APPLYING THE TAKE-GRANT PROTECTION MODEL

Matt Bishop

Technical Report PCS-TR90-151

Applying the Take-Grant Protection Model

Matt Bishop

Department of Mathematics and Computer Science
Dartmouth College
Hanover, NH 03755

ABSTRACT

The Take-Grant Protection Model has in the past been used to model multilevel security hierarchies and simple protection systems. The models are extended to include theft of rights and sharing of information, and additional security policies are examined. The analysis suggests that in some cases the basic rules of the Take-Grant Protection Model should be augmented to represent the policy properly; when appropriate, such modifications are made and their effects with respect to the policy and its Take-Grant representation are discussed.

Keywords: theoretical foundations, Take-Grant Protection Model, security policy, isolation, multilevel security.

1. Introduction

The Take-Grant Protection Model [12][15] is one of a number of abstract formulations of protection systems. Like other such models, it is primarily a theoretical tool used to analyze the safety question; unlike other models, it presents a framework in which that question is answered "yes," and enables the derivation of results indicating the conditions under which rights can be transferred as well as the complexity of determining whether or not those conditions hold in a particular system. Because of these characteristics, if it could be used to analyze an abstract model of an existing or proposed system, it would be possible either to prove in that model that rights could be transferred, or that they could not be.

In this, it differs from other formal models of protection. The *access control matrix model* [8][10][14] is a matrix-oriented description of the rights that *subjects*, or actors, have over *objects*, or passive entities. It is a very general model that does not constrain in any way the set of commands that may be used to manipulate rights. As one would expect, answering the safety question using such a system is quite complex, and for the most general case, it is undecidable. However, if the commands are mono-conditional (or monooperational), the safety question can be answered. The difference between this model and Take-Grant is that Take-Grant explicitly states four rules

Portions of this work were supported by a Burke Award from Dartmouth College.

controlling the propagation of rights, and four rules and information, and so — even though those rules are not mono-operational — the safety question can be answered.

The *schematic protection model* [17] is another model of protection which, like the access control matrix model, focuses on the issue of being able to answer the safety question and not determining necessary and sufficient conditions for sharing rights or information. Like the access control matrix model, it presumes no rules for propagation of rights, and ignores completely the questions of theft of rights and propagation of information.

Those questions have been studied extensively in the Take-Grant Protection Model. Work by Snyder [20] on the transfer of rights without the cooperation of the owner of those rights extended the results about sharing to situations of theft; work by Snyder and Bishop [4] introduced the notion of information flow and transfer into the model, and recent work by Bishop [3] presented results about the theft of information. Bishop [2], Jones [11], Snyder [18], and Wu [21] have applied the model to some existing computer systems and simple security policies. The contribution of this paper is the expansion of some of those representations to include theft of rights and the sharing of information as well as representing some new policies. We begin by briefly reviewing the Take-Grant Protection Model, and then derive a simplification of one of those results, which will make analysis of the security policies in the following section much simpler. We analyze the security policies of complete isolation, the owner having the ability to share rights nd information, a multilevel security policy, and a tree-based hierarchical policy such as is found in many computer systems. Finally, we conclude with some discussion and suggestions for future work.

2. The Basic Take-Grant Protection Model

We present this section as background into the model; the reader desiring a more complete presentation should turn to the references. On the other hand, the reader familiar with the model is encouraged to turn to the next section!

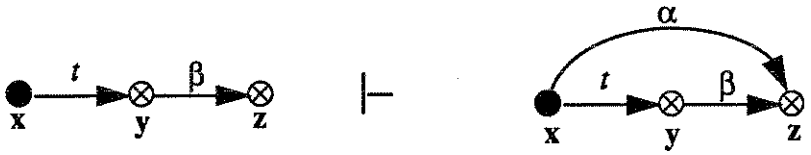
The Take-Grant Protection Model represents systems as graphs. The vertices are of two colors: *subjects* correspond to the active nodes such as processes, and *objects* correspond to the passive entities as files. (In modelling a system, representing an entity may not be straightforward; for example, a segment of memory might be an object when not in use, but a subject when being executed. We treat this as a point of interpretation and hence beyond the scope of this paper.) Edges connecting these nodes are labelled with sets of rights; these sets are subsets of a finite set of rights which for this description we shall take to be $R = \{ t, g, r, w \}$, where t represents the *take right*,

g the *grant right*, *r* the *read right*, and *w* the *write right*. Application of the rights is modelled by applying *graph rewriting rules*, which either change the protection state of the graph(*de jure* rules) or provide a representation of how information flows throughout the system (*de facto* rules).

In protection graphs, the subjects are represented by ● and objects by ○. Vertices which may be either subjects or objects are represented by ⊗. Pictures are very often used to show the effects of applying a graph rewriting rule on the graph; the symbol ⊢ is used to mean that the graph following it is produced by the action of a graph rewriting rule on the graph preceding it. The symbol ⊢* represents several rule applications. The term *witness* means a sequence of graph rewriting rules which produce the predicate or condition being witnessed, and a witness is often demonstrated by listing the graph rewriting rules that make up the witness (usually with pictures).

The *de jure* rules manipulate edges (and vertices) in the graph. They are:

take: Let **x**, **y**, and **z** be three distinct vertices in a protection graph G_0, and let **x** be a subject. Let there be an edge from **x** to **y** labelled γ with $t \in \gamma$, an edge from **y** to **z** labelled β, and $\alpha \subseteq \beta$. Then the *take* rule defines a new graph G_1 by adding an edge to the protection graph from **x** to **z** labelled α. Graphically,



The rule is written "**x** takes (α to **z**) from **y**."

grant: Let **x**, **y**, and **z** be three distinct vertices in a protection graph G_0, and let **x** be a subject. Let there be an edge from **x** to **y** labelled γ with $g \in \gamma$, an edge from **x** to **z** labelled β, and $\alpha \subseteq \beta$. Then the *grant* rule defines a new graph G_1 by adding an edge to the protection graph from **x** to **z** labelled α. Graphically,

The rule is written "**x** grants (α to **z**) to **y**."

create: Let x be any subject in a protection graph G_0 and let $\alpha \subseteq R$. *Create* defines a new graph G_1 by adding a new vertex y to the graph and an edge from x to y labelled α. Graphically,

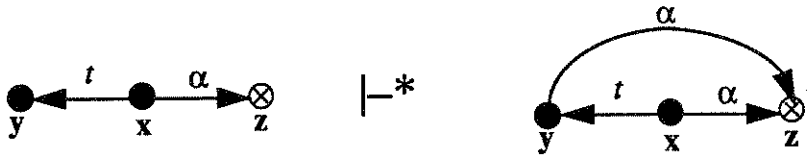
The rule is written "x creates (α to new vertex) y."

remove: Let x and y be any distinct vertices in a protection graph G_1 such that x is a subject. Let there be an explicit edge from x to y labelled β, and let $\alpha \subseteq R$. Then *remove* defines a new graph G_1 by deleting the α labels from β. If β becomes empty as a result, the edge itself is deleted. Graphically,

The rule is written "x removes (α to) y."

The edges which appear in the above graph are called *explicit* because they represent authority known to the protection system. Note that there is a duality between the take and grant rules when the edge labelled *t* or *g* is between two subjects. Specifically, with the cooperation of both subjects, rights can be transmitted backwards along the edges. The following two lemmata [12] demonstrate this:

Lemma 1.



Lemma 2.

As a result, when considering the transfer of authority between cooperating subjects, neither direction nor label of the edge is important, so long as the label is in the set { *t*, *g* }.

Definition. A *tg-path* is a nonempty sequence $v_0, \ldots, v_n$ of distinct vertices such that for all *i*, $0 \le i < n$, v_i is connected to v_{i+1} by an edge (in either direction) with a label containing *t* or *g*.

Note that the vertices in a tg-path may be either subjects or objects.

Definition. Vertices are *tg-connected* if there is a tg-path between them.

Definition. An *island* is a maximal tg-connected subject-only subgraph.

Any right that one vertex in an island has can be obtained by any other vertex in that island. In other words, an island is a maximal set of subjects which possess common rights. With each tg-path, associate one or more words over the alphabet $\{\grave{t}, \bar{t}, \grave{g}, \bar{g}\}$ in the obvious way. If the path has length 0, then the associated word is the null word ν.

Definition. A vertex v_0 *initially spans* to v_n if v_0 is a subject and there is a tg-path between v_0 and v_n with associated word in $\{\grave{t}^* \grave{g}\} \cup \{\nu\}$.

Definition. A vertex v_0 *terminally spans* to v_n if v_0 is a subject and there is a tg-path between v_0 and v_n with associated word in $\{\grave{t}^*\}$.

Definition. A *bridge* is a tg-path with v_0 and v_n both subjects and the path's associated word in $\{\grave{t}^*, \bar{t}^*, \grave{t}^* \grave{g} \bar{t}^*, \grave{t}^* \bar{g} \bar{t}^*\}$.

The following predicate formally defines the notion of *transferring authority.*

Definition. The predicate *can•share*(α, x, y, G_0) is true for a set of rights α and two vertices x and y if and only if there exist protection graphs $G_1, \ldots, G_n$ such that $G_0 \vdash^* G_n$ using only *de jure* rules, and in G_n there is an edge from x to y labelled α.

In short, if x can acquire α rights over y, then *can•share*(α, x, y, G_0) is true. The theorem which establishes necessary and sufficient conditions for this predicate to hold is [15]:

Theorem 3. The predicate *can•share*(α, x, y, G_0) is true if and only if there is an edge from x to y in G_0 labelled α, or if the following hold simultaneously:

(3.1) there is a vertex $s \in G_0$ with an s-to-y edge labelled α;

(3.2) there exists a subject vertex x′ such that x′ = x or x′ initially spans to x;

(3.3) there exists a subject vertex s′ such that s′ = s or s′ terminally spans to s; and

(3.4) there exist islands $I_1, \ldots, I_n$ such that p′ is in I_1, s′ is in I_n, and there is a bridge from I_j to I_{j+1} $(1 \leq j < n)$.

Finally, if the right can be transferred without any vertex which has that right applying a rule, the right is said to be stolen. Formally:

Definition. The predicate *can•steal*(α, x, y, G_0) is true if and only if there is no edge labelled α from x to y in G_0, there exist protection graphs $G_1, \ldots, G_n$ such that $G_0 \vdash^* G_n$ using only *de jure* rules,

in G_n there is an edge from **x** to **y** labelled α, and if there is an edge labelled α from **s** to **q** in G_0, then no rule has the form "**s** grants (α to **q**) to **z**" for any $\mathbf{z} \in G_j$ $(1 \leq j < n)$.

Essentially, this says that *can•steal* is true if *can•share* is true, the thief did not have the right initially, and no owner of the right in the initial graph gave it away. It can be shown [17]:

Theorem 4. The predicate *can•steal*(α, **x**, **y**, G_0) is true if and only if all of the following hold:

(4.1) there is no edge labelled α from **x** to **y** in G_0;

(4.2) there exists a subject vertex **x´** such that **x´** = **x** or **x´** initially spans to **x**;

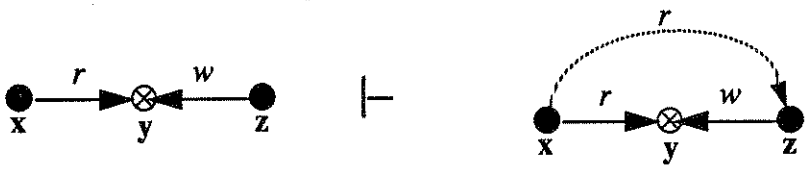
(4.3) there is a vertex **s** with an edge from **s** to **y** labelled α in G_0;

(4.4) *can•share*(t, **x´**, **s**, G_0) is true.

The *de facto* rules differ from the *de jure* rules in that they do not alter the authorities in the graph; they merely record patterns of information flow, so we cannot use an explicit edge to show the result of the application. Still, some indication of the paths along which information can be passed is necessary so we use a dashed line, labelled by r, to represent the path of a potential *de facto* transfer. Such an edge is called an *implicit* edge. Notice that implicit edges cannot be manipulated by *de jure* rules, since the *de jure* rules can only affect authorities recorded in the protection system, and implicit edges do not represent such authority.

The following set of *de facto* rules was introduced in [4] to model the transfer of information:

post: Let **x**, **y**, and **z** be three distinct vertices in a protection graph G_0, and let **x** and **z** be subjects. Let there be an (implicit or explicit) edge from **x** to **y** labelled α with $r \in \alpha$ and an edge from **z** to **y** labelled β, where $w \in \beta$. Then the *post* rule defines a new graph G_1 with an implicit edge from **x** to **z** labelled r. Graphically,



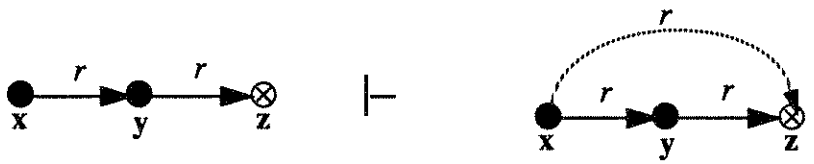
The rule is written "**z** posts to **x** through **y**," and is so named because it is reminiscent of **y** being a mailbox to which **z** posts a letter that **x** reads.

pass: Let **x**, **y**, and **z** be three distinct vertices in a protection graph G_0, and let **y** be a subject. Let there be an edge from **y** to **x** labelled α with $w \in \alpha$ and an (implicit or explicit) edge from

y to z labelled β, where $r \in \beta$. Then the *pass* rule defines a new graph G_1 with an implicit edge from x to z labelled *r*. Graphically,

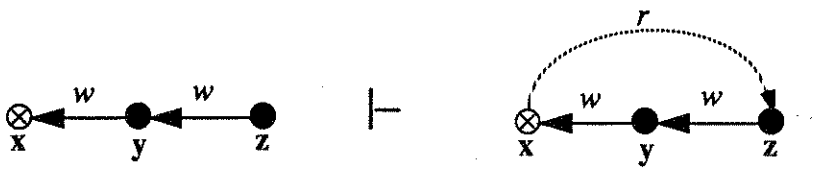
The rule is written "y passes from z to x," and is so named because y acquires the information from z and passes it on to x.

spy: Let x, y, and z be three distinct vertices in a protection graph G_0, and let x and y be subjects. Let there be an (implicit or explicit) edge from x to y labelled α with $r \in \alpha$ and an (implicit or explicit) edge from y to z labelled β, where $r \in \beta$. Then the *spy* rule defines a new graph G_1 with an implicit edge from x to z labelled *r*. Graphically,



The rule is written "x spies on z using y," and is so named because x is "looking over the shoulder" of y to monitor z.

find: Let x, y, and z be three distinct vertices in a protection graph G_0, and let y and z be subjects. Let there be an edge from y to x labelled α with $w \in \alpha$ and an edge from z to y labelled β, where $w \in \beta$. Then the *spy* rule defines a new graph G_1 with an implicit edge from x to z labelled *r*. Graphically,



The rule is written "x finds from z through y," and is named because x is completely passive; z and y give it information, which x then "finds."

Note that these rules add implicit and not explicit edges. Further, as these rules model information flow, they can also be used when any edge labelled *r* is implicit.

Definition. An *rwtg-path* is a nonempty sequence $v_0, \ldots, v_n$ of distinct vertices such that for all *i*, $0 \le i < n$, v_i is connected to v_{i+1} by an edge (in either direction) with a label containing *t*, *g*, *r* or *w*.

With each rwtg-path, associate one or more words over the alphabet $\{\grave{t}, \bar{t}, \vec{g}, \bar{g}, \grave{r}, \bar{r}, \vec{w}, \bar{w}\}$ in the obvious way.

Definition. A vertex v_0 *rw-initially spans* to v_n if v_0 is a subject and there is an rwtg-path between v_0 and v_n with associated word in $\{\grave{t}^* \vec{w}\} \cup \{v\}$.

Definition. A vertex v_0 *rw-terminally spans* to v_n if v_0 is a subject and there is an rwtg-path between v_0 and v_n with associated word in $\{\grave{t}^* \grave{r}\}$.

Definition. A *connection* is an rwtg-path with v_0 and v_n both subjects and the path's associated word in $C = \{\grave{t}^* \grave{r}, \bar{w}\bar{t}^*, \grave{t}^* \grave{r}\bar{w}\bar{t}^*\}$.

Definition. The predicate *can•know*(x, y, G_0) is true if and only if there exists a sequence of protection graphs $G_0, \ldots, G_n$ such that $G_0 \vdash^* G_n$, and in G_n there is an edge from x to y labelled *r* or an edge from y to x labelled *w*, and if the edge is explicit, its source is a subject.

In short, if x can obtain an implicit or explicit edge labelled *r* to y, then *can•know*(x, y, G_0) is true. The theorem establishing necessary and sufficient conditions for *can•know* to hold is [4]:

Theorem 5. The predicate *can•know*(x, y, G_0) is true if and only if there exists a sequence of subjects $u_1, \ldots, u_n$ in G_0 ($n \geq 1$) such that the following conditions hold:

(5.1) u_1 = x or u_1 rw-initially spans to x;

(5.2) u_n = y or u_n rw-terminally spans to y;

(5.3) for all i, $1 \leq i < n$, there is an rwtg-path between u_i and u_{i+1} with associated word in $B \cup C$.

We define an information analog to the *can•steal* predicate in which vertices with *r* rights over the target do not grant those rights to others nor participate in transfers of information.

Definition. The predicate *can•snoop*(x, y, G_0) is true if and only if *can•steal*(*r*, x, y, G_0) is true or there is a sequence of graphs and rule applications $G_0 \vdash^* G_n$ for which all of the following hold:

(a) there is no explicit edge from x to y labelled *r* in G_0;

(b) there is an implicit edge from x to y labelled *r* in G_n; and

(c) neither y nor any vertex directly connected to y is an actor in a grant rule or a *de facto* rule resulting in an (explicit or implicit) read edge with y as its target.

Necessary and sufficient conditions for this predicate to hold are [3]:

Theorem 6. For distinct vertices **x** and **y** in a protection graph G_0 with explicit edges only, the predicate *can•snoop*(**x**, **y**, G_0) is true if and only if *can•steal*(r, **x**, **y**, G_0) is true or all of the following conditions hold:

(6.1) there is no edge from **x** to **y** labelled r in G_0;

(6.2) there is a subject vertex $\mathbf{x}'$ such that $\mathbf{x}' = \mathbf{x}$ or $\mathbf{x}'$ rw-initially spans to **x** in G_0;

(6.3) there is a subject vertex $\mathbf{y}'$ such that $\mathbf{y}' \neq \mathbf{y}$, there is no edge labelled r from $\mathbf{y}'$ to **y** in G_0, and $\mathbf{y}'$ rw-terminally spans to **y** in G_0; and

(6.4) *can•know*($\mathbf{x}'$, $\mathbf{y}'$, G_0) is true.

We now consider an alternate formulation of *can•share* that will aid us in applying the Take-Grant Protection Model to various security policies.

3. Alternate Characterizations of *can•share* and *can•know*

The simplicity of the statement for necessary and sufficient conditions for *can•know* leads one to wonder why equally straightforward conditions cannot be found for *can•share*. A moment's reflection will lead to the following:

Lemma 7. The predicate *can•share*(α, **x**, **y**, G_0) is true if and only if there is a vertex $\mathbf{s} \in G_0$ with an edge to **y** labelled α, and a sequence of subjects $\mathbf{u}_1,\ldots,\mathbf{u}_n$ such that all of the following hold simultaneously:

(7.1) $\mathbf{u}_1 = \mathbf{x}$ or $\mathbf{u}_1$ initially spans to **x**;

(7.2) $\mathbf{u}_n = \mathbf{s}$ or $\mathbf{u}_n$ terminally spans to **s**; and

(7.3) there is a tg-path between $\mathbf{u}_i$ and $\mathbf{u}_{i+1}$ for all i such that $1 \leq i < n$.

Proof: ($\Longrightarrow$) Assume *can•share*(α, **x**, **y**, G_0) is true. By (3.1), there is a vertex $\mathbf{s} \in G_0$ with an edge to **y** labelled α; by (3.2), there is a vertex $\mathbf{u}_1$ satisfying (7.1), and by (3.3) there is a vertex $\mathbf{u}_n$ satisfying (7.2).

Consider now condition (3.4). Let $\mathbf{b}_j$ and $\mathbf{e}_j$ be the endpoints in I_j of the bridges connecting islands I_{j-1} with I_j and I_j with I_{j+1}, respectively. From theorem 1, we have $\mathbf{x}' = \mathbf{b}_1$ and $\mathbf{y}'' = \mathbf{e}_n$. Now, let $\mathbf{b}_j = \mathbf{z}_j^1, \ldots, \mathbf{z}_j^{kj} = \mathbf{e}_j$ be any path contained entirely in I_j. Then the members of the set

$\{ \mathbf{z}_j^i \mid 1 \leq j \leq n, 1 \leq i \leq k_j \}$

are the vertices $\mathbf{u}_i$, $1 \leq i \leq n$, and (7.3) holds.

($\Leftarrow$) Now assume that there is a vertex $\mathbf{s} \in G_0$ with an edge to $\mathbf{y}$ labelled α, and a sequence of subjects $\mathbf{u}_1,\ldots,\mathbf{u}_n$ such that (7.1)-(7.3) hold. Then (3.1) holds by assumption, (3.2) by choosing $\mathbf{x}' = \mathbf{u}_1$, and (3.3) holds by choosing $\mathbf{s}' = \mathbf{u}_n$.

Finally, note that vertices in the same island have a bridge of length 1 between them, so if $\mathbf{u}_i$ and $\mathbf{u}_{i+1}$ are directly connected, they belong to the same island, and if not (*i.e.*, there are objects on the bridge connecting them), then they belong to different islands. In either case, let the set of islands in G_0 be $\{ I_i \}$; then the set of islands in (3.4) is merely the set

$\{ I_j \mid \exists$ at least one k with $\mathbf{u}_k \in I_j \}$

This proves condition (3.4), and the lemma. ■

Intuitively, the lemma holds because the islands are simply those parts of the tg-path between $\mathbf{u}_1$ and $\mathbf{u}_n$ which consist of directly connected subjects. The advantage of this formulation is that one need not define the "islands" of the original theorem, and can instead consider simply tg-paths irrespective of maximal subject-only groupings just as in the test for *can•know* one need only consider rwtg paths and rw-terminal and rw-initial spans. So there is now a pleasing symmetry between the necessary and sufficient conditions for sharing rights and sharing information. Of course,

Corollary 8. Truth or falsity of the predicate *can•share* can be determined in time linear in the size of the graph.

Given the symmetry between lemma 7 and the theorem for *can•know*, one would expect the latter could be reformulated along the lines of the theorem for *can•share*:

Lemma 9. Let $\mathbf{x}$ and $\mathbf{y}$ be vertices in a protection graph G_0. Then *can•know*($\mathbf{x}$, $\mathbf{y}$, G_0) is true if and only if *can•share*(r, $\mathbf{x}$, $\mathbf{y}$, G_0) is true or all of the following conditions hold:

(9.1) there is a subject $\mathbf{x}'$ such that $\mathbf{x}' = \mathbf{x}$ or $\mathbf{x}'$ rw-initially spans to $\mathbf{x}$;

(9.2) there is a subject $\mathbf{y}'$ such that $\mathbf{y}' = \mathbf{y}$ or $\mathbf{y}'$ rw-terminally spans to $\mathbf{y}$;

(9.3) there is a sequence of islands I_j, $1 \leq j \leq n$, such that there is a bridge or connection from I_j to I_{j+1}, $1 \leq j < n$, and $\mathbf{x}' \in I_1$ and $\mathbf{y}' \in I_n$.

Proof: The proof is similar to that of lemma 7 and is left to the reader. ■

4. Modelling Specific Security Policies

We now consider some security policies and how they might be modelled using the Take-grant Protection Model. Each of these policies is realistic in the sense that an existing computer system uses them or they have been described in the literature as appropriate under certain condi-

tions for an operational system. We first express the policy in terms of the predicates *can•share*, *can•steal*, *can•know* and *can•snoop*, and then ask whether the policy can be modelled using the rules of the Take-Grant Protection Model. If it cannot, we describe any necessary modifications to the rules and the effect of those modifications on the predicates.

We shall explore the representation using two criteria. Let H be the set of protection graphs satisfying the security policy (we shall use subscripts to distinguish between Hs representing different policies). We shall require that the set of rules used to model the policy be sound with respect to H:

Definition. A set of graph rewriting rules P is *sound* with respect to a set of protection graphs H if applying any finite sequence of elements of P to any element $h \in H$ produces a graph $h' \in H$.

In addition, we shall examine the completeness of the rule set with respect to H and the set of original rules O = { take, grant, create, remove }:

Definition. Let $h, h' \in H$, and $h \vdash^* h'$ using the rules in O. Then if for some $P \subseteq O$, $h \vdash^* h'$ using only elements of P, the subset P is *complete*.

Note that "subset" here includes restricting the rewriting rule in some way, such as not allowing creates to add take edges.

4.1. Complete Isolation of Each Process

The policy of *total isolation* of each process prevents breaches of security and solves the confinement problem by preventing *any* transfer of information or rights among subjects [13].

To prevent any information or rights transfer (illicit or otherwise) it suffices to make all four predicates always be false. Hence for this policy, H_1 is the set of graphs with elements satisfying

$$\neg can\bullet share(\alpha,\mathbf{x},\mathbf{y},h) \wedge \neg can\bullet steal(\alpha,\mathbf{x},\mathbf{y},h) \wedge \neg can\bullet know(\mathbf{x},\mathbf{y},h) \wedge \neg can\bullet know(\mathbf{x},\mathbf{y},h)$$

for all protection graphs $h \in H_1$, all pairs of vertices **x** and **y** in h, and all subsets of rights $\alpha \subseteq R$.

To characterize this requirement in terms of the rules, note that by theorems 5 and 7, it suffices to prevent any bridges or connections between any two subjects to enforce this condition; in that case, both conditions (5.3) and (7.3) can never hold. Because the **x** and **y** referred to in the theorems may be subjects, it is not possible to prevent conditions (5.1), (5.2), (7.1), and (7.2) *a priori* from occurring; hence the above constraint is also necessary. Thus, H_1 may be characterized in terms of Take-Grant rights by:

Requirement. To enforce complete isolation of subjects, no bridges or connections may exist between any two subjects in the protection graph.

Determining whether or not a protection graph meets this requirement is easy: just look for bridges or connections between subjects. So:

Lemma 10. Testing a graph for violation of the restriction may be done in time linear in the number of edges of the graph.

Also, when a new *de jure* rule is applied, we can test for violation of the restriction just by looking at the paths affected by the rule application; in the worst case, this requires checking every edge in the graph. So:

Lemma 11. Determining whether or not an application of a *de jure* rule violates the restriction may be done in time linear to the number of edges of the graph.

This does not require any changes to the take or grant rules, since in a graph in which creation is disallowed, bridges and connections may not be constructed unless they already exist. However, it does require changing the create rule, since if one subject creates another, the parent may give itself any rights in R to the child. Should one of these rights be take, grant, read, or write, there will be a bridge, connection, or both between the parent and the child. Hence to make the rules sound with respect to this set of protection graphs H_1, the create rule must not allow any of those rights to be in the set of rights the parent has over its child:

mcreate (modified create): Let **x** be any subject in a protection graph h, and let $\alpha \subseteq R$. The modified create rule *mcreate* defines a new graph h' by adding a new vertex **y** to the graph and an edge from **x** to **y** labelled α' such that if **y** is a subject, $\alpha' = \alpha - \{ t, g, r, w \}$, and if **y** is an object, $\alpha' = \alpha$.

To see the set $P_1 = \{$ take, grant, mcreate, remove $\}$ is sound with respect to H_1, simply note bridges and connections cannot be added where they did not exist. This set of rules is not complete with respect to O and H_1, because using rules in O, one can derive a graph with two subjects that are not connected to each other having read rights over a single object (see figure 1), but one cannot obtain the same graph using the rules in P_1 because at no time can the read right be transferred from one subject to another. These arguments can be made rigorous, and show:

Lemma 12. P_1 is sound with respect to H_1, but not complete with respect to O and H_1.

As an aside, we note that if the policy is eased to allow sharing of rights between processes with the same ancestors (which would be reflected by modifying the requirement and the definition

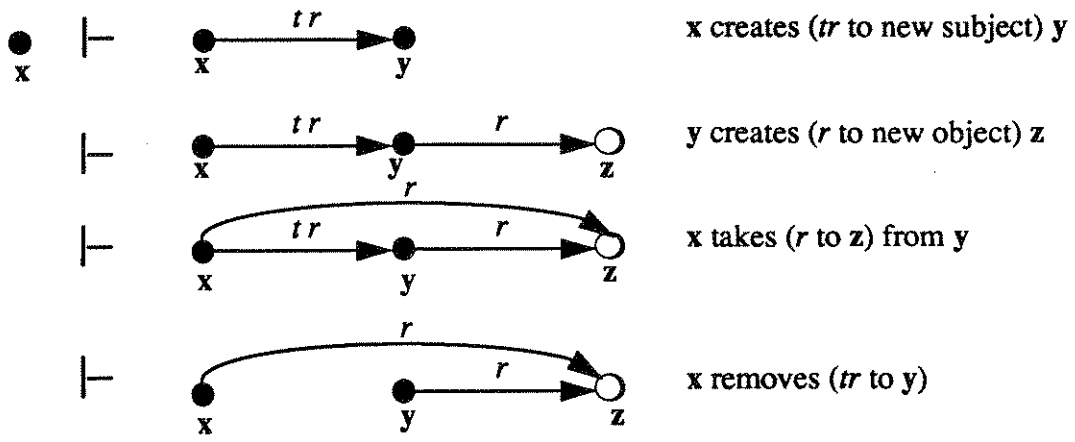


Figure 1. Example of lack of completeness of P_1 for a security policy of complete isolation.

of the set H_1 appropriately), the create rule need not be modified to produce a sound and complete set of rules. The complexity of testing for violations of the restriction is still linear in the size of the graph.

However, this security policy is so restrictive as to be uninteresting, banning (among other things) any take, grant, read, or write edges, and most paths, between subjects. So let us turn to a slightly less restrictive, and more interesting, policy.

4.2. Limited Sharing: Owner Controls Dissemination of Rights and Information

In some access control systems, the rights to manipulate files or data is under the control of the owner of that data. For example, in the Unix® operating system [16], the owner of a file controls who may access it. So this policy allows sharing of rights and information with consent of the owners of that information.

We must at this point define *ownership*. Two definitions are possible; one, more common in database work, holds that possession of a right is "ownership" of some kind. The second, more common to operating systems, defines ownership as a separate right. We examine both here.

4.2.1. Ownership Conferred by Possession of a Right

This definition, used for example in System R [9],creates a system in which mere possession of a right enables a subject to propagate that right. Hence for this policy and definition, H_2 is the set of graphs with elements satisfying

$$can\bullet share(\alpha,\mathbf{x},\mathbf{y},h) \wedge \neg can\bullet steal(\alpha,\mathbf{x},\mathbf{y},h) \wedge can\bullet know(\mathbf{x},\mathbf{y},h) \wedge \neg can\bullet snoop(\mathbf{x},\mathbf{y},h)$$

for all protection graphs $h \in H_2$, all pairs of vertices **x** and **y** in h, and all subsets of rights $\alpha \subseteq R$. The reasoning is that any two subjects must be able to share rights or information, but no set of subjects can steal rights or information without the consent of the owner. To satisfy the first two predicates, the owner must cooperate in any transfer of rights; to satisfy the latter two, the owner must cooperate in any sharing of information. We also note that any pair of subjects can share rights or information by the construction of this condition.

For *can•share*(α,**x**,**y**,h) to be true but *can•steal*(α,**x**,**y**,h) to be false, condition (4.4) must be false since negating any of conditions (4.1)-(4.3) negates either of conditions (3.1) or (3.2), making *can•share*(α,**x**,**y**,h) false. For (4.4) to be false, by lemma 7 either there must be no tg-paths between any two subjects or no take edges incident upon any subject. We note that the former again renders *can•share*(α,**x**,**y**,h) false, and that the latter renders *can•snoop*(**x**,**y**,h) false (as all rw-terminal spans will have length 1, so condition (6.3) fails) without also negating *can•know*(**x**,**y**,h). Clearly, the latter characterization of H_2 is best:

Requirement. To implement the above policy, no take edges may be incident upon any subject.

Obviously, testing for this condition is simple, as is testing for a violation when a *de jure* rule is applied:

Lemma 13. Testing a graph for a violation of the restriction may be done in time linear in the number of edges in the graph. Further, determining whether or not an application of a *de jure* rule violates the restriction may be done in constant time.

The obvious way to prevent creation of take edges is to make the appropriate modification to the *mcreate* rule described in the previous section (call the new create rule *ncreate*, and the rule set P_{21}). The argument for soundness is almost identical to the one for *mcreate*. And the argument for completeness fails also. To see why, notice that the grant rule does not add any edges to the source of the edge labelled grant. Hence, lemma 2 cannot hold (in fact, the proof that it is true using the set O of original Take-Grant Protection Model rules requires the use of a take rule). Now consider a graph h in H_2. As there are no take edges, the only take edges that could be added using the rules in O would be to new vertices. If those vertices are subjects, any subsequent manipulation of those rights would require an application of lemma 2, which is false given the replacement of the create rule by the ncreate rule. Hence:

Lemma 14. P_{21} is sound but not complete with respect to O and H_2.

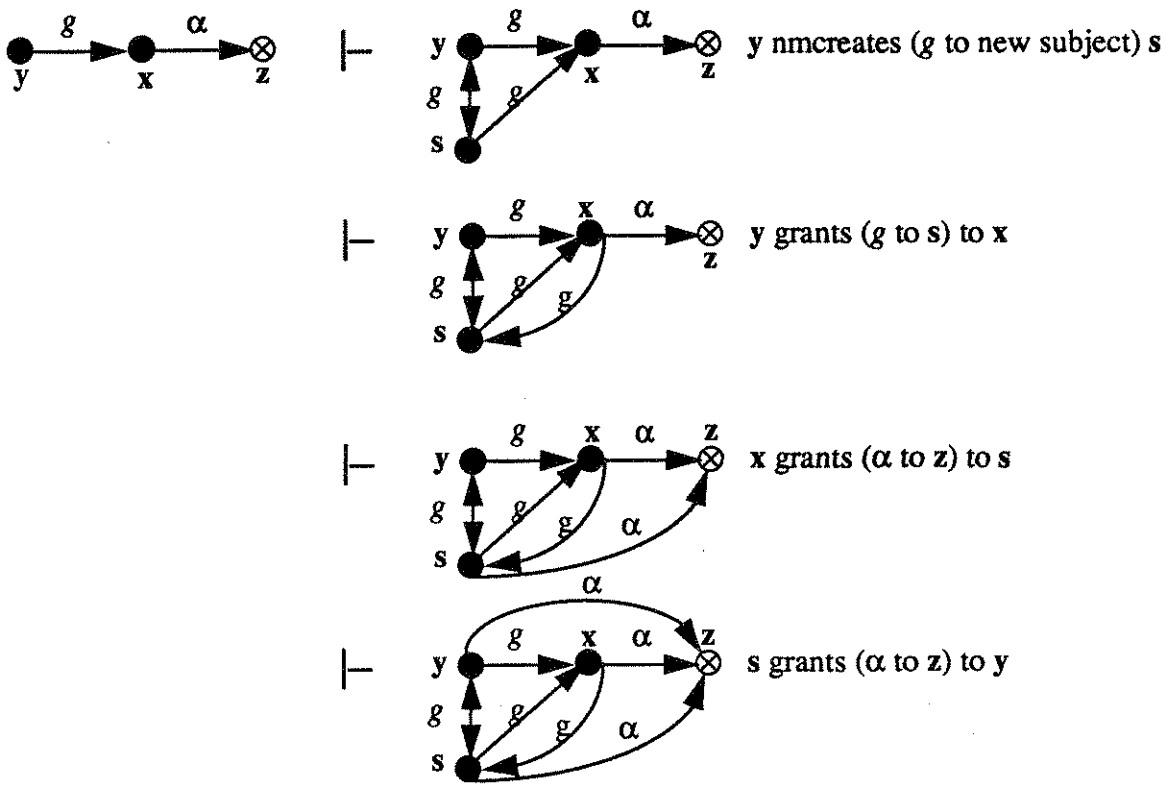


Figure 2. Proof of lemma 2 using the rule set P_{22} = { take, grant, nmcreate, remove }.

A simple way to make the new rule set P_{21} complete as well as sound is to change *ncreate* to require the creation of grant edges to every existing subject when a new subject is created:

nmcreate (*new modified create*): Let **x** be any subject in a protection graph h, and let $\alpha \subseteq R$. The modified create rule *mcreate* defines a new graph h' by adding a new vertex **y** to the graph and an edge from **x** to **y** labelled α' such that if **y** is a subject, $\alpha' = \alpha - \{ t \}$, and if **y** is an object, $\alpha' = \alpha$. Also, if **y** is a subject, edges labeled { g } are added from **y** to each subject vertex in h.

Using the rule set P_{22} = { take, grant, nmcreate, remove }, creating a new vertex would add an incoming grant edge to the parent, in which case the proof of lemma 2 is straightforward (see figure 2). Then the argument for completeness goes as before, except that as lemma 2 is true, the transfer of any rights from a newly created subject to the parent using take can be emulated by the child granting the parent the rights. If the child is an object, of course, the issue never arises since the only way a right can be added to the child is by the parent granting the right, in which case application of the take rule is redundant and can be eliminated. This can be made formal, and shows:

Lemma 15. P_{22} is both sound and complete with respect to O and H_2.

4.2.2. Ownership As a Right

Using this definition, ownership is itself a right and only the owner may distribute rights. This version is similar to the propagation prevention mechanism such as HYDRA's environmental rights mechanism [5],[11] and, at least in operating systems, is more common than an unconstrained ability to transmit rights.

First, note that the rules for the Take-Grant Protection Model fail to capture the system's distinction between owning an object and owning a right to an object. The simplest incorporation of this distinction is a modification of the grant rule.Define a new ownership right o. Then:

mgrant (modified grant rule): Let **x**, **y**, and **z** be three distinct vertices in a protection graph G_0, and let **x** be a subject. Let there be an edge from **x** to **y** labelled γ with $g \in \gamma$, an edge from **x** to **z** labelled β with $o \in \beta$, and $\alpha \subset \beta$ with $o \notin \alpha$. Then the modified grant rule *mgrant* defines a new graph G_1 by adding an edge to the protection graph from **x** to **z** labelled α.

In short, no rights to an object may be granted unless the grantor has an o right to the object. Further, the o right cannot be granted. Let P_{23} = { take, mgrant, nmcreate, remove }. Then:

Lemma 16. Given the set of rules P_{23}, only the owner of a target can propagate a right to that target.

Now, the set of graphs under consideration is still H_2. But how does the change in the grant rule affect the restriction necessary to ensure that owners can share information or rights, but that those cannot be stolen or snooped? As both stealing and snooping may involve the propagation of rights, if neither is possible, then limiting the propagation of rights further will not enable either. In other words, the modified grant rule limits the spread of rights and so those predicates that are false in the original model will be false in the modified model. So the requirement for the unconstrained case is at least sufficient.

To carry the analysis a bit deeper, a right can clearly be shared between the owner and any other subject since, by the statement of the security policy, all subjects have grant rights over one another. However, a simple construction will show that rights can be propagated between endpoints of bridges as well, but no further. Hence given the modified grant rule, *can•share* is true if and only if the number of subjects in condition (7.3) is 2.

We note that the rule set P_{23} is sound since no take edges are added by any of the rules. Completeness follows from the same argument as before. Hence:

Lemma 17. P_{23} is both sound and complete with respect to O and H_2.

4.3. Multilevel Security Policy

Multilevel policies are quite common in existing systems (see for example [6],[16]). Here we generalize the results in [2] and extend them to include *can•share*, *can•steal* and *can•snoop*. Our policy is a generalization of the Bell-LaPadula model [1] in which a set U of rights may not extend from a lower to a higher level, and a set D of rights that may not extend from a higher to a lower level.We shall assume that $r \in U$ and $w \in D$ so that the simple security and *-properties hold.

This policy associates with each subject and object a *security level* and a set of *categories*. Informally, a subject can access an object (in any way other than by using the rights in D) if the security level of that object is no greater than the level of the subject and the object's set of categories is a (possibly improper) subset of the subject's categories; similarly, the subject can access an object (in any way other than by using the rights in U) if the security level of the object is no less than that of the subject and the object's set of categories is a (possibly improper) subset of the subject's categories. Formally, define the functions $sl(\mathbf{x})$ and $cs(\mathbf{x})$ to be the security level and set of categories of the vertex **x**, respectively; then **x** *dominates* **y** (written $\mathbf{x} \geq \mathbf{y}$) if both $sl(\mathbf{x}) \geq sl(\mathbf{y})$ and $cs(\mathbf{y}) \subseteq cs(\mathbf{x})$, and **x** *strictly dominates* **y** (written $\mathbf{x} > \mathbf{y}$) if $\mathbf{x} \geq \mathbf{y}$ and $sl(\mathbf{x}) \neq sl(\mathbf{y})$ or $cs(\mathbf{y}) \not\subseteq cs(\mathbf{x})$. (Following Gasser [7], we shall call the pair $(sl(\mathbf{x}), cs(\mathbf{x}))$ an *access class*.)

We consider two types of systems: those with mandatory access controls only, and those with both mandatory and discretionary access controls.

4.3.1. Mandatory Access Controls Only

In a multilevel security policy with mandatory controls only, permission of the owner of rights to an object need not be acquired to obtain access to that object; if access can be granted according to the multilevel rules, it will be. This security policy may then be summarized by requiring the elements of H_{31} to satisfy:

$$[\forall \delta \subseteq D, \upsilon \subseteq U]\ \mathbf{x} > \mathbf{y} \Leftrightarrow \neg can\bullet share(\delta,\mathbf{x},\mathbf{y},h) \wedge \neg can\bullet share(\upsilon,\mathbf{y},\mathbf{x},h) \wedge can\bullet know(\mathbf{x},\mathbf{y},h)$$

for all protection graphs $h \in H_{31}$, all pairs of vertices **x** and **y** in h, and all subsets of rights $\alpha \subseteq R$. This captures that if **x** strictly dominates **y**, **x** cannot υ to **y** and **y** cannot δ to **x** (where υ and δ are subsets of U and D, respectively) and that the owner need not consent to any transfers of information or rights (hence the omission of *can•steal* and *can•snoop*).

We extend the characterization of a similar (straightforward Bell-LaPadula) model in [2] to describe the class of protection graphs in H_{31}. Each set of nodes in a single access class is represented as a *protection level*:

Definition. A *protection level* is a set of subjects in a protection graph G such that, for every pair of vertices **x** and **y** in the set, $can\bullet share(\alpha,x,y,G)$, $can\bullet share(\alpha,y,x,G)$, $can\bullet know(x,y,G)$ and $can\bullet know(y,x,G)$ are true.

This says that within a protection level (access class), all vertices can obtain any information or rights from any other vertex in that level.

Definition. Two subject vertices **x** and **y** are said to be *joined* if $can\bullet know(x,y,G)$ is true, and $can\bullet know(y,x,G)$, $can\bullet share(\upsilon,y,x,G)$, and $can\bullet share(\delta,x,y,G)$ are false.

It can be shown (along the lines of the proof of proposition 4.4 in [2]) that this relation defines a partial ordering; so, if the members of one access class dominate another, the members of the protection level representing the first access class are joined to the second, and we say the first protection level is *higher* than the second (or, the second is *lower* than the first). Intuitively, what the two definitions say is that within a protection level, any amount of sharing of information or rights is fine, but between protection levels, information may flow upward, and any sharing of rights that does not involve adding edges labelled with rights in U going from a lower protection level to a higher one, or edges labelled with rights in D going from a lower protection level to a higher.

Given this representation, it can be shown that [2]:

Requirement. To enforce a multilevel security policy of the sort described here, there can be no bridges or connections between protection levels.

In essence, this forbids *any* transfer of information or rights between protection levels, which certainly satisfies the policy. It is also straightforward to prove:

Lemma 18. Testing a graph for a violation of the restriction may be done in time linear in the number of edges in the graph. Further, determining whether or not an application of a *de jure* rule violates the restriction may be done in constant time.

This representation is probably overly restrictive because it limits the sharing of rights other than those in U and D, and since the multilevel policy deals only with those rights, any representation should deal only with them. In [2], three modifications to the rules in O are considered; we generalize the most interesting here:

mlstake (multilevel secure take): Let **x**, **y**, and **z** be three distinct vertices in a protection graph G_0, and let **x** be a subject. Let there be an edge from **x** to **y** labelled γ with $t \in \gamma$, an edge from **y** to **z** labelled β, and $\alpha \subset \beta$. Then the multilevel secure take rule *mlstake* defines a new graph G_1 by adding an edge to the protection graph from **x** to **z** labelled α' such that if $\mathbf{x} > \mathbf{y}$, $\alpha' = \alpha - D$ and if $\mathbf{y} > \mathbf{x}$, $\alpha' = \alpha - U$.

mlsgrant (multilevel secure grant): Let **x**, **y**, and **z** be three distinct vertices in a protection graph G_0, and let **x** be a subject. Let there be an edge from **x** to **y** labelled γ with $g \in \gamma$, an edge from **x** to **z** labelled β, and $\alpha \subset \beta$. Then the multilevel secure grant rule *mlsgrant* defines a new graph G_1 by adding an edge to the protection graph from **x** to **z** labelled α' such that if $\mathbf{x} > \mathbf{y}$, $\alpha' = \alpha - D$ and if $\mathbf{y} > \mathbf{x}$, $\alpha' = \alpha - U$.

These rules simply modify the original take and grant rules to prevent adding edges with rights in U from a vertex in a lower protection level to a higher one, or edges with rights in D from a higher protection level to a lower one. If P_{31} is the set of rules obtained by replacing the take and grant rules in O with the mlstake and mlsgrant rules, it can be shown [2]:

Lemma 19. P_{31} is both sound and complete with respect to O and H_{31}.

4.3.2. Mandatory and Discretionary Access Controls

In this case, first mandatory access controls determine if the desired access is allowed by the system; if it is, then the discretionary access controls determine if access is allowed by the owner of the information. So, with this policy, in addition to the statement of mandatory access, a component for discretionary access must be included, and the elements in H_{32} must satisfy:

$$[[\forall \delta \subseteq D, \upsilon \subseteq U]\ \mathbf{x} > \mathbf{y} \Leftrightarrow \neg can\bullet share(\delta,\mathbf{x},\mathbf{y},h) \wedge \neg can\bullet share(\upsilon,\mathbf{y},\mathbf{x},h) \wedge can\bullet know(\mathbf{x},\mathbf{y},h)] \wedge \neg can\bullet steal(\alpha,\mathbf{y},\mathbf{x},h) \wedge \neg can\bullet snoop(\mathbf{x},\mathbf{y},h)$$

for all protection graphs $h \in H_{32}$, all pairs of vertices **x** and **y** in h, and all subsets of rights $\alpha \subseteq R$. In addition to the limits discussed in the mandatory-only case, this formula also captures that the owner must consent to any transfers of information or rights by the inclusion of *can•steal* and *can•snoop*.

We now redefine *protection level* and *joined* to disallow stealing and snooping in the obvious way. Then in addition to their being no bridges or connections between protection levels, there can also be no take edges incident upon subjects by the reasoning in section 4.2.1. Hence:

Requirement. To enforce a multilevel security policy involving both mandatory and discretionary access controls, there must be no take edges incident upon subjects, nor any bridges or connections between protection levels.

Clearly, violations can be tested in time linear in the number of edges of the graph, and when a *de jure* rule is applied, examining the new edge will reveal violations. As before, this requirement, while effective, is quite restrictive.

Let us add the right t to both sets U and D, and further alter the create rule to bar creation of any edge labelled t and add grant edges to all subjects in the new vertex's protection level when the new vertex is a subject. (This is in the spirit of the *nmcreate* rule in section 4.2.1. However, here the "parents" are simply all vertices in the protection level.) Call the new create rule *mlscreate*. Then let P_{32} = { mlsgrant, mlscreate, remove } and remove the above requirement. What are the effects? If a graph h is in H_{32}, applying those rules will generate a new graph h' also in H_{32}, because no stealing or snooping can be done (since the lack of take edges incident on subjects violates both conditions (4.4) and (6.3), so by theorems 4 and 6, those predicates can never hold), and clearly no edges with labels in U can be added from a lower protection level to a higher, and no edges with labels in D can be added from a higher protection level to a lower; so this set of rules is sound with respect to H_{32}. Next, notice that since no take edges are present, the take rule could never be applied, and further if the ordinary create rule were used, the mlscreate rule could be simulated by having the creator at once grant g rights over all vertices in the protection level to the newly created subject. Hence P_{32} is complete. This can be formalized to show:

Lemma 20. P_{32} is both sound and complete with respect to O and H_{32}.

4.4. Tree-Based Hierarchies

A *tree hierarchy* is a set of entities arranged in a tree structure. We shall use the term *tree-based hierarchy* to refer to a structure derived from a tree hierarchy. An *ancestor* of vertex **x** is a vertex which can acquire a right from **x** with or without **x**'s cooperation. The *level* of **x** is the length of the shortest path to the root containing only the ancestors of **x** and **x** itself. Define a *common ancestor* of vertices **x** and **y** to be the set of vertices which are ancestors to both **x** and **y**. We then consider a security policy that allows vertex **x** to share rights or information with vertex **y** if and only if a common ancestor allows the sharing. We specifically consider those hierarchies which are initially trees, but may degenerate into directed graphs.

This policy is a generalization of one discussed in [18]; protocols to implement it were discussed in [21], and similar, less general policies are the basis for many protection schema (for example, for the Unix® operating system, the tree level is 2, with the superuser being the root vertex and all users being the nodes to which the root is ancestor). The focus of this discussion is more fundamental: what is the specific policy being enforced, and how should the rules in O be modified to prevent breaches of security?

Define $anc(\mathbf{x}_1,\ldots,\mathbf{x}_n)$ to be the set of common ancestors of $\mathbf{x}_1, \ldots, \mathbf{x}_n$. Then for this policy, H_4 is the set of graphs satisfying:

$$[can\bullet share(\alpha,\mathbf{x},\mathbf{y},h) \Rightarrow [\exists\, \mathbf{z}\in\ anc(\mathbf{x},\mathbf{y})\ [can\bullet share(\alpha,\mathbf{z},\mathbf{y},h) \wedge can\bullet share(g,\mathbf{z},\mathbf{x},h)]]] \wedge$$
$$[can\bullet know(\mathbf{x},\mathbf{y},h) \Rightarrow [\exists\, \mathbf{z}\in\ anc(\mathbf{x},\mathbf{y})\ [can\bullet know(\mathbf{z},\mathbf{y},h) \wedge can\bullet know(\mathbf{x},\mathbf{z},h)]]] \wedge$$
$$[\ \forall\ \mathbf{z}\in\ anc(\mathbf{x})\ [\neg can\bullet snoop(\mathbf{x},\mathbf{z},h) \wedge\ \neg can\bullet steal(t,\mathbf{x},\mathbf{z},h)]]$$

for all protection graphs $h \in H_4$, all pairs of vertices $\mathbf{x}$ and $\mathbf{y}$ in h, and all subsets of rights $\alpha \subseteq R$. In intuitive terms, this says simply that no vertex may steal or snoop an ancestor, and that if two vertices can share rights or information, one of their common ancestors must have assisted in that sharing.

To characterize such a graph in Take-Grant terms, first note that while each ancestor may have both take and grant authority over its descendents, no vertex may have take authority over any ancestor. (Hence the definition of *ancestor* of a vertex $\mathbf{x}$ ay be characterized succinctly as the set of vertices lying on any path from the root to $\mathbf{x}$ with associated word in $\grave{t}^*$.) We then note that $can\bullet steal(\alpha,\mathbf{x},\mathbf{y},h)$ is true only if some vertex $\mathbf{z}$ has α rights over $\mathbf{y}$, and a common ancestor of $\mathbf{x}$ and $\mathbf{z}$ participates in the theft, the former condition being required by condition (4.3), and the latter following by the common ancestors being foci of access sets which connect that part of the conspiracy graph associated with h containing $\mathbf{x}$ with that part containing $\mathbf{y}$ [20]. Further, both those conditions enable one to exhibit a witness to $can\bullet steal(\alpha,\mathbf{x},\mathbf{y},h)$; hence,

Lemma 21. $can\bullet steal(\alpha,\mathbf{x},\mathbf{y},G)$ is true if and only if there exists a vertex $\mathbf{z}$ with α rights over $\mathbf{y}$ in G, and a common ancestor of $\mathbf{x}$ and $\mathbf{z}$ applies one (or more) rules instantiating the theft.

A similar result holds for $can\bullet snoop$:

Lemma 22. $can\bullet snoop(\mathbf{x},\mathbf{y},G)$ is true if and only if there exists a vertex $\mathbf{z}$ which rw-terminally spans to $\mathbf{y}$ in G but does not have r rights over $\mathbf{y}$ in G, and a common ancestor of $\mathbf{x}$ and $\mathbf{z}$ applies one (or more) rules instantiating the snooping.

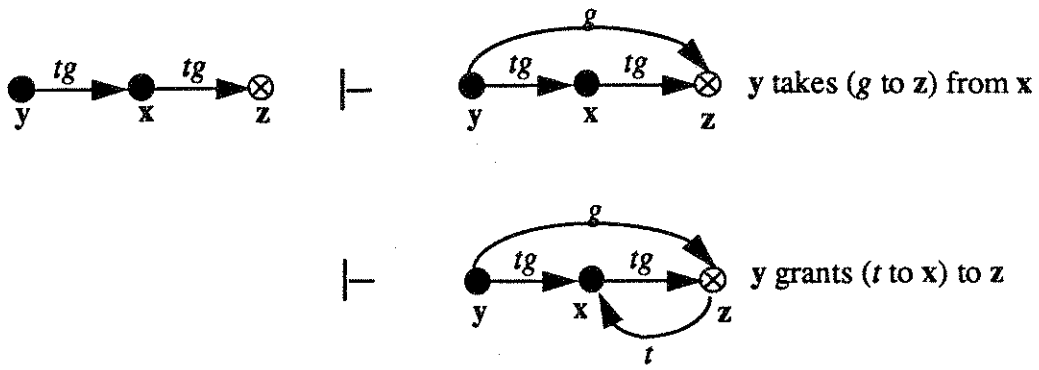


Figure 3. Lack of soundness of O under a tree-based hierarchical security policy.

Unfortunately, a very quick construction shows that O is not sound with respect to H_4 (see figure 3). Hence, to enforce soundness, the rules must be modified. The most obvious modification follows from the previous section; since in essence this is a hierarchy, simply restrict the application of the take and grant rules so that no take edges may be added going up:

tbhtake (tree-based hierarchical take): Let **x**, **y**, and **z** be three distinct vertices in a protection graph G_0, and let **x** be a subject. Let there be an edge from **x** to **y** labelled γ with $t \in \gamma$, an edge from **y** to **z** labelled β, and $\alpha \subset \beta$. The tree-based hierarchical take rule *tbhtake* defines a new graph G_1 by adding an edge to the protection graph from **x** to **z** labelled α' such that if $y \in anc(\mathbf{x}, \mathbf{y})$, $\alpha' = \alpha - \{ t \}$.

tbhgrant (tree-based hierarchical grant): Let **x**, **y**, and **z** be three distinct vertices in a protection graph G_0, and let **x** be a subject. Let there be an edge from **x** to **y** labelled γ with $g \in \gamma$, an edge from **x** to **z** labelled β, and $\alpha \subset \beta$. Then the tree-based hierarchical grant rule *tbhgrant* defines a new graph G_1 by adding an edge to the protection graph from **x** to **z** labelled α' such that, if $y \in anc(\mathbf{x}, \mathbf{y})$, $\alpha' = \alpha - \{ t \}$.

Now, as no take edges ever extend from a vertex to an ancestor, both snooping and thieving are impossible unless an ancestor common to the source and target cooperates. Unfortunately, completeness no longer holds since take edges from a vertex to an ancestor can no longer be emulated with the new rules, as lemma 2 fails for the reasons given in section 4.2.1. So to ensure completeness, a modified create rule similar to the *nmcreate* rule must be used:

tbhcreate (tree-based hierarchical create): Let **x** be any subject in a protection graph h, and let $\alpha \subseteq R$. The tree-based hierarchical create rule *tbhcreate* defines a new graph h' by adding a new vertex **y** to the graph and an edge from **x** to **y** labelled α' such that, if **y** is a subject,

then $\alpha' = \alpha - \{ t \}$, and if y is an object, $\alpha' = \alpha$. Also, if y is a subject, edges labeled $\{ g \}$ are added from y to each subject vertex in *anc*(y).

The proof of completeness now follows, so letting P_4 = { tbhtake, tbhgrant, tbhcreate, remove }:

Lemma 23. P_4 is both sound and complete with respect to O and H_4.

5. Conclusion

The ability to represent useful security policies shows that the Take-Grant Protection Model may prove to be not only a theoretical tool for studying some ramifications of the safety question, but also a powerful, practical model that can be used to study abstract representations of real systems for safety (or methods that could be used to leak information or rights). This paper has provided several such representations. Of course, other methods of abstraction could have been used.

As an example, the four basic rules (take, grant, create, remove) could have been bound together to form *protocols*, and subjects required to apply rules only by executing one (or more) of those protocols. Rather than employ this strategy to ensure soundness, as Snyder [18] and Wu [21] have done, we have considered modifications to the take-grant graph rewriting rules. This has two effects.

A protocol strategy first implies that all subjects will use the primitive operations only through the protocols, and that they can be trusted not to call on the primitive operations directly. For example, in figure 3, Wu's protocols assume that no ancestor will grant take rights in the way that was done there. Given such an assumption (that all subjects are, in that sense at least, honest), a protocol strategy works well. However, it ignores the possibilities of theft and snooping. When those are considered, it becomes clear that the "protocols" *must* be the "primitive operations" to ensure that the subjects only use them to alter the graph.

Secondly, the graph rewriting rules presented here are far more general than the protocols used by Wu. As an example, in her analysis of the tree-based hierarchical security policy, there is an implicit assumption that if a subject wants a right from another node which is not a descendent, a common ancestor must initiate the transfer. In this analysis, we make no such assumption; the vertex may, in effect, "request" the ancestor transfer the right by initiating a transfer. However, to complete the transfer, the ancestor must apply one (or more) rules. The difference is that between initiation and cooperation.

Much work remains before the Take-Grant Protection Model can be used in practise. For example, many systems treat rights reflexively (that is, a subject can take or grant itself rights, as well as read itself).The Take-Grant Protection Model does not. But, as Snyder has pointed out, reflexivity "probably is not fundamental to the Take-Grant Model in the first place," ([19], p. 179) and it has some benefits when making an abstraction of an existing system. When one models a "process," one may actually be modelling several entities: the process thread itself, the process control block (if the process can read or alter it directly), those portions of memory in which variables are stored (for processes which do not read or write *anything* are useless, unless one is considering the problem of covert channels, a subject well beyond the scope of this paper), and files. Each of these should be considered as separate entities, because each interacts with the execution of the process in a different way; and the lack of reflexivity in Take-Grant encourages such a division. This suggests that in the application of a formal model like Take-Grant, irreflexivity may benefit the modelling process. Analysis of a reflexive Take-Grant Protection Model would help determine if this intuition is correct, or when a reflexive system should be used.

We have modified many of the Take-Grant graph rewriting rules to capture compliance with a security policy. Part of the reason for this was to ensure the theorems stating necessary and sufficient conditions for *can•share, can•know, can•steal* and *can•snoop* held. However, it might be possible to produce "meta-theorems" which, for a given rule set, produce conditions under which those four predicates hold. Such meta-theorems would almost be a necessity for any realistic use of the Take-Grant Protection Model in systems analysis, since no two systems will have the same rule sets. This issue requires much further work.

Finally, most computer systems have some concept of "group." Incorporating such a notion into the Take-Grant Model would be invaluable towards its practical application.

6. Reference

[1] Bell, D. and LaPadula, L., "Secure Computer Systems: Mathematical Foundations and Model," M74-244, The MITRE Corp., Bedford, MA (May 1973).

[2] Bishop, M., "Hierarchical Take-Grant Protection Systems," *Proc. 8th Symp. on Operating Systems Principles* (Dec. 1981) pp. 107-123

[3] Bishop, M., "Theft of Information in the Take-Grant Protection Model," *Proceedings of the Computer Security Foundations Workshop* (June 1988) pp. 94-218.

[4] Bishop, M. and Snyder, L., "The Transfer of Information and Authority in a Protection System," *Proc. 7th Symp. on Operating Systems Principles* (Dec. 1979) pp. 45-54.

[5] Cohen, E. and Jefferson, D., "Protection in the HYDRA operating system," *Operating System Reviews* **9**(5) (1975), pp. 141-160.

[6] Department of Defense, *Trusted Computer System Evaluation Criteria.* DOD 5200.28-STD (1985).

[7] Gasser, M., *Building a Secure Computer System*, Van Nostrand Reinhold (1988).

[8] Graham, G. and Denning, P., "Protection: Principles and Practices," *Proc. AFIPS Spring Joint Computer Conf.* **40** (1972) pp. 417-429.

[9] Griffiths, P. and Wade, B., "An Authorization Mechanism for a Relational Database System," *Transactions on Database Systems* **1**(3) (Sep. 1976) pp. 242-255.

[10] Harrison, M., Ruzzo, W., and Ullman, J., "Protection in Operating Systems," *CACM* **19**(8) (Aug. 1976), pp. 461-471.

[11] Jones, A., "Protection Mechanism Models: Their Usefulness," in *Foundations of Secure Computing*, Academic Press, New York City, NY (1978) pp. 237-254.

[12] Jones, A., Lipton, R., and Snyder, L., "A Linear Time Algorithm for Deciding Subject Security," *Proc. 17th Annual Symp. on the Foundations of Computer Science* (Oct. 1976) pp. 33-41.

[13] Lampson, B., "A Note on the Confinement Problem," *CACM* **16**(10) (Oct. 1973) pp. 613-615.

[14] Lampson, B., "Protection," *Fifth Princeton Conf. on Information and Systems Sciences* (Mar. 1971) pp. 437-443.

[15] Lipton, R. and Snyder, L., "A Llinear Time Algorithm for Deciding Subject Security. *J. ACM*. **24**(3) (Jul. 1977) pp. 455-464.

[16] Ritchie, D. and Thompson, K., "The Unix Time-Sharing System. *CACM* **17**(7) (Jul. 1974) pp. 388-402.

[17] Sandhu, R., "The Schematic Protection Model: its Definition and Analysis for Acyclic Attenuating Schemes," *J. ACM* **35**(2) (Apr. 1988) pp. 404-432.

[18] Snyder, L., "On the Synthesis and Analysis of Protection Systems," *Proc. Sixth Symp. on Operating Systems Principles* (Nov. 1977) pp. 141-150.

[19] Snyder, L., "Formal Models of Capability-Based Protection Systems," *IEEE Transactions on Computers* **C-30**(3) (Mar. 1981) pp. 172-181.

[20] Snyder, L., "Theft and Conspiracy in the Take-Grant Protection Model," *JCSS* **23**(3) (Dec. 1981) pp. 333-347.

[21] Wu, M., "Hierarchical Protection Systems," *Proc. of the 1981 Symp. on Privacy and Security* (1981) pp. 113-123.