An Access Control Model for Simplifying Constraint Expression

Jonathon E. Tidswell*
School of Computer Science & Engineering
University of New South Wales
Sydney, NSW 2052, Australia
jont@cse.unsw.edu.au

Trent Jaeger
IBM T. J. Watson Research Center
30 Saw Mill River Road
Hawthorne, NY 10532, USA
jaegert@watson.ibm.com

ABSTRACT

Assurance that an access control configuration will not result in the leakage of a right to an unauthorized principal, called *safety*, is fundamental to ensuring that the most basic of access control policies can be enforced. Safety is achieved either through the use of limited models or the verification of safety via constraints. Currently, almost all critical safety requirements are enforced using limited models because constraint expression languages are far too complex for typical administrators to use properly. We propose a new approach to expressing constraints that has the following properties: (1) an access control policy is expressed using a graphical model in which the nodes represent sets (e.g., of subjects, objects, etc.) and the edges represent binary relationships on those sets and (2) constraints are expressed using a few, simple set operators on graph nodes. While it is possible to extend the semantics of the basic graph model in several ways, and we propose some we found useful, the basic result is that a wide variety of safety policies can be expressed with simple, binary constraints. We demonstrate this model using several examples ranging from safety expression for multilevel security models to separation of duty. Our hope is that this model can be a base for defining critical safety requirements for models that have more flexibility that traditional multilevel models.

1. INTRODUCTION

An important feature of an access control model is the ability to verify the *safety* of its configurations (i.e., the policies expressed using the access control model). A configuration is said to be *safe* if no rights can be leaked to an unauthorized principal [12]. Obviously, the verification that a configuration is safe is necessary to ensure any kind of mandatory access control (MAC) policy, such as multilevel security or separation of duty. Unfortunately, safety cannot be verified for an arbitrary configuration of a general access control model (e.g., Lampson's protection matrix [15]).

To overcome this problem two approaches have been taken: (1) restrict the access control model, such that safety can be proven in general for that model, or (2) augment the access control model with expressions, typically called *constraints*, that describe the safety requirements of any configuration, such that the safety of each configuration can be verified (i.e., ensure that no right is leaked to an unauthorized principal). The first approach limits the flexibility of the access control model, which results in models that either support only coarse-grained policies [4, 8] or models that are difficult to use because it is hard to ensure that the restrictions are satisfied [3]. The second approach suffers from the fact that constraint expression languages are generally complex (e.g., logical languages [1, 5]), so it is difficult to decide whether a set of constraints really expresses the desired safety requirements properly.

The lack of a simple, comprehensive approach to constraints means that restrictive access control models are used to enforce the most important security requirements (e.g., secrecy). For example, Bell-LaPadula [4] and Domain and Type Enforcement (DTE) [8] require completely trusted principals to assign subjects and objects to types (or labels). A slight generalization [16] enables limitation of a trusted principal's ability to assign objects to types. In general, the set of security types and the assignments of subjects to types in these models is fixed, so if the trusted principals are truly trusted then the configurations are safe by default. On the other hand, dynamic models, such as role-based access control (RBAC) are typically used to express *least privilege* policies, such as Chinese Wall [9] and Separation of Duty (SOD) [27]. These policies often require that a principal's or role's rights change dynamically based on some behavior to prevent an unauthorized action (e.g., signing a check).

We observe that there is a continuum in the trade-off between the expressive power of an access control model and the ease of safety enforcement. In a restricted model, such as Bell-LaPadula, constraints are implicit in the model's definition (e.g., a subject of one label cannot write to any object of a 'lower' security label). Therefore, safety enforcement is trivial, but policy expression is limited. On the other hand, general policy expression models, such as RBAC, make constraints explicit concepts and permit the definition of arbitrary constraints. In this case, the expression of safety

*This work was done while this author was on an internship at the IBM T. J. Watson Research Center.

CCS '00 Athens, Greece

requirements has proven to be difficult. However, we have recently found that a variety of common access control policies can be enforced by a few, simple constraint types [30]. Therefore, we claim that an access control model which uses these simple constraints can provide significant expressive power while reducing the complexity of access control policy specification and safety evaluation.

In this paper, we propose a base access control model which enables the definition of restricted access control models and extensions to a general model. The base model consists of subjects, objects, authorization relations, and constraints. However, the subject, object, and authorization relation sets relevant to constraints are expressed explicitly, so constraints can be simply defined in terms of set relationships (e.g., disjoint and subset) on these sets. We then introduce inheritance to the model, and show that variations on typical role-based access control (RBAC) semantics for inheritance are useful for simplifying the resulting models. Further, we identify other modeling concepts that aid in keeping the specification of access configurations and constraints simple. Whether one believes that such concepts may simplify access control modeling and the verification of safety is somewhat subjective, so we describe a number of example constraints using the access control model. Our hope is that with these concepts, access control configurations with greater flexibility and dynamicism can be created where straightforward safety verification is feasible.

The paper is structured as follows. In Section 2, we present background on safety enforcement and discuss the effectiveness of previous approaches. In Section 3, we describe our approach to safety verification. In Section 4, we define our access control model, including the constraint and inheritance models, and demonstrate it on a variety of examples. In Section 5, we discuss the current status of this access control model and how we plan to address some open problems. In Section 6, we conclude and outline future work.

2. BACKGROUND

A wide variety of authorization policies and systems to express those policies have been proposed over the years. While a typical categorization of policies has been between mandatory and discretionary policies, we prefer to distinguish between static and dynamic security policies. In a *static security policy*, subjects and objects are associated with one, fixed security type (or label). A subject's, or object's, type implies the value of its authorization relations. Examples of static security policies include multilevel security (MLS) policies, such as Bell-LaPadula [4], and Domain and Type Enforcement (DTE) [8]. These policies are safe by definition, in that any change in an assignment of a subject or object to a type (or label) can only be done by a fully trusted principal. Mistakes in a change of assignment will not be caught because of this full trust assumption.

These policies, although they have been moderately successful, are quite restrictive. For example, the ability for an object in one MLS category to be transferred to another category is impossible unless it is done by a fully trusted principal. Thus, various alternatives to static security policies have been proposed. A small, but significant, extension of a static security policy is made in the SeaView policy [16] where the trust in downgrader tasks (i.e., those able to change the type of an object) is limited to a subset of the system types. On the one hand, this limits the possibility of error because downgraders can be specialized to their task. On the other hand, more downgraders may violate the safety requirements of the system, so safety verification becomes necessary.

A *dynamic security policy* permits the subjects and objects to be assigned to types and authorization relations as desired, typically by the subjects's, or object's, owner or an administrator. A wide variety of dynamic policies have been defined, but, in general, those that permit system users to control the distribution of access rights to their objects arbitrarily cannot be used to enforce safety. In such cases, a user can give any of their rights to a potentially unauthorized user. Recent advances in security models, such as role-based access control (RBAC) [26], propose enabling the administrators to control the distribution of rights according to their 'administrative rights'. Instead of trusting such administrators completely within the scope of their administrative rights, these models include the concept of *constraints* to verify that the assignment of a permission or user to a role (i.e., authorization relation) is legal.

For dynamic policies, a variety of common restrictions upon the possible values of authorization relations have been identified. First, Clark and Wilson [10] identified the concept of *separation of duty*. Basically, separation of duty requires that each operation in a process be performed by a different person. Therefore, a user may hold the right to execute all operations on a particular object, but it may apply only one of those operations per object. There are many variations on separation of duty, such as: (1) dynamic separation of duty [21], in which a person is restricted from assuming two or more particular roles simultaneously; (2) operational separation of duty [27], in which a user may be restricted from executing all operations in a set on any particular object; and (3) Chinese wall [9], in which accessing an object from one set precludes the future access of objects from a conflicting set. There are also constraints on users, such as the permission sets of two users must be disjoint. Safety verification is necessary to ensure that none of these policies is violated.

Since Harrison, Ruzzo, and Ullman showed that the safety problem was undecidable, research has focused on two areas: (1) determining whether safety could be decided for access control models with limited, but practical, expressive power and (2) defining constraint languages to express verifiable safety requirements.

First, the *take-grant* model has a linear time safety algorithm, but there is still a significant difference in expressive power between take-grant and HRU [28, 7, 22]. Sandhu *et al* eliminates most of this difference in his models (SPM, TAM, ESPM, and non-monotonic ESPM) [21, 3, 22, 2]. They demonstrated that an access control model could be designed for which safety is efficiently decidable (i.e., in polynomial time) given a few restrictions, which were claimed to be reasonable for almost any policy.

Ultimately, despite proven expressive power and safety determination, these access control models have not been adopted in practice. We claim that there are two primary reasons for the lack of acceptance: (a) these models are rather complex to use, both due to the subtlety of the restrictions and the complex relationship between SPM/TAM types and capabilities and (b) it is difficult to both define the safety requirements and write practical algorithms that enforce these requirements. Simply stating an initial config-

uration is difficult enough, but system administrators must also define the safety criteria and, thusfar, few, practical safety algorithms have been implemented.

Second, constraints have been part of most RBAC models of recent years [25, 24, 17, 5, 6], but with a few exceptions highlighted below they have always been specified using rule-based systems. Unfortunately rule-based systems, while highly expressive, are harder to visualize and thus to use; thusfar they have been avoided by practitioners.

Ahn and Sandhu [1] propose a limited logical language called RSL99 for expressing separation of duty constraints in a RBAC model. RSL99 still provides significant expressive power, but remains quite complex. The combination of quantification functions and modeling concept functions makes the constraints expressed in the language difficult to visualize.

Nyanchama and Osborn [18] define a graphical model[1] for role-role relationships which includes a combined view of role inheritance and separation of duty constraints based on roles. Recently Osborn and Guo [20] extended the model to include constraints involving users. However neither the basic model nor the extended model distinguish between accidental relationships and explicitly constructed relationships. Thus these models do not support policies with a historical component. Furthermore (as Nyanchama and Osborn noted) the lack of object typing in RBAC models makes it hard to model workflow constraints. Based on the Dynamically Typed Access Control model [29] we have demonstrated [30] that it is possible to construct graphical representations for most of these constraints in the context of role-based access control. This paper extends and generalizes these ideas focusing on the graphical expression of constraints.

3. AN ALTERNATIVE SAFETY APPROACH

To be able to use constraints to ensure safety we must find both a suitable way to express the constraints, and an efficient algorithm for evaluating or verifying the constraints.

We make two general observations about constraint verification: (1) constraint verification is simply a series of set comparisons and therefore the complexity is mostly dependent on the amount of recursion and quantification in the constraint expressions (2) the complexity in constraint expression is mainly in the expression of the sets that are to be compared.

First, as in the constraints described above, we either want to verify that two sets contain no common members (i.e., are *disjoint*) or that one set does not subsume another set (i.e., is not a *superset*). An example of the latter case is a separation of duty constraint in which a user may perform some, but not all of the operations in a restricted set. The only other type of relationship that we have found necessary is that of a cardinality check. In our approach to safety, constraint expression is built upon these basic concepts. Since constraints specify legal (or illegal) configurations based without reference to the validity of other constraints there is no recursion that dynamic programming (or caching) cannot eliminate and since we do not provide any way to express explicit quantification it cannot be misused.

Second, if constraint verification is a simple set comparison, then the complexity of expressing constraints must be due to the complexity of specifying these sets. Two major causes of complexity are dynamic constraints (e.g., Chinese Wall) and constraints on concepts implicit in the model (e.g., objects of a particular permission). In the first case, the constraint is often specified as a rule (e.g., if user can access object A, then user cannot access object B). As described above, even limited logical languages used to express such rules are difficult to use. In the second case, the aggregation of concepts in access control models, such as permissions, which makes it simpler to express access control policy actually makes it more difficult to express constraints. For example, in role-based access control (RBAC) models a constraint on a particular right to a particular object requires the decomposition of permissions (an aggregate of objects and rights) complicating constraint expression. In our approach, the dynamics of access control policy are captured in the model, and it is possible to decompose aggregate concepts such as permissions into their base concepts, when necessary. For example, permissions are defined in terms of objects and rights, but the administrator only needs to see the objects and rights if they are relevant to some constraint.

We also have found that the typical expression of more advanced access control concepts, in particular inheritance, limits our ability to express constraints easily. In RBAC models, inheritance is a set-subset relation on the permissions of roles. However, we find it useful to define inheritance relations over all system concepts, including subjects and constraints. This makes explicit the associations amongst permissions, subjects, objects, and constraints. In addition, we have found it useful for some policies to express limitations on the permissions and direction of inheritance. For example, in MLS policies only read and execute privileges are inherited upwards, while write privileges are inherited downwards. While Osborn [19] showed that MLS policies can be expressed in a traditional RBAC model, the limitations of inheritance added significant complication to the resulting role graph.

The last major way in which we attempt to simplify constraint expression is by using a graphical model for expressing constraints. That is, we extend the 'graphical role model' used with significant success in RBAC to enable the expression of constraints directly. Thus, our hope is that the need for a completely separate logical language to express constraints can be eliminated. Constraints simply become relations between access control concepts, as inheritance is a relation between roles in RBAC. Obviously, the addition of constraints to the role graph complicates the role graph, but typically constraints only need to be made explicit when constraint specification and analysis is being performed. That is, constraints can be layered over an existing model independently, so the effects of specifying constraints can be hidden when they are not the focus.

In the remainder of the paper, we define and demonstrate our approach to safety, using our model for the simple expression of constraints. We first define the base concepts of the access control model and define the functions for computing the sets upon which constraints will be based. This base access control model consists of subjects, objects, and authorization relations. Since subjects and objects quite often are aggregations themselves, we provide an extensible approach to expressing these concepts, such that access policy and constraints can be effectively specified. We next de-

[1]Their graphs are directed acyclic graphs with a top and a bottom.

fine the constraint model which is based on the set operators identified above. Surprisingly, it has not been necessary to extend these operators, although some extensions to inheritance have helped limit the complexity of the model. Upon review of a number of example policies, two further notions seem important: (1) recurring relationship patterns can be composed into higher-level concepts to further simplify the model and (2) other constraints that limit assignment can be used to simplify the verification of constraints. Finally, we discuss a set of ancillary issues to the application of the graphical safety in practice.

4. OUR MODEL

In this section we introduce a basic access control model consisting of subjects, objects and an authorisation relation, and then using examples extend the basic model to include nodes representing higher level concepts, such as users. Having introduced concept nodes we can define constraints between these nodes, and finally inheritance between instances of concept nodes.

4.1 A Basic Access Control Model

The most primitive access control (authorisation) relation is the matrix identified by Lampson [15]: $subject \times object \rightarrow rights$. We do not assume that authorisation is decided strictly on subjects and objects but take a slightly more abstract view of the authorisation relation as shown in Figure 1, in which elements of the set X are assigned to elements of the set Z, and elements of the set Y are also assigned to elements of the set Z (the assignments are many-to-many).

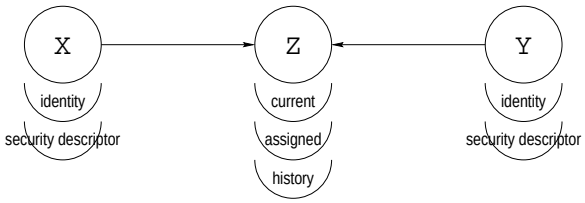

Figure 1: Our basic authentication relationship.

In this model, Z is the authorisation relationship while X and Y are the parameters of the relationship. In the most primitive case X and Y may be subjects and objects (or processes and files); but they may also be multilevel security policy labels; or type enforcement types; or simply user or owner identities.

As a result of defining Z as a relation we can define functions to examine parts of the relationship. Thus we define $X(z)$ to be the subset of X which is assigned to the element $z, z \in Z$; equally we can define the set $Y(z)$ to be the subset of Y assigned to $z, z \in Z$.

We also define $X(y), y \in Y$ and $Y(x), x \in X$ to be the subsets of X and Y that are indirectly connected to y and x as $X(y) \equiv X(Z(y))$ and $Y(x) \equiv Y(Z(x))$. We use the fairly standard shorthand $X(Y)$ for $\bigcup_{y \in Y} X(y)$.

Unfortunately, a simple authorisation relation is inadequate. As Harrison, Ruzzo and Ullman [12] effectively demonstrated, the safety of an access control configuration depends more on the authorisations available in the future than on the current authorisations. Furthermore, many security policies (such as Chinese Wall [9] and operational separation-of-duty [27]) depend not only on the current authorisation relationship but on the history of previous authorisation decisions. Therefore we introduce two distinct aspects to the authorisation relation Z: the authorisations that have ever been activated (*history*), the authorisations that are currently activated (*current*). We can also identify the authorisations that have been assigned but not yet activated (*assigned*). With constraints defining potential inconsistencies between current activations, the set of assigned but unactivated authorisations is a superset or worst-case approximation of the authorisations that may be assigned. We have not seen any use for the history of assigned but unactivated authorisations.

For brevity of expression, we superscript the functions of the authorisation relation to indicate whether we are talking about historic authorisations (Z^H), current authorisations (Z^C) or assigned authorisations (Z^A); consequently we must also superscript the functions derived from Z. Thus we denote the X's that have ever been indirectly related to $y, y \in Y$ by writing $X^H(y)$. Where it does not cause confusion we omit the superscript (C) for currently activated authorisations.

Example 1. We now have enough basics to introduce our first example (see Figure 2), a simple Chinese Wall policy [9] consisting of a set of x's (X) and two sets of y's ($Y1$ and $Y2$) with the restriction that any particular $x, x \in X$, may access y's from only one set, not both. For a more concrete example, consider the situation of a consultant in a large consulting company which has competing companies as clients. It is desirable that the consultant avoid conflict-of-interest (perceived or real) in giving any advice; this is normally achieved by blocking the consultant from seeing confidential information from two or more clients that are, or are likely to be, competing. Since the consultant is likely to remember information previously read from a clients files, consultants must be blocked not only the basis of current clients but of previous clients. It may help to think of X's as consultants and Y's as client files. Effectively a Chinese Wall requires there to be no overlap in the subset of X that has ever had active the authorisation to access an element of $Y1$ with the subset of X that has active the authorisation to access an element of $Y2$ (and vice versa). Since the history of activated authorisations is a (non-strict) superset of the current active authorisations ($X(A) \subseteq X^H(A)$) we can just compare the historical activations of $Y1$ and $Y2$, thus we get the simple constraint: $X^H(Y1) \cap X^H(Y2) = \emptyset$.

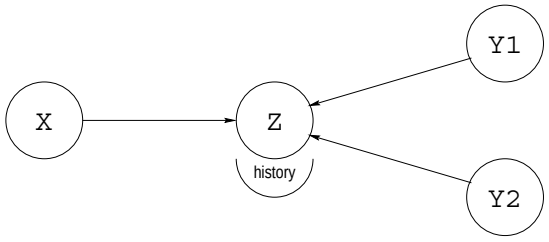

Figure 2: A simple Chinese Wall policy between $Y1$ and $Y2$: the history of Xs acting on $Y1$ should not overlap the history of Xs acting on $Y2$ $X^H(Y1) \cap X^H(Y2) = \emptyset$.

We will show how to specify this constraint graphically when we introduce constraint concepts in Section 4.3.

4.2 Concept Nodes

In the previous section we stated that the parameters (X and Y) to the authorisation relation are not necessarily subjects and objects, and listed some concepts that are commonly associated with X and Y. In Figure 3 subjects and objects are assigned to a concept node which is then in turn assigned to some authorisation relation; thus concept nodes are aggregators and classifiers of subjects and objects. We refer to these as concept nodes because it is our intention to allow security modellers to create nodes representative of whatever concepts are in their security model. In this figure we show only one type of concept node between subjects and objects and the authorisation relation, but as many types of concept nodes may be used as are necessary to capture the desired model.

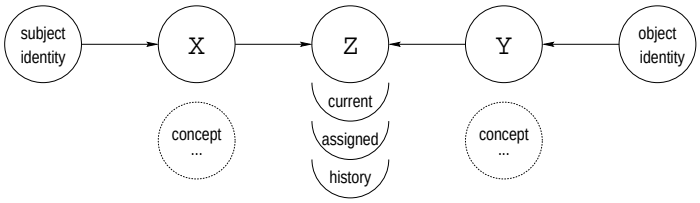

Figure 3: The authentication relationship with one concept node between subjects (objects) and the authorisation relation.

The authorisation relation always reduces to the rights subjects have to objects, so a graph without subjects and/or without objects is incomplete. Thus, we extend our functional notation in a natural manner, and may now denote the subjects (objects) of a particular concept C by $S(C)$ ($O(C)$), and may even denote the objects of a particular subject by $S(O)$, and the inverse by $S(O)$. Of course they must be superscripted appropriately.

Intuitively, aggregation and classification of subjects and objects into concept nodes will make it easier to specify the security policy in the model. The reason is that concept nodes are then used to specify constraints. In some models, such as lattice based secrecy models, the constraints are implicitly captured in the static initial assignment of permissions; while in more dynamic policies, such as role-based access control, constraints are an explicitly formal part of the policy specification.

To identify the nodes necessary for a particular model it is merely necessary to think about the concepts being used:

UBAC In user-based access control the only node that needs to be introduced is one for *users*, subjects are then assigned to users upon creation and access control is based upon user identity.

Capabilities In capability systems it is necessary to introduce a *capability* node which captures the combination of an access type (such as read or write) and a particular object, authorisation is then based on determining whether the subject has a legitimate capability.

LBAC In lattice-based access control models we need to introduce concept nodes for security labels and for exceptions (such as downgraders and assured pipelines).

RBAC In role-based access control it is necessary to introduce concept nodes for users, for roles, for sessions and for permissions (a capability style combination of rights and object identity).

TE In type enforcement models it is necessary to introduce nodes for subject domains and object types.

In addition to these nodes it is common to discuss and think in terms of simple groups, such as groups of objects, or groups of concepts such as groups of users or groups of capabilities (commonly called capability lists). In many cases these groups do not require a distinct type of concept node, however sometimes (such of groups of objects in a model with permissions/capabilities) groups have important, but subtly different, semantics to singleton sets of that concept, so they should be assigned to a different type of concept node.

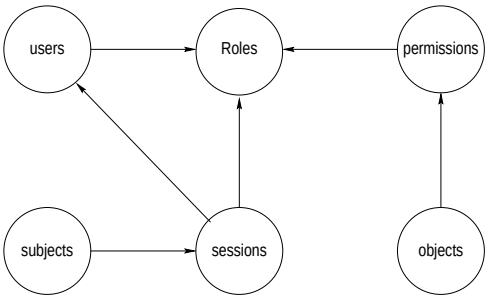

Figure 4: The concept nodes in a classical RBAC model.

Figure 4 shows the concepts and the assigment relationships between them of the basic RBAC0 model of Sandhu et al [25].

Permission or capability nodes are constructed from a fixed, and *a priori* known, set of rights and a dynamic set of objects. This makes it possible to consider known decompositions of permissions in advance. So in addition to being able to describe the permissions that act on an object ($P(o), o \in O$), and the objects referenced by a particular permission ($O(p), p \in P$) we can decompose the permissions and describe all the objects that are accessible by a particular right from some permission; for example, all the objects readable (r) from some permission ($O_r(p), p \in P$), or all the permissions that convey the authority to write (w) to an object ($P_w(o), o \in O$), or all the permissions which have ever conveyed the right to an object ($P_w^H(o)$).

4.3 Constraint Model

The approach we advocate for safety verification is to define an initial configuration of authorisation relationships and to place constraints that limit the ways that the configuration can be modified. The constraints are to ensure that the authorisation relationships always remain acceptable (i.e., are safe).

Since we define our basic model using sets, the natural way to define constraints is as binary relationships between pairs of sets. We chose to limit ourselves to binary relationships for two major reasons. Firstly, they are easy to describe and draw as labelled edges in a two-dimensional graph, which we hope makes them easier to understand. Secondly, they are

simpler and more compact than ternary (or higher) relationships so the algorithms and data structures are more efficient.

In addition to their algorithmic benefits, we believe that the minor loss of expressive power of binary relationships versus ternary relationships is beneficial to modelling: it simplifies the construction of a fixed point, that point necessary to construct constraints on constraints. Furthermore, we have found that many common constraints can be expressed using only binary relationships [30].

There are two broad categories of constraints. The first is based around the notion of subsets and set equality; thus for example, we have test for *equality* ($=$), *subset* ($\subset$), and *not subset or equal* ($\not\subseteq$). In addition to the standard subset operators we define two sets to be *incomparable* ($\not\sim$) if neither is a subset of the other (except in the degenerate case in which one is empty).

$$A \not\sim B \stackrel{\text{def}}{=} (A \not\subseteq B) \wedge (B \not\subseteq A) \vee (A = \emptyset) \vee (B = \emptyset)$$

The second is based around the notion of overlap between two sets when neither is necessarily a subset of the other, and is defined by limiting *maximal cardinality* of their intersection; so we write $|A \cap B| \leq n$ for two sets A and B. The notion of two sets having no overlap, which we refer to as being *disjoint*, is so common that we give it a special symbol ($\perp$), and write $A \perp B$ for $|A \cap B| = 0$.

It is frequently convenient to denote the application of the same function to both sides of a constraint operator by subscripting the operator with the function name. Thus instead of $X^H(A) \perp X^H(B)$ we may write $A \perp^H_X B$. The most common usage of this is apply to constraints to the objects assigned to a node (subscripted o), the rights held by a node (subscripted r) or the inheritance relationships [discussed in Section 4.4] (subscripted $+$).

The separation of duty constraint of our Chinese wall example (see Figure 2) could be written $|X^H(Y1) \cap X^H(Y2)| = 0$ or $X^H(Y1) \perp X^H(Y2)$. By applying the function to the disjoint comparison instead, we can write the constraint as $Y1 \perp^H_X Y2$, or graphically as in Figure 5.

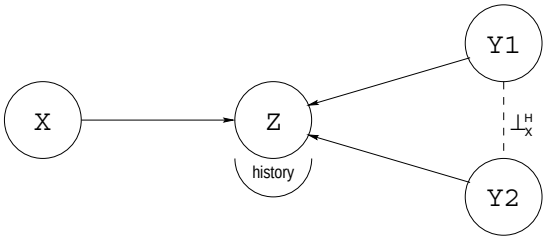

Figure 5: The simple Chinese Wall policy of Figure 2 expressed using a graphical constraint.

To evaluate the the constraint $Y1 \perp^H_X Y2$ we need to find $X^H(Y1)$ and $X^H(Y2)$. Due to the assignment from X to Z it is possible to find $X^H(z), z \in Z$, and due to the assignment of $Y1$ to Z it is possible to calculate $Z^H(y_1)$, $y_1 \in Y1$, by combining these functions we can calculate $X^H(z)$ where $z \in Z^H(y_1)$, and $y_1 \in Y1$ which is $X^H(Y1)$. Similarly we can calculate $X^H(Y2)$, and thus we can evaluate $X^H(Y1) \perp X^H(Y2)$.

We evaluate all constraints using the same pattern, so we could define an arbitrary constraint limiting each $x, x \in X$'s to concurrently having rights to at most 5 $Y1$'s by using a maximal cardinality constraint: $\forall x \in X, |Y1(x)| \leq 5$. This would be draw graphically as an edge from X to itself labelled $|5|_{Y1}$.

Example 2. In a RBAC user-user separation of duty constraint [27], it is forbidden for two users both to be assigned to some particular role. The constraint is enforced by defining a group to which all (both) the users to which the constraint is to apply are assigned and then by defining are constraint between this group and the role to limit the number of users in common. We could write this as $|Users(Group) \cap Users(Role1)| \leq 1$, or using our standard rule of applying functions to constraint operators in preference to both parameters of the operator we write $Group\ |1|^H_{Users}\ Role1$, or if we wished this to apply for all time based on the first activation of the role by a user in the group as $Group\ |1|^H_{Users}\ Role1$ for which is easy to construct the graph.

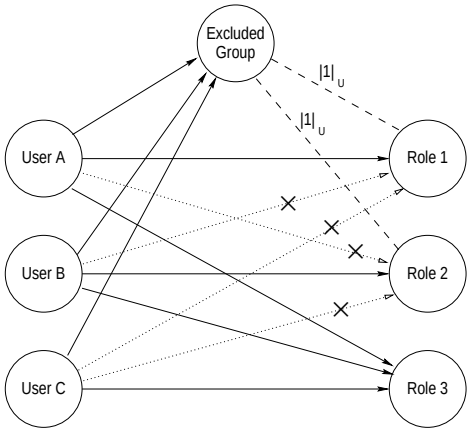

Figure 6: The graphical representation of a user-user separation of duty constraint. Note $UserB$ may not be assigned to $Role1$, and $UserC$ may not be assigned to $Role1$, or $Role2$.

In Figure 6 we have 3 users all belonging to a group with a constraint in relation to 2 of the 3 roles. We could have any number of users in the group and the group can have constraint relationships with any number of roles. This is an example of using concept nodes for aggregation, while it may be possible to implicitly define the group using some quantification or by simply enumerating the users, we believe creating an explicitly named group is clearer and simpler. If it helps, the groups may be defined hierarchically via a number of subgroups which may themselves directly participate in some constraints.

Example 3. In a simple lattice[2] security model information may stay at the same level or flow up the lattice, it may not flow down. So in a simple two-level lattice, subjects at the high level may read objects at high or low security, but only write objects at the high level; while low level subjects may read objects only of the low level but may write either low high level objects. For consistency, objects are restricted to only have one level. This reduces to a simple

[2]See Denning [11] or Sandhu [23] for a more complete explanation of lattice models.

constraint: that high subjects may not have permissions to write any objects that low subjects have permissions to read, $O(P_w(High)) \perp O(P_{rx}(Low))$, or $P_w(High) \perp_O P_{rx}(Low)$. One representation of this is shown in Figure 7.

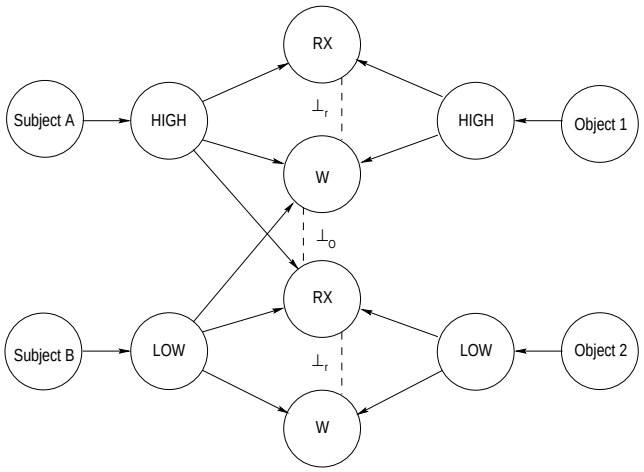

Figure 7: Lattice security implemented with constraints.

While this describes a safe interpretation of the lattice model, the representation will not scale effectively to larger lattices. Each node representing a subject label will have permissions assigned to it for each node in the lattice that dominates it and for each node that it dominates: the number of edges to each node is proportional to the overall size of the lattice not to the local connectivity of each node. For a simple fully ordered lattice of n nodes each node will have $n+1$ permissions assigned to it — it will have two for objects of its own level and one for every level above it (write up) and one for every level below it (read down). Therefore graphs representing even small lattices quickly become an unintelligible tangle of edges. In the next section we define an inheritance mechanism that solves this problem.

4.4 Inheritance

Constraints (implicit or explicit) are a critical part of managing the safety of a security model, but as we demonstrated in the previous section it is easy to construct a representation that does not scale effectively. A large graph that is not easily understood, and is thus hard to check for errors, does not contribute to safety. Therefore we introduce inheritance to support safe efficient reuse of subgraphs by constructing a subset ordering within each concept node. The basic model extended with inheritance is shown in Figure 8.

Inheritance (drawn as an arrow from parent to child labeled with a '+') between concept nodes grants to the child the rights, constraints and other inheritance relationships of the parent.

The inheritance mechanism solves the problem of too many edges in the graph because permissions are implied transitively across multiple inheritance links. Thus each subject label in the lattice in Figure 9 will have a number of incoming edges directly proportional to its local connectivity, not to the overall size of the lattice. (two assignment for objects on its own level and one inheritance each for the level above (write up) and the level below (read down)). In Figure 10 we moved the inheritance to the permissions directly, but

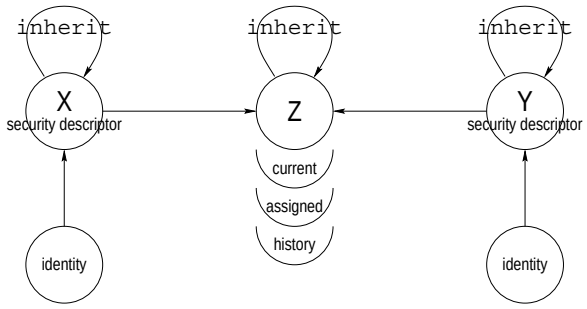

Figure 8: Our authentication relationship with inheritance on concept nodes.

the number of edges is related to the local connectivity of the node not the overall size of the graph.

In Figure 9 we label the permission nodes in our lattice example and inherit up the lattice hierarchy the permissions *P1* and inherit down the lattice the permissions *P3*. This is consistent with the typical description of the flow of permissions in lattice models, but the bidirectional inheritance (High inherits from Low and Low inherits from High) constructs cycles, which is contrary to most other models of inheritance in security models, such as that of RBAC3 [25].

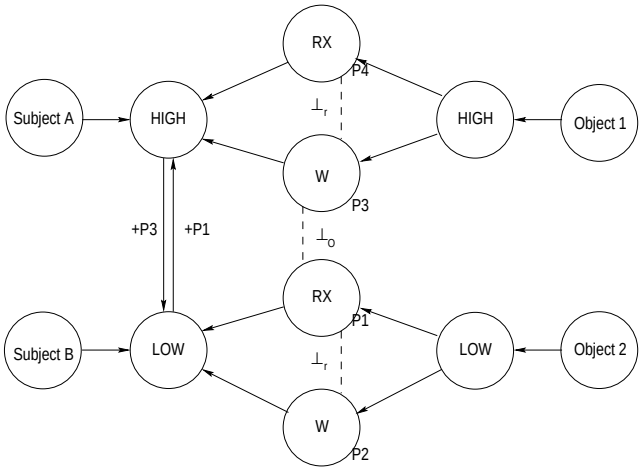

Figure 9: MultiLevel security implemented with constraints and directed inheritance.

While this representation closely parallels the intuitive understanding of permission flows in a lattice, the cost of dealing with cycles and the necessity of labelling arbitrary nodes and edges in order to define inheritance introduces some technical complexity. Another approach is shown in Figure 10 where we define the inheritance relationship between the specific permissions, thus avoiding cycles and the need to provide arbitrary labels for nodes and edges.

So far our examples of inheritance have focussed on inheriting assignments, specifically users and permissions, between nodes. However inheritance may be also be applied to constraints, and since inheritance is transitive it naturally applies to itself.

In Figure 11 solid arrows represent inheritance and the dashed line represents some arbitrary symmetric constraint ρ. Thus C inherits ρ from A then there is an implicit ρ re-

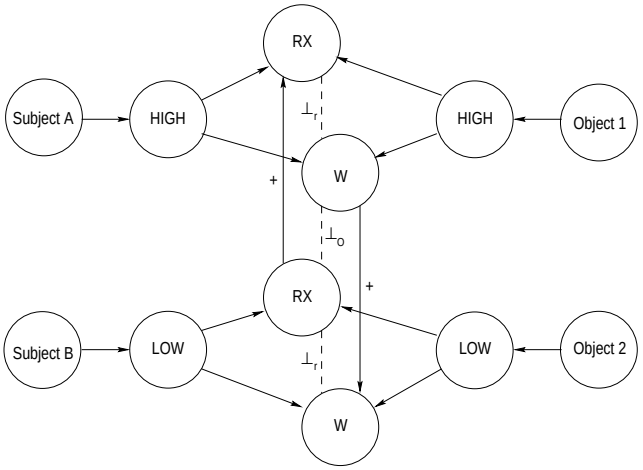

Figure 10: MultiLevel security implemented with constraints and inheritance.

lationship defined on C—B. Further, E inherits from C the inheritance from A, so C inherits ρ from A creating an implicit ρ relationship defined on E—B. Weaker relationships may also be implied, such as between E—D or E—F, but they are dependent on the specifics of ρ.

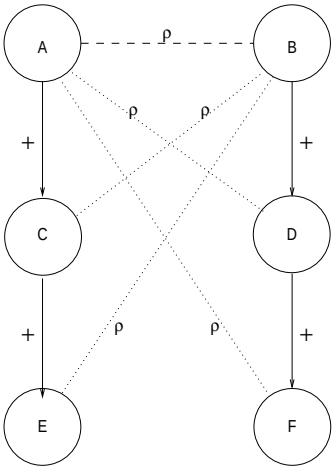

Figure 11: The inheritance of a (symmetric) constraint.

In Figure 12 solid arrows represent inheritance and the dashed arrow represents an assignment from A to F. Then there is an implicit assignment from A to D and from A to B.

Example 4. A simple dynamic separation of duty exists between a group of roles where a user is allowed to have active only one role at a time. This is specified, in Figure 13, in terms of roles having *incomparable* sets of permissions (if any role was a subset of another, the separation would not make sense) and then having the user inherit this constraint. This ensures that even if the user attempts to acquire those permissions via some other role an inconsistency will be identified and the acquisition rejected.

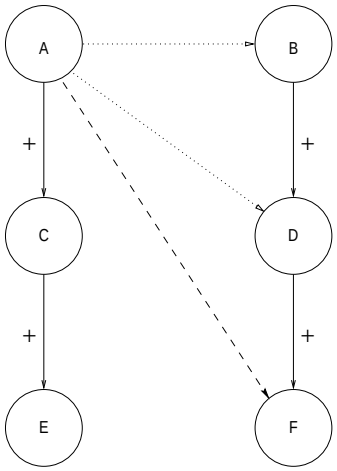

Figure 12: The inheritance of assignment flows from child to parent.

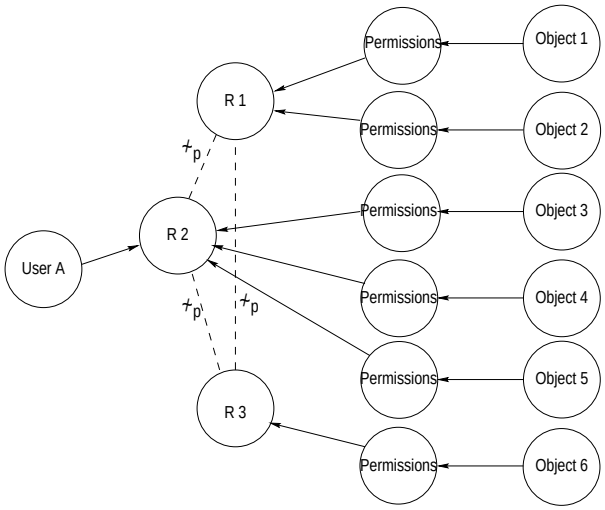

Figure 13: A simple dynamic separation of duty.

Here we have defined the roles to have incomparable sets of permissions rather than disjoint sets of permissions, to demonstrate that in addition to the traditional exclusive definition of mutual exclusion constraints we can model the shared rights mutual exclusion identified by Kuhn [14]. When the user is first assigned to one of the roles the user inherits a constraint against the other roles, the user may not be assigned to any of the other roles as doing so would introduce an inconsistency.

Example 5. Our last example of inheritance is an extended Chinese Wall example, see Figure 14, showing the inheritance of constraints. This figure indicates that at some time in the past the user was assigned to client-4 and therefore had access to an object shared by client-3 and client-4. An attempt to assign client-2 to the user would result in the user inheriting a disjointness constraint on objects belonging to client-3 (including those shared by other clients), since this would introduce an inconsistency with the previous access

to an object shared by clients 2 and 3, client-2 may not be assigned to the user.

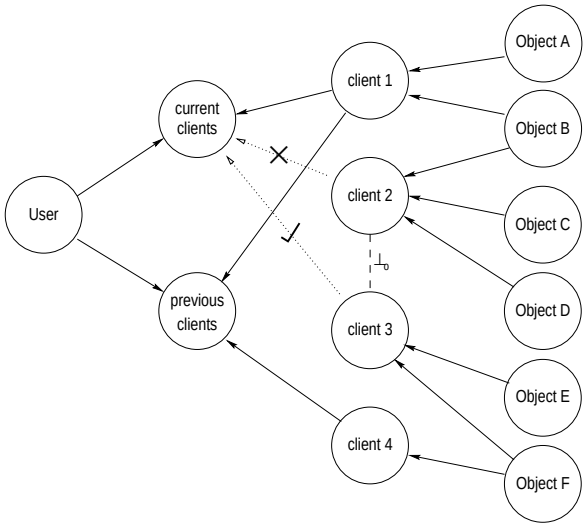

Figure 14: An extended Chinese Wall example showing the inheritance of constraints

5. DISCUSSION

In this section, we discuss a variety of issues related to the definition of an effective constraint model.

5.1 Expressive Power

We have purposefully limited the expressive power of the constraint model to that necessary to express many of the constraints that have been identified in the literature (many are not shown here). For example, constraints are binary relations, but we leave open the possibility of defining n-ary constraints, if necessary. Also, the expressive power of a constraint language is typically compared to that of first order predicate logic. In its current form, constraint expression uses universal rather than existential quantifiers. In general, the need for any explicit quantification greatly complicates the constraint language, so we also view this as an extension that may be necessary, but is discouraged.

5.2 Conflict Resolution

Introducing inheritance naturally introduces the possibility of conflicting specifications. This has been studied before [6; 13, for example] and we adopt the *path override capability* and *denials over grants* policies by defining the inherited relationships to be evaluated in a top down manner with more recent "closer" definitions having precedence, and by evaluating constraints after inheritance and assignment relationships. These policies favour allowing denial-of-service attacks (by denying too much) over breaches of integrity and confidentiality (by allowing too much). We feel this is justified because denial of service attacks are more likely to be brought to an administrators attention, and can thus be addressed, sooner than breaches of integrity and confidentiality which are likely to go unnoticed even if they are not actively hidden.

5.3 Intersection

The combination of security policies (e.g., secrecy and integrity) means that an authorization may require that multiple rights be available. We can solve this either by expressing two role authorization hierarchies, one for secrecy and one for integrity, or by defining *and* relationships on assignments. At present, we prefer the former option.

5.4 Preconditions

Preconditions are useful to limit the ways in which constraints can be violated. For example, if a permission can only be assigned to a set of roles, then it is clear that a constraint on the use of that permission can be defined in terms of the roles, rather than users directly. Since the number of users may be significant, the overhead to compute this constraint is greatly reduced. Even more importantly, the complexity of understanding the impact of that constraint is also reduced. We plan to identify preconditions that can simplify the complexity of individual constraints with the goal of simplifying the overall model.

5.5 Algorithmic Issues

The complexity of verifying a constraint depends on the path length of assignment relationships between the concept nodes in the constraint (e.g., users to objects) and the size of the individual sets. We envision that *dynamic programming* will be valuable in caching the intermediate set values. Since there may be a variety of different sets, management of the state of dynamically programmed data will need to be a first-class issue in the implementation of this model.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a graphical access control model that focuses on the expression of constraints. Previous constraint expression approaches appear to us to be too complex for average system administrators, so we define a new access control model that uses a few basic set concepts to define constraints. Unlike previous efforts we are open to changes in the other modeling concepts, subjects, objects, authorization relations, and inheritance, in order to simplify constraint expression. As a result, we demonstrate a variety of access policies using simple binary relations for constraints. For example, we demonstrate a Bell-LaPadula policy without having to add any new roles, as was the case if a traditional RBAC model was used [19], by generalizing the definition of inheritance. Also, Chinese Wall and separation of duty constraints on both users and permissions are expressed using binary relations. Such relations are easy to verify, and we hold out some hope that they will be simple enough for system administrators to use. We leave open the option of extension of the language (e.g., to n-ary relations or to include existential quantification), but we expect that such extensions must be infrequent for the model to be usable.

7. ACKNOWLEDGEMENTS

We would like to thank John Potter and the anonymous referees for a number of corrections and for numerous suggestions that have improved the readability of the paper.

8. REFERENCES

[1] G. Ahn and R. Sandhu. The rsl99 language for role-based separation of duty constraints. In *Proceedings of the 4th Workshop on Role-Based Access Control*, 1999.

[2] P. Ammann and R. Sandhu. One-representative safety analysis in the non-monotonic transform model. In *Proceedings of the 7th IEEE Computer Security Foundations Workshop*, pages 138–149, 1994.

[3] P. E. Ammann and R. S. Sandhu. Safety analysis for the extended schematic protection model. In *??? Proceeding of the IEEE Symposium on Research in Security and Privacy*, 1991.

[4] D. Bell and L. La Padula. Secure Computer Systems: Mathematical Foundations (Volume 1). Technical Report ESD-TR-73-278, Mitre Corporation, 1973.

[5] 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), Feb. 1999.

[6] E. Bertino, S. Jajodia, P. Samarati, and V. S. Subrahmanian. A Unified Framework for Enforcing Multiple Access Control Policies. In *Proceedings of ACM SIGMOD Conference on Management of Data*, May 1997.

[7] M. Bishop and L. Synder. The transfer of information and authority in a protection system. In *Proceedings of the 7th ACM Symposium on Operating System Principles*, pages 45–54, 1979.

[8] W. E. Boebert and R. Y. Kain. A Practical Alternative to Hierarchical Integrity Policies. In *Proceedings of the 8^{th} National Computer Security Conference*, Gaithersburg, Maryland, 1985.

[9] D. F. C. Brewer and M. J. Nash. The Chinese wall security policy. In *Proceedings of the Symposium on Security and Privacy*, pages 215–228, Oakland, CA, May 1989.

[10] D. D. Clark and D. R. Wilson. A comparison of commercial and military computer security policies. In *Proceeding of the IEEE Symposium on Security and Privacy*, Oakland, California, April 1987.

[11] D. E. Denning. A Lattice Model of Secure Information Flow. *Communications of the ACM*, 19(5):236–242, May 1976.

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

[13] S. Jajodia, P. Samarati, and V. S. Subrahmanian. A Logical Language for Expressing Authorizations. In *Proceedings of the IEEE Symposium on Security and Privacy*, 1997.

[14] D. R. Kuhn. Mutual exclusion of roles as a means of implementing separation of duty in a role-based access control system. In *Proceedings of the 2^{nd} ACM Role-Based Access Control Workshop*, 1997.

[15] B. W. Lampson. Protection. In *Proceedings Fifth Princeton Symposium on Information Sciences and Systems*, March 1971. reprinted in Operating Systems Review, 8, 1, January 1974, pages 18 – 24.

[16] T. Lunt, D. Denning, R. Schell, M. Heckman, and W. Shockley. The SeaView security model. *IEEE Transactions on Software Engineering*, 16(6), June 1990.

[17] E. C. Lupu and M. Sloman. A policy based role object model. In *Proceedings of the 1^{st} IEEE Enterprise Distributed Object Computing Workshop*, October 1997.

[18] M. Nyanchama and S. Osborn. The role graph model and conflict of interest. *ACM Transactions on Information and System Security (TISSEC)*, 2(1), Feb 1999.

[19] S. Osborn. Mandatory access control and role-based access control revisited. In *Proceedings of 2^{nd} ACM Workshop on Role-Based Access Control*, November 1997.

[20] S. Osborn and Y. Guo. Modelling users in role-based access control. In *Proceedings of the 5^{th} ACM Role-Based Access Control Workshop*, July 2000.

[21] R. S. Sandhu. The Schematic Protection Model: Its Definition and Analysis for Acyclic Attenuating Schemes. *Journal of the ACM*, 35(2):404–432, 1988.

[22] R. S. Sandhu. The Typed Access Matrix Model. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1992.

[23] R. S. Sandhu. Lattice-Based Access Control Models. *IEEE Computer*, 26(11):9–19, November 1993.

[24] 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), Feb. 1999.

[25] R. S. Sandhu, E. Coyne, H. L. Feinstein, and C. E. Youman. Role-Based Acess Control Models. *IEEE Computer*, 29(2):38–47, February 1996.

[26] R. S. Sandhu, E. J. Coyne, H. F. Feinstein, and C. E. Youman. Role-based access control: A multi-dimensional view. In *Proceeding of the 10^{th} Annual Computer Security Applications Conference*, December 1994.

[27] R. Simon and M. E. Zurko. Mutual exclusion of roles as a means of implementing separation of duty in a role-based access control system. In *Proceeding of the 10^{th} IEEE Computer Security Foundations Workshop*, June 1997.

[28] L. Synder. On the synthesis and analysis of protection systems. In *Proceedings of the 6th ACM Symposium on Operating System Principles*, pages 141–150, 1977.

[29] J. Tidswell and J. Potter. A Dynamically Typed Access Control Model. In *Proceedings of the Third Australasian Conference on Information Security and Privacy*, July 1998.

[30] J. E. Tidswell and T. Jaeger. Integrated Constraints and Inheritance in DTAC. In *Proceedings of the 5^{th} ACM Role-Based Access Control Workshop*, July 2000.