Role:** students (course)
- **Administrator:** VITC
- **Permission Transform Limits:**
  - {VITC, VITC obj, r, public activities(course)}
  - {VITC, VITC obj, r, public supplementals(course)}

[3] A permission in the permission transform limits are defined by its: server, object type, operations, and object group specification.

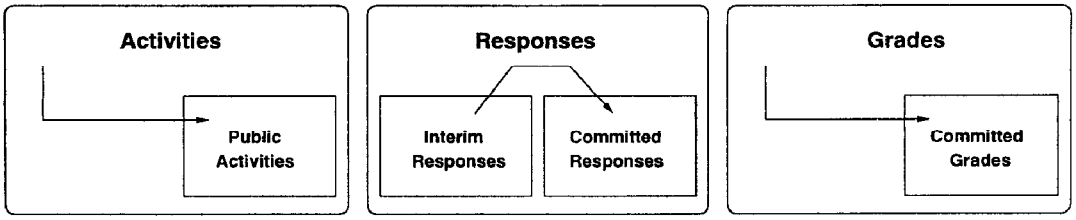

Figure 2: VITC Object Groups

The specified permissions enable students to read the course materials prepared by the instructor after they are released.

Additionally, students need to be able to create responses to activities, but, students must only be able to access their own responses. Therefore, we define a role *student_groups(course, group_id)* to define a group of one or more students in a course. The role definition of student groups is specified as follows:

- **Role:** student groups (course, group id)
- **Administrator:** VITC
- **Permission Transform Limits:**
  - {VITC, VITC obj, rw, interim responses(course, group id)}
  - {VITC, VITC obj, r, committed responses(course, group id)}
  - {VITC, VITC obj, r, committed grades(course, group id)}

Instructors use the VITC to create response objects for activities (i.e., to complete assignments) and assign them to student groups. Only the students in the student group for which the interim response is generated may examine or modify it. The interim response is committed to the VITC whereupon a committed response is created. Instructors commit grades to a specific student group, so only that student group can see the grade.

5.2 User-Role Administration

Another issue is how to manage the assignment of students to student groups. VITC server administrators add students and instructors to a course (e.g., at registration time). Since instructors determine the membership of student groups, they must be able to manage the user-role assignment for these roles.

Using the RBAC model, it is possible to define the user transform limits for the student groups role. The VITC can define that instructors have the following rights to assign students to roles.

- **Role:** student groups (course, group id)
- **Administrator:** instructors (course)
- **User Transform Limits:**
  - students(course)

The constraint that must be enforced here is that the instructors can only assign the students in their courses to student groups for that course.

5.3 Exceptional Course Permissions

In addition to the default course permissions, the VITC clients may wish to distribute some of their rights. For example, an instructor may wish to make an exemplary response available to all students in the course or perhaps in another course. Also, it may be permissible for students to distribute their own committed responses, so they can study for an upcoming exam in a group. The RBAC model can support these exceptions by enabling students and instructors to also act as delegators. However, we want the VITC to be able to properly administer the system, so the fundamental constraints are not violated.

To enable these additional permissions to be delegated, we add the following permission transform limits for the course students.

- **Role:** students (course)
- **Administrator:** VITC
- ... (as above)
- **Administrator:** instructors(course)
- **Permission Transform Limits:**
  - {VITC, VITC obj, r, graded responses}
  - {VITC, VITC obj, r, activities}
- **Administrator:** students(course)
- **Permission Transform Limits:**
  - {VITC, VITC obj, r, graded responses(course)}
  - {VITC, VITC obj, r, committed grades(course)}

This role definition enables both instructors and students to delegate access to the committed responses and grades to all course students. Instructors may be able to provide students with activities and responses from other courses, but students may only distribute responses and grades from the current course.

The VITC must ensure that the delegation does not violate the fundamental access control constraints. Recall the fundamental constraints form Section 4. Since these exceptions add permissions, permission-role constraints need to be checked. Since write access is not propagated by these delegations, the first fundamental constraint cannot be violated. The second constraint prevents students from distributing responses before the submission date. This restriction can be enforced by preventing an instructor from delegating access to a committed response until after it has been graded. Since students cannot have access to all committed responses by default, the membership of objects to the permission transform limit is changed dynamically. Therefore, a constraint predicate is necessary to verify the modification of the transform limit (e.g., specified using first-order logic). Again, a transform is necessary to actually confer the permission to the students (i.e., the permission transform limits only permit the delegation of the right).

6 Constraint Model

From this example, three useful generalizations that can aid in the simplification of safety analysis are apparent. First, it is useful to identify the sets of permissions and principals that can be assigned to a role unconditionally, which we will call *constraint sets*. In the example, these are referred to as the default permission and principal assignments. The creation of such constraint sets enables some safety analysis to be performed at role initiation time. The permissions defined by these constraint sets can be checked directly.

Second, an additional by-product of the constraint set approach in this example is that the lower-level constraints are largely removed by the constraint sets. This appears to be an artifact of the creation of disjoint object groups for the default rights. Since the object groups are disjoint, the default constraint sets do not change. Focusing on management of the exceptional cases simplifies the safety analysis, but may not be as broadly practical in other examples.

Third, in this example, all constraints on assignments to regular roles can be represented as constraint sets. For example, the current constraint sets for permissions and principals can be associated with each regular role (i.e., instructor and student). Actual assignments can be made within these constraint sets by the students, instructors, and the VITC. Changes to the constraint sets are made by administrative roles, and these changes may be limited by constraint predicates. Since all changes to constraint sets in this example are exceptional, approval may also require a second authority.

Therefore, we extend the definition of a regular role to a tuple, $\{n, P, R, CS, C\}$ where: (1) n is the name of the role; (2) P is the set of principals assigned to the role; (3) R is the set of permissions assigned to the role; (4) CS is the constraint sets of the role; and (5) C is the set of other constraints, such as cardinality, that do not affect assignment. A constraint set is also a tuple, $\{A, P, R\}$ where: (1) A is the set of administrators of this constraint set; (2) P is the set of principals that may be assigned in this constraint set; and (3) R is the set of permissions that may be assigned in this constraint set. An administrator can make any assignment within his principal and permission sets.

Thus, constraint sets can provide some degree of dynamicism because a variety of assignments can be made within them.

Other common role constraints, such as role cardinality (number of permissions or principals in a role) and role activation (number of roles that can be activated), would have to be captured by other types of constraints. However, both of these are distinct types of constraints from those about permissions, so they could perhaps be handled as orthogonal constraints. Our example problem does not use such constraints, so more research is necessary here.

Of course, constraint sets themselves are like administrative roles, and similarly, constraints are necessary to manage administration. Such management of changes to the membership of the constraint sets will require some form of higher-level constraints. That is, the recursive application of constraint sets to constraint sets does not appear to yield any benefit: a constraint set already defines administrative limits. At present, two approaches come to mind: (1) common higher-level constraints, such as separation of duty, and (2) general constraint predicates. Traditional RBAC applications can be supported by general constraints, such as separation of duty. Examples, such as VITC, need support for the definition of general predicates on constraint sets.

7 Summary

In this paper, we examine the affect of new features in role-based access control (RBAC) models on the complexity of the RBAC constraint model. While constraints have been a part of RBAC models since the very beginning, the number and complexity of constraints originally was small. Typically, few administrative changes to roles were envisions, and only separation of duty constraints have been needed. With the advent of RBAC model features, such as context-sensitive roles, it is possible for fine-grained management of permissions, and with that comes the need for fine-grained control via constraints. We have examined a specific example of a virtual university and the constraints it requires. Based on this example, we have identified three ideas unconditional role assignment constraints in terms of sets of permissions and principals can be used for all regular roles. Only administrative roles may require constraints in the form of constraint predicates. Thus, safety verification for regular roles can be simplified, and the focus can move to administration actions that change role assignment constraints. In the future, we plan to investigate the constraint model requirements of other examples if they exhibit similar properties.

References

[1] E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information System Security*, 1(2), February 1999.

[2] F. Chen and R. Sandhu. Constraints for role-based access control. In *Proceedings of the 1st Workshop on Role-Based Access Control*, 1995.

[3] D. F. Ferraiolo, J. F. Barkley, and D. R. Kuhn. A role based access control model and reference implementation within a corporate intranet. *ACM Transactions on Information System Security*, 2(1), February 1999.

[4] L. Giuri and P. Iglio. Role templates for content-based access control. In *Proceedings of the Second ACM Role-Based Access Control Workshop*, November 1997.

[5] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, August 1976.

[6] T. Jaeger, T. Michailidis, and R. Rada. Access control in a virtual university. In *Proceedings of the Fourth International Workshop on Enterprise Security*, 1999. To appear.

[7] T. Jaeger, A. Prakash, J. Liedtke, and N. Islam. Flexible control of downloaded executable content. *ACM Transactions on Information System Security*, 2(2), May 1999.

[8] E. C. Lupu and M. Sloman. Reconciling role-based management and role-based access control. In *Proceedings of the Second ACM Role-Based Access Control Workshop*, November 1997.

[9] R. S. Sandhu. The typed access control model. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 122–136, 1992.

[10] R. S. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 model for role-based administration of roles. *ACM Transactions on Information System Security*, 1(2), February 1999.

[11] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control: a multi-dimensional view. In *Proceedings of the Tenth Computer Security Applications Conference*, pages 54–62, 1994.

[12] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[13] D. F. Sterne, M. A. Branstad, B. S. Hubbard, B. A. Mayer, and D. M. Wolcott. An analysis of application specific security policies. In *Proceedings of the 14th National Computer Security Conference*, pages 25–36, 1991.

[14] L. Synder. On the synthesis of protection systems. In *Proceedings of the Sixth ACM Symposium on Operating System Principles*, pages 141–150, November 1977.

[15] D. J. Thomsen. Role-based application design and enforcement. In *Database Security, IV: Status and Prospects*, pages 151–168, 1991.