Proc. IEEE Symposium on Research in Security and Privacy, Oakland, California, May 16-18, 1994, pages 230-241

On the Minimality of Testing for Rights in Transformation Models

*Ravi S. Sandhu and Srinivas Ganta**
Center for Secure Information Systems
&
Department of Information and
Software Systems Engineering
George Mason University
Fairfax, VA 22030-4444
{sandhu,gsriniva}@isse.gmu.edu

Abstract

In this paper we define and analyze a family of access control models, called transformation models, which are based on the concept of *transformation of rights*. In these models, propagation of access rights is authorized entirely by existing rights for the object in question. Transformation models are useful for expressing various kinds of consistency, confidentiality, and integrity controls. These models also generalize the monotonic transform model of Sandhu, and its non-monotonic extension (NMT) by Sandhu and Suri. We argue that NMT is inadequate for expressing the document release example discussed by Sandhu and Suri, because it can test only one access matrix cell in its state changing commands. We then analyze the relative expressive power of testing two access matrix cells in state changing commands versus testing more than two. Our conclusion is that it suffices to allow testing for two cells.

1 Introduction

In this paper we define and analyze a family of access control models called Transformation Models. These models are based on the concept of *transformation of rights*, which simply implies that possession of rights for an object by subjects allows those subjects to get and lose rights for that object and also give and revoke rights (for that object) to other subjects. Hence, in these models propagation of access rights is authorized entirely by existing rights for the object in question. (More generally, propagation could also be authorized by existing rights for the source and destination subjects, for example, in models such as HRU [5], SPM [11], TAM [13].)

The concept of transformation of rights allows us to express a large variety of practical security policies encompassing various kinds of consistency, confidentiality and integrity controls. In this paper, we demonstrate the expressiveness of Transformation Models, by expressing some practical policies using these models and citing other examples published elsewhere in the literature.

The concept of transformation of access rights was introduced by Sandhu in [12]. Based on it the Monotonic Transform Model and its Non-Monotonic extension (NMT) [14] were proposed. The simplicity and expressive power of NMT is demonstrated in [14] by means of a number of examples. But we have discovered that NMT cannot adequately implement the document release example given in [14]. The reason behind this is the limited testing power of NMT. This led us to the formulation of the Transformation Model (TRM) introduced in this paper. TRM substantially generalizes NMT.

In TRM the propagation of access rights is authorized entirely by existing rights for the object in question. TRM can therefore be very efficiently implemented in a distributed environment using a simple client-server architecture. In such an implementation, each server acts as a mediator for the set of objects it manages. All accesses to an object pass through its server who determines the validity of the request. The server maintains an access control list (ACL) for

*The work of both authors is partially supported by National Science Foundation grant CCR-9202270 and by the National Security Agency contract MDA904-92-C-5141. We are grateful to Nathaniel Macon, Howard Stainer, and Mike Ware for their support and encouragement in making this work possible.

each of the objects it is required to manage. An ACL is associated with each object, specifying the subjects who can access the object and the access right(s) authorized for each of them. The ACL makes the access to the object dependent on the identity of the subject. Every time a subject makes a request, the request is checked against this list. The access request is valid only if the access requested is authorized by the rights present for the subject in the ACL for the object. ACL's are dynamic and hence to maintain them each server only needs to know about the state changing commands for the object type that it manages. Moreover, the authorization can be checked locally at each server without requiring communication with other servers. This makes the system extremely modular and easy to maintain. In particular, new object types can be introduced (along with their servers) without any need to inform previously existing servers.

We also define two special cases of TRM called the Unary Transformation Model (UTRM) and the Binary Transformation Model (BTRM). UTRM commands are authorized by checking for rights in a single cell of the access matrix, whereas in BTRM commands such testing is limited to exactly two cells. We then analyze the relative expressive power of these models. One of our conclusions is that UTRM is not adequately expressive. On the other hand, we formally prove that BTRM is as expressive as TRM in general (where authorization for commands can be checked by testing any number of cells). Hence our conclusion is that it suffices to have models which test for two cells.

Although the syntax of Typed Access Matrix Model (TAM) [13] and the transformation models proposed in this paper are alike, these models differ in some important respects. The main difference is that in TRM propagation of access rights is authorized entirely by existing rights for the object in question, whereas in TAM this testing may involve rights for other objects (and subjects) in the system. Also TRM allows testing for absence of rights and TAM does not allow testing for absence of rights (although in extensions of TAM such tests have been incorporated [1, 15]). TAM extended to allow testing for absence of rights is a generalization of TRM.

The rest of the paper is organized as follows. Section 2 defines the Transformation Model (TRM). It also describes two models, UTRM and BTRM, which are defined as restricted cases of TRM. In section 3, we show how these models can enforce some practical policies. In section 4, we argue that UTRM, which tests for one cell, cannot conveniently express some simple policies. Section 5 proves that BTRM, which tests for two cells, is as expressive as TRM in general. Finally, section 6 concludes the paper.

2 The Transformation Model (TRM)

In this section we formally define and intuitively motivate the Transformation Model. TRM is an access control model in which authorization for propagation of access rights is entirely based on existing rights for the object in question. As argued in the introduction, this leads to an efficient implementation in a distributed environment using a simple client-server architecture.

The protection state in TRM can be viewed in terms of the familiar access matrix. There is a row for each subject in the system and a column for each object. In TRM, the subjects and objects are disjoint. TRM does not define any access rights for operations on subjects, which are assumed to be completely autonomous entities. The $[X, Y]$ cell contains rights which subject X possesses for object Y.

TRM consists of a small number of basic constructs and a language for specifying the commands which cause changes in the protection state. For each command, we have to specify the authorization required to execute that command, as well as the effect of the command on the protection state. We generally call such a specification as an *authorization scheme* (or simply scheme) [13].

A scheme in the TRM is defined by specifying the following components.

1. A set of access rights R.
2. Disjoint sets of subject and object types, TS and TO, respectively.
3. A collection of three classes of state changing commands: *transformation commands, create commands*, and *destroy commands*. Each individual command specifies the authorization for its execution, and the changes in the protection state effected by it.

The scheme is defined by the security administrator when the system is first set up and thereafter remains fixed. It should be kept in mind that TRM treats the security administrator as an external entity, rather than as another subject in the system. Each component of the scheme is discussed in turn below.

As explained in the introduction, TAM [13] and TRM are strongly related. They differ in state changing commands. In TRM, propagation of access rights is authorized entirely by existing rights for the object in question, whereas in TAM this authorization can involve testing rights for multiple objects. TRM does allow testing for absence of rights, while the original definition of TAM [13] does not allow for such testing. If TAM is augmented with testing for absence of rights (as in [1], [15]), it is then a generalization of TRM.

2.1 Rights

Each system has a set of rights, R. R is not specified in the model but varies from system to system. We will generally expect R to include the usual rights such as *own, read, write, append* and *execute*. However, this is not required by the model. We also expect R to generally include more complex rights, such as *review, pat-ok, grade-it, release, credit, debit*, etc. The meaning of these rights will be explained wherever they are used in our examples.

The access rights serve two purposes. First, the presence of a right, such as r, in the $[S, O]$ cell of the access matrix may authorize S to perform, say, the read operation on O. Secondly, the presence of a right, say o, or the absence of right o, in $[S, O]$ may authorize S to perform some operation which changes the access matrix, e.g., by entering r in $[S', O]$. The focus of TRM is on this second purpose of rights, i.e., the authorization by which the access matrix itself gets changed.

2.2 Types of Subjects and Objects

The notion of type is fundamental to TRM. All subjects and objects are assumed to be strongly typed. Strong typing requires that each subject or object is created to be of a particular type which thereafter does not change. The advantage of strong typing is that it groups together subjects and objects into classes (i.e., types) so that instances of the same type have the same properties with respect to the authorization scheme.

Strong typing is analogous to tranquility in the Bell-LaPadula style of security models [2], whereby security labels on subjects and objects cannot be changed. The adverse consequences of unrestrained non-tranquility are well known [4, 7, 8]. Similarly, non-tranquility with respect to types has adverse consequences for the safety problem [13].

TRM requires that a disjoint set of subject types, TS, and object types, TO, be specified in a scheme. For example, we might have TS={*user, security-officer*} and TO={*user-files, system-files*}, with the significance of these types indicated by their names.

2.3 State Changing Commands

The protection state of the system is changed by means of TRM commands. The security administrator defines a finite set of commands when the system is specified. There are three types of state changing commands in the TRM, each of which is defined below.

2.3.1 Transformation Commands

We reiterate that every command in TRM has a condition which is on a single object and the primitive operations comprising the command are only on that object. In all the commands the last parameter in the command is the object which is being manipulated, and the first parameter is the subject who initiates the command.

A *transformation command* has the following format:

command $\alpha(S_1 : s_1,\ S_2 : s_2,\ \ldots,\ S_k : s_k,\ O : o)$
  if *predicate* **then**
    $op_1;\ op_2;\ \ldots;\ op_n$
end

The first line of the command states that α is the name of the command and $S_1, S_2, \ldots, S_k, O$ are the formal parameters. The formal parameters S_1, S_2, $\ldots$, S_k are subjects of types s_1, s_2, $\ldots$, s_k, respectively. The **only** object formal parameter O is of type o and is the last parameter in the command.

The second line of the command α is the predicate and is called the *condition* of the command. The predicate consists of a boolean expression composed of the following terms connected by the usual boolean operators (such as $\wedge$ and $\vee$):

$$r_i \in [S, O] \text{ or } r_i \notin [S, O]$$

Here r_i is a right in R, S can be substituted with any of the formal subject parameters $S_1, S_2, \ldots, S_k$, and O is the sole object parameter. Simply speaking the predicate tests for the presence and absence of some rights for subjects on object O. Given below are some examples of TRM predicates:

1. $approve \in [S_1, O] \wedge$ $prepare \notin [S_2, O]$
2. $prepare \in [S, O] \wedge$ $assign \in [S_1, O] \wedge$ $creator \notin [S, O]$
3. $own \in [S, O] \vee$ $write \in [S, O]$
4. $r_1 \in [S_1, O] \wedge (r_2 \in [S_1, O] \vee r_1 \in [S_2, O]) \wedge r_3 \in [S_2, O] \wedge r \in [S_3, O]$

If the condition is omitted, the command is said to be an *unconditional command*, otherwise it is said to be a *conditional command.*

The third line of the command consisting of sequence of operations op_1; op_2; $\ldots$; op_n is called the *body* of α. Each op_i is one of the following two primitive operations:

- **enter** r **into** $[S, O]$
- **delete** r **from** $[S, O]$

Here again, r is a right in R, S can be substituted with any of the formal subject parameters $S_1, S_2, \ldots, S_k$ and O is the sole object parameter. It is important to note that all the operations enter or delete rights for subjects on object O alone.

The **enter** operation enters a right $r \in$ R into an existing cell of the access matrix. The contents of the cell are treated as a set for this purpose, i.e., if the right is already present, the cell is not changed. The **delete** operation has the opposite effect of **enter**. It (possibly) removes a right from a cell of the access matrix. Since each cell is treated as a set, **delete** has no effect if the deleted right does not already exist in the cell.

A TRM command is invoked by substituting actual parameters of the appropriate types for the formal parameters. The condition part of the command is evaluated with respect to its actual parameters. The body is executed only if the condition evaluates to true.

Some examples of *transformation commands* are given below.

command *transfer-ownership* $(S_1 : s, S_2 : s, O : o)$
  if $own \in [S_1, O]$ **then**
  enter *own* in $[S_2, O]$
  delete *own* from $[S_1, O]$
end

command *grade* $(S_1 : professor, S_2 : student,$
  $O : project)$
  if $own \in [S_2, O] \wedge grade \in [S_1, O]$ **then**
  enter *good* in $[S_2, O]$
  delete *grade* from $[S_1, O]$
end

command *issue-check* $(S_1 : clerk, O : voucher)$
  if $prepare \notin [S_1, O] \wedge approve \notin [S_1, O]$ **then**
  enter *issue* in $[S_1, O]$
end

The command *transfer-ownership* transfers the ownership of a file from one subject to another. In the command *grade*, the professor gives right *good* to the student's project after the student who owns the project has requested the professor to grade it. In command *issue-check*, a clerk gets an *issue* right for a check only if he/she is not the one who prepared and approved it.

2.3.2 Create Commands

A *create command* is an unconditional command. The creator of an object gets some rights for the created object like *own, read*, etc., as specified in the body of the command. No subject other than the creator will get rights to the created object in the create command. Subjects other than the creator can subsequently acquire rights for the object via transformation commands. In short, the effect of a create command is to introduce a new column in the matrix with some new rights for the subject who created it.

A typical *create command* is given below.

command *create* $(S_1 : s_1,\ O : o)$
  create object O
  enter *own* in $[S_1, O]$
end

In the general case the body of the command may enter any set of rights in the $[S_1, O]$ cell.

A create command is necessarily an unconditional command as the command cannot check for rights on an object which does not exist, and TRM commands do not allow testing for rights on objects other than the object which is being created. The create object operation requires that the object being created have an unique identity different from all other objects.

2.3.3 Destroy Commands

A *destroy command* is, in general, a conditional command. The effect of a destroy command on the matrix will be removal of the corresponding column from the access matrix. A typical destroy command is given below.

command *destroy* $(S_1 : s_1,\ O : o)$
  if $own \in [S_1, O]$ **then**
  destroy object O
end

In this case the condition ensures that only the owner can destroy the object. More generally, deletion can be authorized by some combination of rights possessed by the destroyer.

2.4 Summary of TRM

To summarize, a system is specified in TRM by defining the following finite components.

1. A set of rights R.
2. A set of disjoint subject and object types TS and TO respectively.
3. A set of state-changing transformation, creation and destroy commands.
4. The initial state.

We say that the rights, types and commands define the system *scheme*. Note that once the system scheme is specified by the security administrator it remains fixed thereafter for the life of the system. The system state, however, changes with time.

2.5 The Unary Transformation Model (UTRM)

The Unary Transformation Model is a simpler version of TRM in which testing in a command can be on only one cell of the matrix. A UTRM predicate consists of a boolean expression composed of the following terms:

$$r_i \in [S_j, O] \text{ or } r_i \notin [S_j, O]$$

where r_i is a right in R and S_j can be any one of the formal subject parameters, but all the terms in the expression must have the same S_j. In other words, the predicate tests for the presence and absence of rights for a single subject S_j on object O. Usually S_j will be the first parameter in the command, since that is the one who initiates the command.

UTRM generalizes the model called NMT (for Non-Monotonic Transform) [14]. The transformation commands in NMT, viz., grant transformation and internal transformation, are easily expressed as UTRM commands (as they test for rights in one cell of the matrix). NMT is a restricted version of UTRM as the state changing commands in NMT test only one cell and modify at most two cells.

As we will argue in section 4, UTRM, and therefore NMT, cannot adequately express some simple policies of practical interest.

2.6 The Binary Transformation Model (BTRM)

The Binary Transformation Model is also a simpler version of TRM in which testing in a command can involve up to two cells of the matrix. A BTRM predicate consists of a boolean expression composed of the following terms:

$$r_i \in [S_j, O] \text{ or } r_i \notin [S_j, O]$$

where r_i is a right in R and S_j can be any one of the formal subject parameters, but the expression can have at most two different S_j's from the given parameters. In other words, the predicate tests for the presence and absence of rights for at most two subjects (on object O). One of the S_j's will typically be the first parameter which is the initiator of the command.

Sections 4 and 5 will discuss the relationship between the expressive power of TRM, UTRM, and BTRM.

3 Examples

In this section we motivate the utility of TRM, UTRM and BTRM. It has been shown in [12] that monotonic transformation of access rights can implement rights amplification [3], various kinds of copy flags [6] and synergistic authorization [9]. It has further been shown in [14], that NMT can express both the concepts of transfer-only privileges and countdown privileges. As these models are instances of UTRM, the above concepts can also be expressed by UTRM, BTRM and TRM.

In this section we take some other policies and see how they can be specified by TRM. In particular, we show how the Bell-LaPadula model [2], transaction control expressions [10], and a document release example [14] are implemented by these models. Bell-LaPadula is enforced by UTRM (and obviously can be enforced by BTRM and TRM). BTRM expresses both the document release and transaction control expression examples. The document release example does not need testing for absence of rights, whereas transaction control expressions do need testing for absence of rights.

3.1 BLP with Tranquility

The concept of mandatory access controls was introduced by Bell and LaPadula [2]. They defined a model, commonly called the Bell-LaPadula or the BLP model.

The key idea in BLP is to augment discretionary access controls with *mandatory access controls*, so as to enforce information flow policies. BLP takes a two step approach to access control. First there is a discretionary access matrix D, the contents of which can be modified by subjects (in some manner which we do not need to specify). However, authorization in D is not sufficient for an operation to be carried out. In addition, the operation must also be authorized by the mandatory access control policy, over which users have no control.

The mandatory access control policy is expressed in terms of security labels attached to subjects and objects. A label on an object is called a *security classification*, while a label on a subject is called a *security clearance*. Moreover, the classifications and clearances once assigned cannot be changed. This assumption is known as *tranquility*.

The specific mandatory access rules given in BLP are as follows (where λ signifies the security label of the indicated subject or object):

- *Simple-Security Property*: Subject s can read object o only if $\lambda(s) \geq \lambda(o)$.
- $\star$-*Property*: Subject s can write object o only if $\lambda(s) \leq \lambda(o)$.

We now see how the simple-security and $\star$-properties are implemented in UTRM (actually NMT).

The labels of subjects are the types of the subjects and labels on the objects are the types of the objects. For example, the type of an object whose label is λ_i is $O\lambda_i$, and the type of a subject whose label is λ_i is $S\lambda_i$.

The simple-security property is enforced by giving commands for each λ_i and λ_j, such that $S\lambda_i \geq O\lambda_j$ as shown below. Let the right r^- represent the discretionary permission for subject S to read object O. Let r be the right authorizing S to actually perform the read operation on O. The right r^- is entered in accordance with some discretionary which we do not specify (it does not matter what this discretionary policy is). If r^- exists in $[S, O]$, then r is entered into the matrix (by BLP) to really give the ability for S to read O.

> **command** *read-ij*$(S : S\lambda_i, O : O\lambda_j)$
> **if** $r^- \in [S, O]$ **then**
>   **enter** r in $[S, O]$
> **end**

There is a *read-ij* command for each λ_i and λ_j such that $S_i \geq O_j$. If $S\lambda_i \not\geq O\lambda_j$, there is no corresponding *read-ij* command, so in such cases r^- cannot be converted to r.

The $\star$-property is similarly enforced by giving commands for each λ_i and λ_j such that $S\lambda_i \leq O\lambda_j$ as shown below. In the command *write-ij* the right w^- (which gives the discretionary permission to write under some unspecified discretionary policy) is tested for presence in $[S, O]$. If w^- exists in $[S, O]$, then w is entered into the matrix to really give the ability for S to write on O.

> **command** *write-ij*$(S : S\lambda_i, O : O\lambda_j)$
> **if** $w^- \in [S, O]$ **then**
>   **enter** *write* in $[S, O]$
> **end**

In this case, if $S\lambda_i \not\leq O\lambda_j$, there is no *write-ij* command, so w^- cannot be converted to w. Create and Destroy are also constrained by the $\star$-property because they modify the state of the object. They are omitted here, but can be easily worked out by the interested reader.

It appears that BLP with non-tranquility cannot be enforced by TRM (although we do not have a formal proof). This is due to the fact that the label of a subject or an object is encoded as a type in the above construction. As typing is strong, it is not possible to change the type of a subject or an object. Other means of encoding a subject label, such as in a special column of the access matrix, requires testing in more than one column, contrary to TRM. Similarly encoding the label in one or more rows requires changing the contents of multiple columns when the subject's label is changed. This is not allowed in TRM.

3.2 Document Release Example

Next let us take the document release example discussed in [14]. In this section we show how this example can be implemented by BTRM. The next section argues informally why NMT and UTRM cannot adequately enforce this example. (The document-release solution given in [14] has a flaw and does not solve the problem correctly.)

In the document release problem, a scientist creates a document and hence gets *own, read* and *write* rights to it. After preparing the document for publication, the scientist asks for a review from a patent officer. In the process, the scientist loses the *write* right to the document, since it is clearly undesirable for a document to be edited during or after a (successful) review. After review of the document, the patent officer grants the scientist an approval. It is reasonable to disallow further attempts to review the document after an approval is granted. Thus the *review* right for the document is lost as approval is granted. After obtaining approval from the patent officer, the scientist can publish the document by getting a *release* right for the document. (The problem discussed in [14] also requires approval by a security officer prior to document release, but that aspect of the problem is not germane to the discussion here.)

To express this policy, we employ the following rights and types:

- R = {*own, read, write, review, pat-ok, pat-reject, release*}
- TS = {*sci, po*}, TO = {*doc*}

The *own, read,* and *write* rights have their usual meaning. The other rights correspond to stages in the approval process. The right *review* lets a patent officer review a document; *pat-ok* is the right that is returned if the patent review is satisfactory otherwise *pat-reject* is returned; and *release* authorizes release of the document. Subject types *sci* and *po* are abbreviations for scientists and patent officers respectively, and there is a single object type *doc*.

The following TRM (or more precisely BTRM) commands enforce the desired policy:

command *create-doc*$(S : sci, O : doc)$
  create object O
  enter *own* in $[S, O]$
  enter *read* in $[S, O]$
  enter *write* in $[S, O]$
end

command *rqst-review*$(S : sci, P : po, O : doc)$
if *own* $\in [S, O] \wedge$ *write* $\in [S, O]$ **then**
  enter *review* in $[P, O]$
  delete *write* from $[S, O]$
end

command *get-approval*$(S : sci, P : po, O : doc)$
if *review* $\in [P, O] \wedge$ *own* $\in [S, O]$ **then**
  enter *pat-ok* in $[S, O]$
  delete *review* from $[P, O]$
end

command *get-rejection*$(S : sci, P : po, O : doc)$
if *review* $\in [P, O] \wedge$ *own* $\in [S, O]$ **then**
  enter *pat-reject* in $[S, O]$
  delete *review* from $[P, O]$
end

command *release-doc*$(S : sci, O : doc)$
if *pat-ok* $\in [S, O]$ **then**
  enter *release* in $[S, O]$
  delete *pat-ok* from $[S, O]$
end

command *revise-doc*$(S : sci, O : doc)$
if *pat-reject* $\in [S, O]$ **then**
  enter *write* in $[S, O]$
  delete *pat-reject* from $[S, O]$
end

The scientist creates a document using the command *create-doc*. After preparing the document the scientist asks the patent officer to review it through command *rqst-review*. The scientist gets approval to release through command *get-approval* or a rejection via *get-rejection*. In the former case the scientist gets the *release* permission by means of the command *release-doc*. In the latter case the scientist gets the *write* permission by means of the command *revise-doc* so as to revise the document if appropriate.

3.3 Transaction Control Expressions

We show how one of the examples of the transaction control expressions given in [10] can be expressed in BTRM. A transaction control expression (TCE) represents the potential history of an information object. The classic example of a transient object is a voucher that ultimately results in a check being issued. The potential history of a voucher is represented by the following transaction control expression [10].

prepare • clerk;
issue • clerk;

Each *term* in this expression has two parts. The first part names a transaction. The transaction can be executed only by a user with the *role* specified in the second part. For simplicity in discussion assume each user has only one role. So 'prepare • clerk' specifies that the prepare transaction can be executed on a voucher only by a clerk. The semi-colon signifies sequential application of the terms. That is, a clerk can execute the issue transaction on a voucher only after a clerk has executed the preceding prepare transaction. Finally, separation of duties is specified by requiring that the users who execute different transactions in the transaction control expression all be distinct.

We now show how the given TCE is specified in BTRM. We make use of the following sets of types and rights:

1. Rights R= $\{prepare, prepare', issue, issue'\}$
2. Subject types TS= $\{clerk\}$, and object types TO= $\{voucher\}$

Rights are used as a means of keeping track of the current location in the progression of a transaction control expression. *Undecorated rights*, i.e., those rights without a trailing apostrophe, are used to indicate that current operation in the transaction control expression is in progress. *Decorated rights*, i.e., those rights with a trailing apostrophe, are used to indicate that current operation in the transaction control expression is complete. The decorated rights are useful in ensuring both separation and coincidence of duties.

The BTRM commands for the voucher transaction control expression are given below. Each step of the TCE is translated into two commands: the first indicating that the step in question is in progress, and the second indicating that the step has been completed.

(a) **command** *begin-prepare-voucher* (C : *clerk*, V : *voucher*)
  create subject V
  enter *prepare* **into** $[C, V]$
end

(a$'$) **command** *complete-prepare-voucher* (C : *clerk*, V : *voucher*)
 if *prepare* $\in [C, V]$ **then**
  delete *prepare* **from** $[C, V]$
  enter *prepare$'$* **into** $[C, V]$
end

(b) **command** *begin-issue-check* (C_1 : *clerk*, C_2 : *clerk*, V : *voucher*)
 if *prepare$'$*$\in [C_2, V] \wedge$ *prepare$'$*$\notin [C_1, V]$ **then**
  delete *prepare$'$* **from** $[C_2, V]$
  enter *issue* **into** $[C_1, V]$
end

(b$'$) **command** *complete-issue-check* (C : *clerk*, V : *voucher*)
 if *issue*$\in [C, V]$ **then**
  delete *issue* **from** $[C, V]$
  enter *issue$'$* **into** $[C, V]$
end

To control progress of the TCE, the clerk in command (a) creates a voucher object and acquires the undecorated right *prepare*, indicating that the first operation of the TCE is in progress. Once the voucher has been prepared command (a$'$) is invoked to indicate, via the *prepare$'$* right, that voucher preparation is complete. Command (a$'$) can be invoked only by the same clerk who invoked command (a) for a given voucher. Command (a$'$) enters the *prepare$'$* right in the $[C, V]$ cell to record which clerk prepared the voucher. The command (b) gives the named clerk the *issue* right for the voucher, provided the voucher has been prepared, and the specific clerk named in the command does *not* hold the *prepare$'$* right for the voucher. (This is where the facility to test for absence of rights is crucial.) Command (b$'$) subsequently indicates, via the *issue$'$* right, that the check has been issued. At this point the voucher's TCE is complete and the voucher can be archived. (The BTRM command for archival has been omitted for simplicity.)

4 Expressive Power of UTRM

We now analyze the expressive power of TRM and its variations. We first argue that UTRM is not sufficiently adequate to express the document release example of the previous section. In the next section we then show that BTRM is equivalent to TRM in terms of expressive power. Hence BTRM can enforce the document release example, and all the policies enforced by TRM.

Recall that UTRM is a restricted version of TRM. It is the same as TRM except that the testing in a command can only be on a single cell. Our conclusion in this section is that it is difficult to conveniently enforce the document release example in UTRM.

Consider the document release example given in the previous section. All the commands, except *get-approval* and *get-rejection* shown below, are UTRM commands. These two commands are BTRM command as they test two cells.

command *get-approval*(S : *sci*, P : *po*, O : *doc*)
if *review* $\in [P, O] \wedge$ *own* $\in [S, O]$ **then**
  enter *pat-ok* in $[S, O]$
  delete *review* from $[P, O]$
end

command *get-rejection*(S : *sci*, P : *po*, O : *doc*)
if *review* $\in [P, O] \wedge$ *own* $\in [S, O]$ **then**
  enter *pat-reject* in $[S, O]$
  delete *review* from $[P, O]$
end

The *get-approval* command tests for rights in two cells of the matrix. More specifically, it tests if the patent officer has the *review* right for the document and if the scientist is the owner of the document. If this condition is satisfied the command gives the right, *pat-ok*, to the owner.

If the *get-approval* command does not test for the *own* right, then the command might give the *pat-ok* right to some other scientist who is not a owner. The system will then halt in an unwanted state as the scientist who creates the document cannot get the *release* right for it. This is due to the fact that the scientist cannot request a second review prior to receiving a response for the first one (this is achieved by conditioning the request for review on presence of the *write* right, which is then removed until a rejection is received). At the same time, the patent officer can give the *pat-ok* only once to one scientist (as the patent officer loses the *review* right in this process). Therefore if the patent officer gives the right *pat-ok* to a scientist who is not owner, the actual owner cannot get the *release* right and the system halts in an unwanted state.

If the *get-approval* command does not test for the *review* right then a patent officer can grant *pat-ok* for documents which the scientist can still write. Moreover, this can be done whether or not a request for review has been made. The danger of this approach is obvious. But then the required policy cannot be conveniently enforced by UTRM. Note that similar considerations apply to the *get-rejection* command.

In short, to enforce the document release example, it appears there is a need for commands which tests for two cells of the matrix. Since UTRM (and NMT lack) such commands, they cannot conveniently express the document release example.

The foregoing discussion argues informally that UTRM is inadequate for the document release example. In our further work [16], we have actually proved the theoretical equivalence of UTRM and TRM, which indicates that UTRM has the expressive power to enforce this example. However, our construction used in proving the theoretical equivalence of UTRM and TRM requires commands with all subjects as parameters, and is not practically viable.

5 Expressive Power of BTRM

In this section, we establish the relationship between the expressive power of TRM and BTRM. Recall that BTRM is a restricted version of TRM. Hence to prove equivalence, we need to show that for every TRM scheme, there exists an equivalent BTRM scheme. We will show how any given TRM command can be simulated by multiple BTRM commands.

The Boolean condition of any TRM command, say Y, can be converted into the familiar *disjunctive normal form* which consists of a disjunction (i.e., $\vee$) of *minterms*. Each minterm is a conjunction (i.e., $\wedge$) of primitive terms of the form $r_i \in [S_i, O]$ or $r_i \notin [S_i, O]$. The command Y can then be factored into multiple commands, each of which has one minterm as its condition and the original body of Y as its body.

Therefore, to show how an arbitrary TRM command can be simulated, it is enough to consider a TRM command which has the format of the command X given below.

command $X\,(S_1 : s_1, S_2 : s_2, \ldots, S_n : s_n, O : o)$
if $P_1 \wedge P_2 \ldots \wedge P_n$ **then**
  operations in $[S_1, O]$
  operations in $[S_2, O]$
  ...
  operations in $[S_n, O]$
end

In the above command, each P_i is itself composed of terms $r_j \in [S_i, O]$ or $r_j \notin [S_i, O]$, where $r_j \in$ R. Intuitively P_i tests for the presence of, and absence of some rights in the *single* cell $[S_i, O]$. Some P_i's in the condition may be missing (or empty), in which case they are equivalent to the logical constant **true**. In the body of command X, the phrase "**operations** in $[S_i, O]$" denotes a sequence of enter, delete, or no operations in the $[S_i, O]$ cell. Note that the types $s_1, s_2, \ldots, s_n$ need not all be distinct. The formal parameters $S_1, S_2, \ldots, S_n$ must of course be distinct, but the actual parameters used on a particular invocation of this command may have repeated parameters as allowed by parameter types. (These are the usual conventions in most programming languages.)

We now consider how the TRM command X can be simulated by several BTRM commands. As X tests multiple cells, it is obvious that the simulation of X cannot be done by a single BTRM command. Since BTRM can test for only two cells, the simulation of X can be done by multiple commands in the BTRM system. The key to doing this successfully is to prevent other BTRM commands from interfering with the simulation of the given TRM command, X. The simplest way to do this is to ensure that TRM commands can be executed in the BTRM simulation only one at a time. To do this we need to synchronize the execution of successive TRM commands in the BTRM simulation.

This synchronization is achieved by introducing an extra subject called $LOCK$ of type *lock*, and an extra right, L. The role of $LOCK$ is to sequentialize the execution of simulation of TRM commands in the BTRM system. The type *lock* is assumed, without loss of generality, to be distinct from any type in the given TRM system. Thus the initial state of the BTRM system consists of the initial state of the given TRM system augmented with a subject $LOCK$ and with the right L, in all cells of the row represented by $LOCK$. (For objects created subsequently, the L right can be entered by a UTRM command invoked immediately after the creation.)

The BTRM simulation of X proceeds in five phases, as illustrated in figures 1 and 2. In these figures we show only the relevant portion of the access matrix,

	O
$LOCK$	X
S_1	1
S_2	2
$\ldots$	
S_n	n

(a) End of phase I

	O
$LOCK$	$X, T_1, T_2 \ldots, T_n$
S_1	1
S_2	2
$\ldots$	
S_n	n

(b) End of phase II

	O
$LOCK$	TX
S_1	1
S_2	2
$\ldots$	
S_n	n

(c) End of phase III

	O
$LOCK$	$TX, 1, 2, \ldots, n$
S_1	*operations*
S_2	*operations*
$\ldots$	
S_n	*operations*

(d) End of phase IV

Figure 1: BTRM simulation of the authorized TRM command X

	O
$LOCK$	FX
S_1	1
S_2	2
$\ldots$	
S_n	n

(a) End of phase III

	O
$LOCK$	$FX, 1, 2, \ldots, n$
S_1	
S_2	
S_n	

(b) End of phase IV

Figure 2: BTRM simulation of unauthorized TRM command X

and only those rights introduced specifically for the BTRM simulation. Since the focus is on a single object the matrix actually reduces to a single column for that object.

The first phase is to make sure that no other BTRM command corresponding to another TRM command can execute on object O until the simulation of X is complete. The second phase is where each predicate P_i is tested, and in the third phase the conjunction of the P_i's is tested. The fourth phase is where the body of X is executed (provided the third phase evaluates to **true**). The fifth and final phase makes sure that all the bookkeeping rights used in the simulation are removed from $[LOCK, O]$, and also enters the right L back in $[LOCK, O]$ to indicate that the simulation of another TRM command can now begin. Each of the phases and the commands used are explained briefly below.

The BTRM command *I-X-invocation* corresponds to phase I. It deletes the right L from $[LOCK, O]$, to make sure that no other BTRM command (simulating some other TRM command) can execute on object O until the simulation of X is complete. It ensures that the actual parameters of X are used in the simulation by entering rights $1, 2 \ldots, n$ in cells $[S_1, O], [S_2, O], \ldots, [S_n, O]$ respectively. It also enters the right X in $[LOCK, O]$ to indicate that the simulation of the command is currently in progress. The matrix, after the execution of command *I-X-invocation* resembles figure 1(a). The Phase I BTRM command is given below.

> **command** *I-X-invocation* $(S_1 : s_1, S_2 : s_2, \ldots,$
> $S_n : s_n, LOCK : lock, O : o)$
> **if** $L \in [LOCK, O]$ **then**
>   **enter** 1 in $[S_1, O]$
>   **enter** 2 in $[S_2, O]$
> $\ldots$
>   **enter** n in $[S_n, O]$
>   **enter** X in $[LOCK, O]$
>   **delete** L from $[LOCK, O]$
> **end**

In phase II, each command in *II-i-X-success* tests if the P_i part of the condition of X is true. If so, the command enters a right T_i in $[LOCK, O]$ to indicate that P_i is true. In phase II, if all the commands successfully test for their predicate, then the matrix after phase II will resemble figure 1(b). Similarly each command in *II-i-X-failure* tests if P_i is false, and if so, enters the right F_i in $[LOCK, O]$ to indicate that P_i is false. Phase II commands are given below. There is a different *II-i-X-success* and *II-i-X-failure* command for each value of $i = 1 \ldots n$.

> **command** *II-i-X-success* $(LOCK : lock, S_i :$
> $s_i, O : o)$
> **if** $X \in [LOCK, O] \wedge i \in [S_i, O] \wedge P_i$ **then**
>   **enter** T_i in $[LOCK, O]$
> **end**

> **command** *II-i-X-failure*$(LOCK : lock, S_i :$
> $s_i, O : o)$
> **if** $X \in [LOCK, O] \wedge i \in [S_i, O] \wedge \neg P_i$ **then**
>   **enter** F_i in $[LOCK, O]$
> **end**

Note that these are BTRM commands because testing P_i only involves tests in the $[S_i, O]$ cell.

In phase III, the command *III-X-complete-testing* tests if the condition of X is true by testing for all the rights $T_1, T_2 \ldots, T_n$ in $[LOCK, O]$. If they all exist, it then deletes all rights R[1] from $[LOCK, O]$ and enters right TX. TX is entered to indicate that the simulation of the body of X can now begin as the condition of X is true. The matrix after execution of *III-X-complete-success* resembles figure 1(c). Similarly in phase III, the command *III-X-complete-failure* tests if any one of $F_1, F_2, \ldots, F_n$ is in $[LOCK, O]$ and if so, it then deletes all rights R from $[LOCK, O]$ and enters right FX. FX is entered to indicate that the condition of X is false and hence the simulation of X can no longer continue. If command *III-X-complete-failure* executes in phase III, the matrix resembles figure 2(a). It is important to note that in phase III only one of *III-X-complete-success* or *III-X-complete-failure* can execute. Phase III commands are given below.

command *III-X-complete-testing* $(LOCK : lock, O : o)$
if $X \in [LOCK, O] \wedge T_1 \in [LOCK, O] \wedge T_2 \in [LOCK, O] \ldots \wedge T_n \in [LOCK, O]$ **then**
  delete R from $[LOCK, O]$
  enter TX in $[LOCK, O]$
end

command *III-X-complete-failure* $(LOCK : lock, O : o)$
if $X \in [LOCK, O] \wedge (F_1 \in [LOCK, O] \vee F_2 \in [LOCK, O] \ldots \vee F_n \in [LOCK, O])$ **then**
  delete R from $[LOCK, O]$
  enter FX in $[LOCK, O]$
end

In phase IV, each command in *IV-i-X-operations* checks for TX in $[LOCK, O]$, and if found performs operations in cell $[S_i, O]$. It also enters i in $[LOCK, O]$ to indicate that operations in $[S_i, O]$ have been done. If the testing of X is successful, the matrix at the end of phase IV resembles figure 1(d). Similarly, if the condition of X fails, then each command represented by *IV-i-X-garbage-removal* removes the right i in cell $[S_i, O]$ and it also enters the right i in $[LOCK, O]$ to indicate the removal of i. If the testing of X is a failure, the matrix at the end of phase IV resembles figure 2(b). The commands in phase IV are given below.

command *IV-i-X-operations* $(LOCK : lock, S_i : s_i, O : o)$
if $TX \in [LOCK, O] \wedge i \in [S_i, O]$ **then**
  delete i from $[S_i, O]$
  operations in $[S_i, O]$
  enter i in $[LOCK, O]$
end

command *IV-i-X-garbage-removal* $(LOCK : lock, S_i : s_i, O : o)$
if $FX \in [LOCK, O] \wedge i \in [S_i, O]$ **then**
  delete i from $[S_i, O]$
  enter i in $[LOCK, O]$
end

Finally, in phase V, command *V-X-done* checks to see if all operations of phase IV have been completed. This is done by checking for the presence of all rights $1, 2, \ldots, n$ in $[LOCK, O]$. If all the rights are in $[LOCK, O]$, this command clears the $[LOCK, O]$ cell and then enters right L in $[LOCK, O]$ indicating that the simulation of some other TRM command on object O can now begin. The Phase V command is given below.

command *V-X-done* $(LOCK : lock, O : o)$
if $1 \in [LOCK, O] \wedge 2 \in [LOCK, O] \ldots \wedge n \in [LOCK, O]$ **then**
  delete R from $[LOCK, O]$
  enter L in $[LOCK, O]$
end

In the TRM system, command X is initiated by S_1, the first parameter. In the BTRM simulation of X, the subject S_1 (which initiates X) initiates the only phase I command (*I-X-invocation*), and all other commands simulating X are initiated by $LOCK$ as a result. This is a reasonable assumption as $LOCK$ cannot initiate any command unless S_1 initiates *I-X-invocation*. This can be literally taken as S_1 authorizing $LOCK$ to initiate all other commands in the BTRM simulation.

In our construction all BTRM commands except the one in phase I have a maximum of three parameters. The command in phase I has more than three parameters. This command can be easily simulated by multiple BTRM commands, each of which have a maximum of three parameters. Hence we also conclude that TRM is equivalent to BTRM with only three parameters.

A proof sketch for the correctness of the construction is given below.

Theorem 1 *For every TRM system α_1, the construction outlined above produces an equivalent BTRM system α_2.*

Proof Sketch: It is easy to see that any reachable state in α_1 can be reached in α_2 by simulating each

[1] Strictly speaking, the command needs to remove $\{T_1, \ldots, T_n\}$ from $[LOCK, O]$. Since it does no harm to remove rights which are not present, for convenience we delete all rights R. This is similarly done with other commands in phases III and V.

TRM command by BTRM commands, as discussed above. Conversely any reachable state in α_2, with $L \in [LOCK, O]$, will correspond to a reachable state in α_1. A reachable state in α_2, with $L \notin [LOCK, O]$ and which passes the testing phase, will correspond to a state in α_1 where one TRM command has been partially completed. A state in α_2, with $L \notin [LOCK, O]$ and which fails the testing phase, will then lead α_2 to a previous state where $L \in [LOCK, O]$, which is reachable in α_1. Hence the above construction proves the equivalence of TRM and BTRM. A formal inductive proof can be given, but is omitted for lack of space.

6 Conclusion

Transformation models provide a powerful framework for implementing non-discretionary security policies in a simple client-server architecture. In this paper we have introduced this family of access control models, and have analyzed their expressive power. In particular, we have defined three models: the Transformation Model (TRM), the Unary-Transformation Model (UTRM), and the Binary-Transformation Model (BTRM). These models generalize the Monotonic Transform Model of Sandhu, and its Non-Monotonic extension (NMT). We have then shown the utility of these models by expressing some practical policies using them. We have argued that models which only test for a single cell cannot adequately implement simple policies like the document release example. We have then shown that testing for two cells has the same expressive power as testing for multiple cells. Hence, our conclusion is that in practical systems it suffices to have models which test for two cells.

References

[1] Ammann, P.E. and Sandhu, R.S. "Implementing Transaction Control Expressions by Checking for Absence of Access Rights." *Proc. Eighth Annual Computer Security Applications Conference*, San Antonio, Texas, 131-140, December 1992.

[2] Bell, D.E. and LaPadula, L.J. "Secure Computer Systems: Unified Exposition and Multics Interpretation." MTR-2997, Mitre, Bedford, Massachusetts (1975).

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

[4] Denning, D.E. "A Lattice Model of Secure Information Flow." *Communications of ACM* 19(5):236-243 (1976).

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

[6] 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).

[7] McLean, J. "A Comment on the 'Basic Security Theorem' of Bell and LaPadula." *Information Processing Letters* 20(2):67-70 (1985).

[8] McLean, J. "Specifying and Modeling Computer Security." *IEEE Computer* 23(1):9-16 (1990).

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

[10] Sandhu, R.S. "Transaction Control Expressions for Separation of Duties." *Proc. Fourth Aerospace Computer Security Applications Conference*, Orlando, Florida, December 1988, pages 282-286.

[11] Sandhu, R.S. "The Schematic Protection Model: its definition and analysis for acyclic attenuating schemes." *JACM.* 35,2,(April 1988). 404-432.

[12] Sandhu, R.S. "Transformation of Access Rights." *Proc. IEEE Symposium on Security and Privacy*, Oakland, California, May 1989, pages 259-268.

[13] Sandhu, R.S. "The Typed Access Matrix Model" *IEEE Symposium on Research in Security and Privacy*, Oakland, CA. 1992, pages 122-136.

[14] Sandhu, R.S. and Suri, G.S. "Non-monotonic Transformations of Access Rights." *Proc. IEEE Symposium on Research in Security and Privacy*, Oakland, California, May 1992, pages 148-161.

[15] Sandhu, R.S. and Srinivas Ganta. "On Testing for Absence of Rights in Access Control Models." *Proc. The Computer Security Foundations Workshop VI*, Franconia, NH, June 1993, pages 109-118.

[16] Sandhu, R.S. and Srinivas Ganta. "On the Expressive Power of the Unary Transformation Model." *GMU Technical Report ISSE-TR-94-101.*