TRANSFORMATION OF ACCESS RIGHTS

Ravi Sandhu

Department of Computer and Information Science
The Ohio State University, Columbus, Ohio 43210

Abstract

We introduce the concept of transformation of access rights to unify a variety of access-control mechanisms. These mechanisms have mostly been proposed independently of each other to deal with various integrity issues. Their common foundation is abstracted in a model called *transform*. The formalization enables us to investigate the minimal features required to support *transform*. The paper goes on to consider the relation of *transform* to existing access-control models. We show that for the access-matrix model *transform* is outside the class of systems for which safety is known to be decidable. On the other hand we show *transform* is an instance of the decidable cases of the schematic protection model.

1 INTRODUCTION

In this paper we unify a variety of access-control mechanisms which deal with various integrity issues. These mechanisms are mostly taken from the literature. Some have been implemented in actual systems. They all have merit and should certainly be supported, in one form or another, by any protection model which claims to be of general applicability. However considered in isolation these mechanisms are diverse and most have been proposed independently of each other. Simply lumping them together would result in a complex ad hoc model in totality. This is not only inelegant but also casts doubts about prospects for safety analysis (i.e., for determining whether or not a particular subject can obtain a specific right for some given object).

We propose the unifying concept of *transformation of rights* to abstract the common foundation of these mechanisms. Transformation of access rights takes place in two different ways.

1. *Self transformation* or *internal transformation* allows a subject who possesses certain rights for an object to obtain additional rights.

2. *Grant transformation* or *external transformation* occurs in the granting of access rights by one subject to another. The general idea is that possession of a right for an object by a subject allows that subject to give some other right for that object to another subject.

   (If a subject can grant transformed rights to itself external transformation implies internal transformation. In most applications there are additional controls to prevent such "self granting.")

Internal transformations allow us to express consistency in access-control policies such as the requirement that write access implies append access. The well-known technique of amplification [2, 22] for supporting abstract data types and protected subsystems is another instance of internal transformation. The case for abstract data types and protected subsystems is well argued in several classic papers [3, 9, 16, 22]. More recently it has been argued [1] that the "access control triple", which is essentially similar in concept, is necessary for support of integrity policies.

Grant transformations allow us to accommodate various kinds of integrity controls. For instance we can distinguish the ability to access an object from the ability to grant access to that object. This distinction has been suggested as an essential part of "commercial" access-control policies [15] and is implemented in actual systems such as IBM's RACF (Resource Access Control Facility). This distinction of course is one form of separation of duties. Another instance of grant transformations arises when operations on an object are constrained to occur in a specific sequence. This has similarities to the manner in which separation of duties is enforced by transaction control expressions [20].

Section 2 discusses several examples of internal and external transformations in an informal manner. Section 3 develops the *transform* model to unify, and make precise, the common theme running through these examples. This formalization in turn suggests additional applications. It also enables us to investigate the minimal transformation facilities required to support *transform*. In section 4 we go on to consider the relation of *transform* to existing access-control models. We show that for the access-matrix model [6] *transform* is outside the class of systems for which safety is known to be decidable. On the other hand we show *transform* is an instance of the decidable cases of the schematic protection model [18]. This is important since it will be evident from our discussion that transformation of rights is an intricate

CH2703-7/89/0000/0259$01.00

issue for which safety analysis is an essential consideration. Section 5 concludes the paper.

2 EXAMPLES

The simplest example of transformation of rights arises when one right is treated as stronger than another. Consider the typical read, write and append operations on a file, respectively authorized by the rights r, w and a. From the semantics of these operations it is clear that possession of w should imply possession of a. The ability to obtain a weaker right by virtue of possessing a stronger one allows a subject to work with the least privileges needed. In some cases we require the stronger implication that w implies a and both imply r. The motivation is one of integrity in that a subject who writes a file should be able to check whether the writing has been carried out properly, which requires he be able to read the file. This is of course appropriate only in situations where non-disclosure is not an issue.

We can generalize these examples somewhat by allowing different implication relations for different types of files. For instance we may define two types of files respectively with the two implication relations discussed above and a third type of file with no implied rights. However, so long as the ability to obtain implied rights is uniformly available to every subject, internal transformation provides only for consistency in authorization.

Significant power is added by restricting internal transformation to certain subjects. The amplification operation in the Hydra system [2] works in such a fashion, as the basis for implementing abstract data types and protected subsystems. To illustrate amplification consider the example of a stack with push and pop operations implemented in terms of a segment with read and write operations. We need to enforce the following policy.

1. Subjects other than the type manager for stacks can only possess push and pop rights for a stack.
2. The type manager for stacks receives the right to push (or pop) a stack when a subject executes the push (or pop) operation. The manager amplifies the push (or pop) right to obtain r and w rights for the segment containing the stack.
3. Only the type manager for stacks can do such internal transformation.

Predicating the ability to amplify on the type of subject doing the internal transformation enables implementation of abstract data types. Pursuing the example further, we may have stacks implemented in terms of lists which in turn are implemented in terms of segments. Now we have two levels of internal transformation. The first level from push or pop rights to the head, tail or cons rights can only be done by the type manager for stacks. The second level from head, tail or cons rights to r and w rights can only be done by the type manager for lists.

Next consider grant transformations. A simple form of grant transformation occurs with the copy flag which distinguishes between the ability to access an object and the ability to grant access for that object to another subject. The concept goes back to the earliest abstract models for access-control [4, 8] and is a fundamental aspect of discretionary controls. The idea is that possession of a right x authorizes access to the object whereas possession of xc authorizes the ability to grant access to that object to another subject. The xc right is typically made available to the creator of each object. In many models [4, 8, 10, for instance] the ability to grant access is treated as stronger than the ability to perform access, that is possession of xc implies possession of x. Let us for the moment make this assumption, which of course is another example of internal transformation. Now consider the following policies.

1. A user who possesses the xc right for an object can grant the x right for that object to another user.
2. A user who possesses the xc right for an object can grant the xc or x right for that object to another user.

These are both examples of grant transformations. In the first case the xc right is transformed to the x right as part of the grant operation. In the second case there is a choice in the transformation, presumably at the volition of the subject doing the granting. The choice is between the identity transformation of xc to itself or an attenuating transformation of xc to x.

Let us call the copy-flag in the first case as the *one-step copy flag*, denoted xc^1, and in the second case as the *unlimited copy flag*, denoted xc^*. Both these copy flags were proposed in the original access-matrix papers [4, 8]. The interpretation is that xc^* can be transformed to xc^*, xc^1 or x during a grant, whereas xc^1 can only be transformed to x. This idea can easily be generalized to allow for *n-step copy flags* by allowing the grant transformation of xc^n to any one of xc^{n-1}, xc^{n-2}, ..., xc^1 or x. The interpretation of copy flags can also be made to depend on the types of subjects and objects involved in a grant operation. For instance, the copy flag can be interpreted as a one-step flag for sensitive documents whereas for non-sensitive documents it is an unlimited flag. As another example say we distinguish members of a department from outsiders with the policy that the copy flag for grants between members is transformed as an unlimited flag, whereas for grants from a member to an outsider it is transformed as a one-step flag. These are very reasonable policies. It is clear that the possibilities are endless, particularly in large systems with lots of subject and object types.

Next we introduce a new kind of copy flag, called the *separation copy flag*, by dropping the assumption that possession of xc implies possession of x. In this way we draw a clear separation between the ability to grant access and

the ability to perform access. This separation has been suggested by Moffett and Sloman [15] as a fundamental aspect of "commercial" access-control policies. They note such separation is implemented in actual systems citing the example of IBM's RACF (Resource Access Control Facility). In our framework this separation is easily achieved as an instance of grant transformation where xc can only be transformed to x. Now if a subject is allowed to grant to itself the intent of the separation is defeated, since then possession of xc implies possession of x by a grant to oneself. We can prevent this by predicating the grant transformation on the types of subjects involved. Say we distinguish security-officers from users. The transformation of xc to x is allowed in a grant from a security-officer to a user. However in a grant from a security-officer to a security-officer the transformation is from xc to null. This is the policy suggested in [15]. There is the further question of how the ability to grant is obtained in the first place by security-officers. Following [15], this itself can be obtained by grant transformation. The idea is that some user owns the object in question. By possessing the own right for that object the user is authorized to grant xc (by transformation) to a security-officer. That is the owner of an object can delegate the ability to grant access to security-officers. We can play this game again and ask how ownership is acquired. It should be clear by now that this in turn can be achieved by grant transformation if so desired. Alternatively it can be tied to creation of the object or be determined at system initialization.

More general notions of separation of duties can also be viewed as examples of grant transformations to some extent. These relate to sequences of operations on an object which must occur in a prescribed order and must be executed by different types of subjects. For example, consider a policy in which a check is prepared by a clerk, approved by a supervisor and issued by a cashier. This is separation of duties in that the different steps are to be executed by users with different roles (types). We can enforce this policy by transforming the prepare right into an approve right in the clerk to supervisor grant, and again transforming the approve right to an issue right in the supervisor to cashier grant. Note that separation of duties achieved in this way is limited to separation among roles. Consider the modified policy that the check be issued by a clerk, rather than a cashier, with the stipulation that the issuing clerk be different from the one who prepared the check. Controls based solely on types of subjects and objects cannot handle such cases. See [20] for a mechanism which also deals with intra-type separation.

3 THE TRANSFORM MODEL

It is apparent from the foregoing discussion that there is a common theme underlying the several examples we have seen. Our objective in this section is to make this intuition precise by means of a formal model called *transform*. Formalization is rarely useful as an end by itself. In formulating *transform* we hope to uncover additional applications of the transformation theme. An abstract formulation also clarifies the nature of basic mechanisms needed to support such policies in actual systems. Another important goal is to relate *transform* to existing access-control models so as to determine to what extent it is subsumed by these, particularly with regard to safety analysis.

The notion of type is fundamental to most examples we have considered. In fact much of the power of transformation derives from predicating the ability to transform on the types of subjects and objects involved. We therefore assume that subject and objects are classified into types. Object types identify classes of objects which are treated differently for transformation of rights. Subject types similarly identify classes of subjects which have varying ability to transform rights. Subject types also abstract the concept of roles often used in the literature [15, 20, 21, for instance].

We define the sets TS and TO for subject types and object types respectively. Each subject is an instance of some subject type and each object an instance of some object type. We assume strong typing in that the type of a subject or object is determined when it is created and does not change thereafter.

Before considering transformation of rights let us first deal with creation. It is clear subjects need to create objects. There are two issues involved in creation. Firstly subjects need authorization to create objects. Secondly the rights obtained as a result of creation also need to be specified. In *transform* we authorize creation of objects by means of a can-create function as follows.

$$cc: \mathrm{TS} \rightarrow 2^{\mathrm{TO}}$$

The interpretation is that subjects of type u are authorized to create objects of types in $cc(\mathrm{u})$. The effect of creation is defined by create-rules of the following form where R is the set of rights.

$$cr: \mathrm{TS} \times \mathrm{TO} \rightarrow 2^{\mathrm{R}}$$

The interpretation is that when subject U of type u creates an object O of type o the creator U obtains the rights $cr(\mathrm{u,o})$ for O. For example if $cc(\mathrm{user}) = \{\mathrm{file}\}$ and $cr(\mathrm{user,file}) = \{\mathrm{own}\}$ the creator of a file gets the own right for it. For readability we usually drop the set parenthesis around singleton sets, for instance $cc(\mathrm{user}) = \mathrm{file}$ and $cr(\mathrm{user,file}) = \mathrm{own}$.

Now consider the authorization for internal transformation. As discussed earlier internal transformation of rights for an object in a subject's domain involves consideration of their types. So what we need is an internal transformation function of the following form.

$$itrans: \mathrm{TS} \times \mathrm{TO} \times \mathrm{R} \rightarrow 2^{\mathrm{R}}$$

The interpretation of $itrans(\mathrm{u,o,x}) = \{\mathrm{x}_1,\ldots,\mathrm{x}_n\}$ is that a subject of type u who has the x right for an object of type o can obtain the $\mathrm{x}_1, \ldots, \mathrm{x}_n$ rights for that object by internal

transformation. For example, the policy that write implies append and both imply read can be stated in either of the following ways.

$$\begin{aligned} \textit{itrans}(\text{user, file, w}) &= \{\text{a, r}\} \\ \textit{itrans}(\text{user, file, a}) &= \text{r} \\ \textit{itrans}(\text{user, file, r}) &= \phi \end{aligned}$$

(a)

$$\begin{aligned} \textit{itrans}(\text{user, file, w}) &= \text{a} \\ \textit{itrans}(\text{user, file, a}) &= \text{r} \\ \textit{itrans}(\text{user, file, r}) &= \phi \end{aligned}$$

(b)

In (a) the transformation from w to r is achieved directly whereas in (b) it is done indirectly in two steps. We allow for either formulation in the model. The amplification example of a stack implemented by a list which in turn is implemented by a segment can be specified as follows.

$$\begin{aligned} \textit{itrans}(\text{stack-manager, stack, pop}) &= \{\text{head, tail}\} \\ \textit{itrans}(\text{stack-manager, stack, push}) &= \text{cons} \\ \textit{itrans}(\text{list-manager, stack, head}) &= \{\text{r, w}\} \\ \textit{itrans}(\text{list-manager, stack, tail}) &= \{\text{r, w}\} \\ \textit{itrans}(\text{list-manager, stack, cons}) &= \{\text{r, w}\} \end{aligned}$$

All other values of *itrans* are empty

Here the ability to amplify push and pop to head, tail or cons is restricted to the stack manager, while amplification from head, tail and cons to r and w is restricted to the list manager. Realistically of course these would be fragments of a larger specification involving additional types.

The internal transformation function generalizes in an obvious way as follows to amplify sets of rights (as opposed to single rights).

$$\textit{itrans}: \text{TS} \times \text{TO} \times 2^{\text{R}} \rightarrow 2^{\text{R}}$$

The interpretation of $\textit{itrans}(\text{u,o,}\{x_1,\ldots,x_n\}) = \{y_1,\ldots,y_m\}$ is that a subject of type u who has *all* the x_i rights specified on the left hand side for an object of type o can obtain the rights $y_1, \ldots, y_m$ for that object by internal transformation. This is useful in situations described as synergistic authorization in [13] and as command authorization in [5]. For instance consider a situation where a scientist (abbreviated as sci) needs approvals from a security officer and a patent officer before he can release a document (abbreviated as doc) for publication. Say these two approvals are respectively signified by possession of the a_s and a_p rights. We can express this policy as follows.

$$\textit{itrans}(\text{sci, doc, } \{\text{own, } a_s, a_p\}) = \text{release}$$

A scientist then needs to be the owner of a document and must possess the two approvals before he can obtain the right to release the document. The synergy in this internal transformation occurs only if we can guarantee that the a_s and a_p rights are obtained from two independent sources. As we will see this can be achieved by grant transformations.

Grant transformations can be modeled as a *grant* function of the following form.

$$\textit{grant}: \text{TS} \times \text{TS} \times \text{TO} \times \text{R} \rightarrow 2^{\text{R}}$$

The interpretation of $\textit{grant}(\text{u,v,o,x}) = \{x_1,\ldots,x_n\}$ is that a subject of type u who has the x right for an object of type o can grant *one or more* of the $x_1, \ldots, x_n$ rights for that object to a subject of type v. The unlimited copy flag xc^* and the one-step copy flag xc^1 of section 2 can then be specified as follows.

$$\begin{aligned} \textit{grant}(\text{user, user, file, } xc^*) &= \{xc^*, xc^1, \text{x}\} \\ \textit{grant}(\text{user, user, file, } xc^1) &= \text{x} \\ \textit{grant}(\text{user, user, file, x}) &= \phi \end{aligned}$$

The extension to n-step copy flags is obvious. There are actually several ways of specifying even this rather simple policy. For instance we could combine grant and internal transformations to achieve the same net effect as follows.

$$\begin{aligned} \textit{grant}(\text{user, user, file, } xc^*) &= \{xc^*, xc^1\} \\ \textit{itrans}(\text{user, file, } xc^*) &= xc^1 \\ \textit{grant}(\text{user, user, file, } xc^1) &= \text{x} \\ \textit{grant}(\text{user, user, file, x}) &= \phi \end{aligned}$$

This property appears to be inevitable in any sophisticated model. We cannot realistically hope to have a unique specification for a policy in a general model.

The separation copy flag of section 2 is also easily specified as follows.

$$\begin{aligned} \textit{grant}(\text{user, security-officer, file, own}) &= \text{xc} \\ \textit{grant}(\text{security-officer, user, file, xc}) &= \text{x} \\ \textit{itrans}(\text{security-officer, file, xc}) &= \phi \end{aligned}$$

That is a user who owns a file can delegate the authority to grant access to that file to a security officer. The security officer can grant access to that file to other users but cannot himself access it.

Next let us go back to the example of a scientist who needed multiple approvals for releasing a document for publication. We had mentioned that consideration of grants is required for a complete statement. One possibility is shown below.

$$\begin{aligned} \textit{grant}(\text{sci, security-officer, doc, own}) &= \text{review} \\ \textit{grant}(\text{sci, patent-officer, doc, own}) &= \text{review} \\ \textit{grant}(\text{security-officer, sci, doc, review}) &= a_s \\ \textit{grant}(\text{patent-officer, sci, doc, review}) &= a_p \\ \textit{itrans}(\text{sci, doc, } \{\text{own, } a_s, a_p\}) &= \text{release} \end{aligned}$$

As the owner of a document a scientist can request it be reviewed by a security-officer and a patent-officer by granting them the review right. In turn they can grant the scientist

who gave them the review right appropriate approval rights. Finally the scientist can internally transform these rights to acquire the release right.

Consider a slight modification to the above policy. Say that we require separation of duties regarding release of a document. A scientist is responsible for gathering the necessary approvals. The actual release however must be done by a librarian who is responsible for cataloging information about the document before releasing it. To achieve this we can replace the internal transformation above by the following grant transformation.

$$grant(\text{sci, librarian, doc, } \{\text{own, } a_s, a_p\}) \quad = \quad \text{release}$$

To do so we can generalize *grant* as follows in the same way that *itrans* was generalized.

$$grant: \text{TS} \times \text{TS} \times \text{TO} \times 2^{\text{R}} \rightarrow 2^{\text{R}}$$

The interpretation of $grant(u,v,o,\{x_1,\ldots,x_n\}) = \{y_1,\ldots,y_m\}$ is that a subject of type u who has *all* the x_i rights specified on the left hand side for an object of type o can grant *one or more* of the rights $y_1, \ldots, y_m$ for that object to a subject of type v.

This completes our description of the model. To summarize we have the following definition for *transform*.

Definition 1 A policy for transformation of rights is stated in *transform* by specifying the following (finite) components.

1. Disjoint sets of subject types TS and object types TO.
2. A set of rights R.
3. A can-create function $cc: \text{TS} \rightarrow 2^{\text{TO}}$.
4. Create-rules $cr: \text{TS} \times \text{TO} \rightarrow 2^{\text{R}}$.
5. An internal transformation function $itrans: \text{TS} \times \text{TO} \times 2^{\text{R}} \rightarrow 2^{\text{R}}$.
6. A grant transformation function $grant: \text{TS} \times \text{TS} \times \text{TO} \times 2^{\text{R}} \rightarrow 2^{\text{R}}$.

Having formulated this abstract model the natural question is what can be learned from it. It is clear that the abstraction captures a wide variety of powerful features. The full generality of the abstraction is convenient for specifying a desired policy. However the question does arise as to what are the minimal features required to support this model in an actual implementation. For instance do we really need the synergistic ability in *grant* as well as in *itrans*. The scope of this paper does not permit us to investigate such questions completely. We focus on one issue of particular importance. In their general formulation both *itrans* and *grant* are amplifying in that they are required to create new rights. Since this is a powerful facility the question is to what extent can it be minimized. Let us make this question precise.

It is clear that internal transformations are useful only if they are amplifying in the sense that new rights are obtained. That is we can assume without loss of generality,

$$itrans(u,o,R_i) = R_j \Rightarrow R_j \cap R_i = \phi$$

Next consider the grant transformation $grant(u,v,o,R_i) = R_j$. That is possession of R_i rights enables transfer of R_j rights. Clearly if $R_j \subseteq R_i$ such a grant is attenuating or non-amplifying in that the source subject cannot give away rights that he does not possess. But note that the source subject may be able to internally amplify the R_i rights, so in defining attenuation we need to also consider implied rights. Now implied rights can be obtained directly by one application of *itrans* or indirectly by several applications. This leads us to the following definition.

Definition 2 Let *itrans** be the transitive closure of *itrans*. A grant transformation is *attenuating* provided

$$grant(u,v,o,R_i) = R_j \Rightarrow R_j \subseteq R_i \cup itrans^*(u,o,R_i)$$

Otherwise it is *amplifying*.

For example, $grant(\text{user,user,file,x}) = \text{x}$ is trivially attenuating. On the other hand for $grant(\text{user,user,file,xc}) = \text{x}$ we need to consider the interpretation of the copy flag. With the assumption that xc is strictly stronger than x, i.e., $itrans(\text{user,file,xc}) = \text{x}$, the latter grant is attenuating. However for the separation copy flag with $itrans(\text{user,file,xc}) = \phi$ this grant is amplifying. This is clearly consistent with the intuitive concept of attenuation.

One can take issue with this definition in that we are ignoring implied rights in the destination domain. That is what we really need is the following requirement

$$grant(u,v,o,R_i) = R_j \Rightarrow \begin{array}{l} R_j \cup itrans^*(v,o,R_j) \subseteq \\ R_i \cup itrans^*(u,o,R_i) \end{array}$$

Let us call such grants *strictly attenuating*. This requirement is very strong and will not allow for the grants required to support abstract data types or protected subsystems, as illustrated by our stack example. These features are of such fundamental importance that it is clear we cannot limit ourselves to strictly attenuating grants in the framework of *transform*.

The question therefore is whether or not we can limit ourselves to attenuating grants. In other words, do amplifying grants add any power not already available with amplifying internal transformations? The answer turns out to be no, i.e., grant amplifications can be built out of internal amplifications. This observation is significant because it allows us to conceive of mechanisms which only allow internal amplification and require attenuating grants and yet realize the power of *transform*. To see the redundancy of amplifying grants consider the separation copy flag specified earlier as follows.

grant(user, security-officer, file, own) = xc
grant(security-officer, user, file, xc) = x

These grants are clearly amplifying. An equivalent policy with attenuating grants is achieved by introducing new right symbols as follows.

itrans(user, file, own) = delegate
grant(user, security-officer, file, delegate) = delegate
itrans(sec-off, file, delegate) = xc

itrans(sec-off, file, xc) = cando-x
grant(security-officer, user, file, cando-x) = cando-x
itrans(user, file, cando-x) = x

The two amplifying grants of the original policy are respectively simulated by the two sequences above. The general principle is evident from this example. Each amplifying grant is simulated by an internal amplification at the source, followed by a grant with the trivial and attenuating identity transformation, finally followed by another internal transformation at the destination.

A general construction can be outlined as follows. Let $r \in grant(u,v,o,R_i)$ and $r \notin R_i \cup itrans^*(u,o,R_i)$. That is r makes this grant amplifying. Modify the given *transform* specification as follows.

1. Define the new right $\underline{r.u.v.o.R_i}$. The entire symbol signifies a single right. The components in this symbol emphasize that we need a new right for each combination of the components.
2. Modify $itrans(u,o,R_i)$ to include $\underline{r.u.v.o.R_i}$.
3. Modify $grant(u,v,o,R_i)$ by replacing r with $\underline{r.u.v.o.R_i}$.
4. Define $itrans(v,o,\underline{r.u.v.o.R_i}) = r$.

It is clear that r no longer makes this modified $grant(u,v,o,R_i)$ amplifying. By repeating this procedure we can therefore get rid of all amplifying grants. Since new rights are introduced for each iteration of this procedure there is no interaction between different amplifications removed in this way. The original amplifying grants are then simulated as follows.

$$r \in grant(u,v,o,R_i) \Leftrightarrow \begin{cases} itrans(u,o,R_i) = \underline{r.u.v.o.R_i} \\ \underline{r.u.v.o.R_i} \in grant(u,v,o,R_i) \\ itrans(v,o,\underline{r.u.v.o.R_i}) = r \end{cases}$$

The correctness of this construction is self evident. A formal inductive proof can be given showing every reachable state with the former policy has an equivalent counterpart with the modified policy, and vice versa. The details are tedious and shed little insight. We summarize the above discussion by the following theorem.

Theorem 1 *It can be assumed without loss of generality that all grant transformations are attenuating.*

Next consider the question of whether internal amplification is necessary? It appears to be so in the framework of *transform*. However by a slight shift of viewpoint we can replace internal amplification by an attenuating operation. We will return to this issue in section 4.2.

4 INSTANTIATIONS OF TRANSFORM

In this section we consider the relation of *transform* to existing access-control models, specifically the access-matrix model as formalized in [6] and the schematic protection model (SPM) [18]. We have two objectives in doing so. First the ability to instantiate *transform* is a measure of the expressive power of these models. Secondly the SPM instantiation shows that safety is decidable for *transform*. Whereas for the access-matrix model *transform* cannot be instantiated within its decidable cases. This serves as a demonstration of the superior analysis framework provided by SPM.

4.1 THE ACCESS-MATRIX MODEL

The access-matrix model was originally proposed by Lampson [8]. The matrix has a row and a column for each subject and a column for each object. The [I,J] cell contains rights which subject I possesses for the subject or object J. In the original formulation a specific set of rules for modifying the matrix were defined. These correspond to a policy in which the creator of an object has complete discretion over access to it. Harrison, Ruzzo and Ullman [6] observed that the rules for modifying the matrix should properly be determined by the context for each system. They proposed these rules be defined by a set of commands. Each command has a condition part and a body. The condition specifies rights required to exist in the matrix before the body can be executed for its actual arguments. The condition can test for presence of rights but not for their absence. Formally the condition is composed of terms of the form $r \in [X,Y]$ where r is a right, X is a row and Y a column of the matrix. These terms are combined by conjunction or disjunction but negation cannot be used. The body consists of a sequence of primitive operations. The primitive operations enter or delete a right from a cell of the matrix or create a new row or column or destroy an existing row or column.

It turns out that in this model, henceforth called HRU, safety is undecidable under surprisingly weak assumptions. Specifically it was shown by Harrison and Ruzzo [7] that safety is undecidable even if the conditions can have at most two terms. Limiting conditions to a single term is very restrictive. We argue that *transform* cannot be instantiated with such commands. For simplicity we consider the simpler form of *transform* in which only single rights are transformed. The argument thereby applies a fortiori to the more general case of *transform*.

HRU does not have a notion of type built in. However types can be simulated by right symbols. Let X be of type x. This can be represented by putting the x right in the [X,X] cell. This of course requires that the objects of *transform* be treated as HRU subjects, since the [X,X] cell requires a row and column for X. This is acceptable because the only right in row X is the one representing the type of X. With this method for simulating types we can instantiate *transform* in HRU as follows.

1. Let the set of HRU right symbols be R ∪ TS ∪ TO.
2. For every o∈*cc*(u) with *cr*(u,o) = {r_1,...,r_k} define the command:

```
command create.u.o(U,O)
    if    u∈[U,U]
    then create row O;
         create column O;
         enter o in [O,O];
         enter r1 in [U,O];
         ...;
         enter rk in [U,O];
end
```

3. For every *itrans*(u,o,r) = {r_1,...,r_k} define the command:

```
command itrans.u.o.r(U,O)
    if    u∈[U,U] ∧ o∈[O,O] ∧ r∈[U,O]
    then enter r1 in [U,O];
         ...;
         enter rk in [U,O];
end
```

4. For every r_i∈*grant*(u,v,o,r) define the command:

```
command grant.u.v.o.r.ri(U,V,O)
    if    u∈[U,U] ∧ v∈[V,V] ∧ o∈[O,O] ∧
          r∈[U,O]
    then enter ri in [V,O];
end
```

The commands simulating *itrans* and *grant* have multiple terms in their conditions. This appears to be inevitable irrespective of how we encode the types. So HRU certainly has the expressive power to simulate *transform*. However it cannot do so within its decidable cases for safety.

4.2 THE SCHEMATIC PROTECTION MODEL

We begin with a brief review of SPM. The dynamic privileges in SPM are tickets of the form Y/x where Y identifies some unique entity (subject or object) and x is a right. SPM subjects and objects are strongly typed. The type of a ticket is determined by the type of entity it addresses and the right symbol it carries, that is type(Y/x) is the ordered pair type(Y)/x. Tickets are acquired in accordance with rules which comprise the scheme which is defined by specifying the following (finite) components.

1. Disjoint sets of subject types TS and object types TO. Let T = TS ∪ TO.
2. A set of rights R. The set of ticket types is thereby T×R.
3. A can-create function $cc: TS \rightarrow 2^T$.
4. Create-rules of the form $cr_p(u,v) = c/R_1 \cup p/R_2$, $cr_c(u,v) = c/R_3 \cup p/R_4$.
5. A collection of link predicates {$link_i$}.
6. A filter function $f_i: TS \times TS \rightarrow 2^{T \times R}$ for each predicate $link_i$.

The Create Operation. Subjects of type u can create entities of type v if and only if v∈*cc*(u). Tickets introduced as the side effect of creation are specified by a (different) create-rule for every (u,v) such that v∈*cc*(u). Each create-rule has two components shown above, where p and c respectively denote parent and child and the R_i's are subsets of R. When subject U of type u creates entity V of type v the parent U gets the tickets V/R_1 and U/R_2. The child V similarly gets the tickets V/R_3 and U/R_4. For example, file∈*cc*(user) authorizes users to create files. And cr_p(user,file) = c/rw and cr_c(user,file) = ϕ gives the creator r and w tickets for the created file.

The Copy Operation. A copy of a ticket can be transferred from one subject to another leaving the original ticket intact. SPM has a copy flag built in which we denote as k to distinguish it from the copy flags of *transform*. Possession of Y/xk implies possession of Y/x but not vice versa. Let dom(U) signify the set of tickets possessed by U. Let x:k denote x or xk, with multiple occurrences in the same context either all read as x or all as xk. Three independent pieces of authorization are required to copy Y/x:k from U to V.

1. Y/xk∈dom(U), i.e., U must possess Y/xk for copying either Y/xk or Y/x.
2. There is a link from U to V. Links are established by tickets for U and V in the domains of U and V. The predicate $link_i$(U,V) is defined as a conjunction or disjunction, but not negation, of one or more of the following terms for any z∈R: U/z∈dom(U), U/z∈dom(V), V/z∈dom(U), V/z∈dom(V), and **true**. Some examples from the literature are given below [10, 11, 14, 17, respectively].

$$link_{tg}(U,V) \equiv V/g \in dom(U) \vee U/t \in dom(V)$$
$$link_{t}(U,V) \equiv U/t \in dom(V)$$
$$link_{sr}(U,V) \equiv V/s \in dom(U) \wedge U/r \in dom(V)$$
$$link_{u}(U,V) \equiv \textbf{true}$$

3. The last condition is defined by the filter functions $f_i : TS \times TS \rightarrow 2^{T \times R}$, one per predicate $link_i$. The value of $f_i(u,v)$ specifies types of tickets that may be copied from subjects of type u to subjects of type v over a $link_i$. Example values are $T \times R$, $TO \times R$ and ϕ respectively authorizing all tickets, object tickets and no tickets to be copied.

In short $Y/x:k$ can be copied from U to V if and only if

$$Y/xk \in dom(U) \wedge (\exists link_i)[link_i(U,V) \wedge y/x:k \in f_i(u,v)]$$

where the types of U, V and Y are respectively u, v and y. Note that f_i determines whether or not the copied ticket can have the copy flag. This completes our review of SPM. Motivation for defining SPM in this manner and its resulting expressive power are discussed in [17, 18, 19].

Now consider how *transform* can be instantiated in SPM. Note the SPM copy operation is attenuating in that a subject must possess a ticket which is copied from its domain. We have seen in theorem 1 that a *transform* policy can be assumed to have only attenuating grants. To make the grants correspond to SPM copy operations we can go one step further and replace every $grant(u,v,o,R_i) = R_j$ by the following

$$grant(u, v, o, R_i \cup itrans^*(u,o,R_i)) = R_j$$

This will guarantee that R_j is a subset of rights on the left hand side. We must still deal with amplifying internal transformations in some way. To do so we simulate a *transform* object as an SPM subject of limited abilities. The crucial additional assumption is that each object possesses all rights for itself. These rights can always be introduced by the create-rules. Since an object is a passive entity this assumption is quite harmless. Moreover we do not allow grants to object so these are the only rights an object will ever possess. Also we understand that a grant from an object to a subject is actually initiated by the subject.

With this setting we can simulate internal transformations by SPM copy operations. Consider $itrans^*(u,o,R_i) = R_j$. Let U be a *transform* subject of type u and O a *transform* object of type o. These are respectively modeled as SPM subjects of types u and o respectively. Let the possession of O/R_i by U set up a $link_{R_i}$ from O to U. The internal transformation is effected by defining $f_{R_i}(u,o)$ to be o/R_j.

The only tickets that O can ever possess are tickets for itself so the copy operation authorized in this manner has precisely the same effect as the internal transformation. The SPM copy flag is irrelevant to the construction and we assume it is allowed to be carried along by every filter function we define.

This construction is formally expressed by the following SPM scheme for a given instance of *transform*, which is assumed (without loss of generality) to have attenuating *grant*'s.

1. $TS' = TS \cup TO$, $TO' = \phi$
2. $R' = \{x:k|\ x \in R\}$
3. For all $u \in TS$: $cc'(u) = cc(u)$
   For all $o \in TO$: $cc'(o) = \phi$
4. $cr_p'(u,o) = c/R_1$, where $cr(u,o)=R_1$
   $cr_c'(u,o) = c/R'$
5. Define the following link predicates
   $link_u(U,V) \equiv$ **true**
   $link_{R_i}(O,U) \equiv O/R_i \in dom(U)$, for all $R_i \subseteq R$
6. Let R_jk denote $\{xk|\ x \in R_j\}$
   Define $f_{R_i}(o,u) = o/R_jk$, where $itrans^*(u,o,R_i) = R_j$
   Define $f_u(u,v) = \{o/R_jk|\ (\exists R_i)\ grant(u,v,o,R_i)=R_j\}$
   All other values of the filter functions are empty

The simulation can be summarized as follows.

$$itrans^*(u,o,R_i)=R_j \Leftrightarrow \begin{cases} link_{R_i}(O,U) \equiv O/R_i \in dom(U) \\ f_{R_i}(o,u) = o/R_jk \end{cases}$$

$$r \in grant(u,v,o,R_i) \Leftrightarrow \begin{cases} link_u(U,V) \equiv \textbf{true} \\ o/rk \in f_u(u,v) \end{cases}$$

An internal transformation is replaced by a subject copying the transformed tickets from the object's domain. For grant transformations we have already shown that we can assume $r \in R_i$ so they are reduced to copying a ticket over the universal link. Formal correspondence between the original *transform* policy and the constructed SPM scheme can be established by a straightforward inductive proof that the reachable states in both cases are equivalent.

The SPM instantiation is instructive in two respects. Firstly it establishes that safety is decidable for *transform*. This follows from the known result for SPM [18] that safety is decidable provided *cc* is acyclic in the following sense: the directed graph with edges $\{(u,v)|\ v \in cc(u)\}$ is acyclic. Since the only edges in this graph for *cc'* are from types in TS to types in TO, *cc'* is trivially acyclic. Moreover this graph for *cc'* is sparse which guarantees that the decision procedure is efficient [18].

Secondly the instantiation shows that amplification is not required to support *transform*. Internal amplification can be simulated by an attenuating copy operation so long as the amplified tickets are available in some domain. Our construction in particular requires that object tickets be created along with each object and stored as part of the object itself. This is a very reasonable implementation. Its advantage is that creation of tickets (capabilities) is thereby coupled with object creation, at which point we anyway need to create tickets in the creator's domain. By isolating ticket creation in this manner we can limit the use of this very sensitive function. Thereafter tickets can only be copied from one domain to another in an attenuating fashion.

We note that Minsky [12] establishes a similar result for his operation-control model. In addition to tickets this model has a new kind of privilege called an activator. Minsky shows that abstract data types in particular can be realized by attenuating external and internal transformations of activators and tickets. Our construction achieves the same effect without requiring activators. Instead we replace amplifying internal transformations by attenuating copy operations.

5 CONCLUSION

To summarize we have described a wide variety of access-control mechanisms from the literature with the common theme of transformation of access rights. We have unified these mechanisms in a model called *transform*. We have shown that *transform* cannot be instantiated within the cases of the access-matrix model [6] for which safety is decidable. On the other hand *transform* can be instantiated within the efficiently decidable cases of SPM [18]. We have also shown that *transform* can be implemented without any amplifying primitives.

We are convinced that any protection model which claims to be of general applicability should be able to instantiate *transform*. In fact such a model is likely to need facilities beyond those provided by *transform*. The encouraging result is that safety analysis appears to be feasible for such models. It is also encouraging that amplification can actually be built out of non-amplifying primitives since this will simplify implementations.

Our examples demonstrate that fairly complicated policies arise in even rather simple situations. The examples have used a few types of subjects and objects. Realistically in large organizations we would have hundreds of types. The complexity will rapidly multiply. We believe that authorization policies will necessarily be formulated in terms of local and incremental considerations of the kind we have discussed. Safety analysis becomes very important to ensure that the cumulative global effect of local incremental authorizations is consistent with our objectives.

Looking towards the future it is clear that much work remains to be done to produce a truly general and powerful model for access-control. It is not enough to develop a model. We must also develop methodologies for specifying policies in the model. And of course the model must be implementable in terms of a small number of primitive mechanisms.

References

[1] Clark, D.D. and Wilson, D.R. "A Comparison of Commercial and Military Computer Security Policies." *IEEE Symposium on Security and Privacy*, 184-194 (1987).

[2] Cohen, E. and Jefferson, D. "Protection in the Hydra Operating System." *5th ACM Symposium on Operating Systems Principles*, 141-160 (1975).

[3] Dennis, J.B. and Van Horn, F.C. "Programming Semantics for Multiprogrammed Computations." *Communications of ACM* 9(3):143-155 (1966).

[4] Graham, G.S. and Denning, P.J. "Protection - Principles and Practice." *AFIPS Spring Joint Computer Conference* 40:417-429 (1972).

[5] Harkness, W. and Pittelli, P.A. "Command Authorization as a Component of Information Integrity." *Computer Security Foundations Workshop*, 219-226 (1988).

[6] Harrison, M.H., Ruzzo, W.L. and Ullman, J.D. "Protection in Operating Systems." *Communications of ACM* 19(8):461-471 (1976).

[7] Harrison, M.H. and Ruzzo, W.L. "Monotonic Protection Systems." In DeMillo, R.A., Dobkin, D.P., Jones, A.K. and Lipton, R.J. (Editors). *Foundations of Secure Computations.* Academic Press (1978).

[8] Lampson, B.W. "Protection." *5th Princeton Symposium on Information Science and Systems*, 437-443 (1971). Reprinted in *ACM Operating Systems Review* 8(1):18-24 (1974).

[9] Linden, T.A. "Operating System Structures to Support Security and Reliable Software." *ACM Computing Surveys* 8(4):409-445 (1976).

[10] Lipton, R.J. and Snyder, L. "A Linear Time Algorithm for Deciding Subject Security." *Journal of ACM* 24(3):455-464 (1977).

[11] Lockman, A. and Minsky, N. "Unidirectional Transport of Rights and Take-Grant Control." *IEEE Transactions on Software Engineering* SE-8(6):597-604 (1982).

[12] Minsky, N. "The Principle of Attenuation of Privileges and its Ramifications." In DeMillo, R.A., Dobkin, D.P., Jones, A.K. and Lipton, R.J. (Editors). *Foundations of Secure Computations.* Academic Press (1978).

[13] Minsky, N. "Synergistic Authorization in Database Systems." *7th International Conference on Very Large Data Bases*, 543-552 (1981).

[14] Minsky, N. "Selective and Locally Controlled Transport of Privileges." *ACM Transactions on Programming Languages and Systems* 6(4):573-602 (1984).

[15] Moffett, J.D. and Sloman, M.S. "The Source of Authority for Commercial Access Control." *IEEE Computer* 21(2):59-69 (1988).

[16] Saltzer, J.H. and Schroeder, M.D. "The Protection of Information in Computer Systems." *Proceedings of IEEE* 63(9):1278-1308 (1975).

[17] Sandhu, R.S. and Share, M.E. "Some Owner Based Schemes with Dynamic Groups in the Schematic Protection Model." *IEEE Symposium on Security and Privacy*, 61-70 (1986).

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

[19] Sandhu, R.S. "Expressive Power of the Schematic Protection Model." *Computer Security Foundations Workshop*, 188-193 (1988).

[20] Sandhu, R.S. "Transaction Control Expressions for Separation of Duties." *4th Aerospace Computer Security Applications Conference*, 282-286 (1988).

[21] Report of the Invitational Workshop on Integrity Policy in Computer Information Systems (WIPCIS), Bentley College, MA, October 1987.

[22] Wulf, W., Cohen, E., Corwin, W., Jones, A., Levin, R., Pierson, C. and Pollack, F. "Hydra: The Kernel of a Multiprocessor Operating System." *Communications of ACM* 17(6):337-345 (1974).