Theft of Information in the Take-Grant Protection Model

Matt Bishop

Department of Computer Science
University of California at Davis
Davis, CA 95616-8562

ABSTRACT

Questions of information flow are in many ways more important than questions of access control, because the goal of many security policies is to thwart the unauthorized release of information, not merely the illicit obtaining of access rights to that information. The Take-Grant Protection Model is an excellent theoretical tool for examining such issues because conditions necessary and sufficient for information to flow between two objects, and for rights to objects to be obtained or stolen, are known. In this paper we extend these results by examining the question of information flow from an object the owner of which is unwilling to release that information. Necessary and sufficient conditions for such "theft of information" to occur are derived. To emphasize the usefulness of these results, the security policies of complete isolation, transfer of rights with the cooperation of an owner, and transfer of information (but not rights) with the cooperation of the owner are presented; the last is used to model a subject guarding a resource.

Key Words and Phrases: conspiracy, information flow, isolation, sharing, safety problem, security policy, take-grant protection model, theft

1. Introduction

The terms *security* and *safety* are often used interchangeably; in fact, they are not synonyms. The term "safe" applies to a system in which it is not possible to reach a new state in which something undesirable occurs. In terms of security models, this means that a right, or information, can be transferred without authority, and the term "secure" is applied to those systems which are "safe," *i.e*. that do not allow the transfer of rights or information illicitly. When applied to a concrete system, security requires not only that the abstract model of the system be secure, but also that the concrete system correctly implement the abstract model. Because of the complexity of demonstrating correctness of implementation, analyses of the security of the abstract models are far more common than proofs of security.

Portions of this work were supported by grant NAG2-480 from the National Aeronautics and Space Administration to Dartmouth College, were done at Dartmouth College, and stem from the author's Ph.D. dissertation. A preliminary version of this paper was presented at the First Computer Security Foundations Workshop in Franconia, NH (1988).

Among the earliest models used to analyze the security of models of computer systems was the *access control matrix* [10][13][16]; by representing commands as a set of tests for the presence of specific rights in specific locations which, if true, caused the execution of a sequence of primitive operations, Harrison, Ruzzo, and Ullman were able to prove that in general, it cannot be determined whether a system is safe [13]. The efforts to understand the limits of this *safety question* – that is, under what conditions can the security of a system be determined – have spurred the study of several models.

Harrison and Ruzzo used the access control matrix model to study the relationship of the complexity of commands to the safety question. In [13], it had been shown that if each command consisted of exactly one primitive operation, the safety question was decidable; in [12], Harrison and Ruzzo showed that when the system is *monotonic* (that is, no right, no subject, and no object may be deleted or destroyed), the safety question is still undecidable, and restricting the set of tests at the beginning of each command to have as few as two elements (*biconditional*) does not make it decidable. But if that set has only one test, then the safety of monotonic systems is decidable. This work was generalized in [25].

The Schematic Protection Model [23][24] has been used to examine a limited class of systems (called *acyclic attenuating systems*), and the safety question for such schemes was shown to be decidable. These systems place restrictions on the creation of subjects and objects; specifically, at creation a child may not have more rights than its parent (attenuation), and a subject of type A cannot create a subject of type B if a subject of type B can (either directly or, through its descendents, indirectly) create a subject of type A (acyclicness; this holds for all $A \neq B$). This model allows very general rules to be used to determine when one of three primitive operations (*copy*, *demand*, and *create*) may be performed; in terms of the access control matrix model, it essentially restricts the sequences of primitive operations that make up the body of commands to a set of three without restricting the set of tests in the commands.

The Take-Grant Protection Model [15] differs from the access control matrix model and the schematic protection model by specifying both the sequences of primitive operations making up the body of the commands and the set of tests upon which the execution of those sequences is conditioned. Further, and more surprisingly, it also prevents a subject from obtaining rights over itself; this makes representing it in terms of the other two models rather difficult. (If reflexivity were allowed, Take-Grant would fall into the class of acyclic attenuating systems characterizable using the schematic protection model, showing the safety question to be decidable for that system.) However,

irreflexivity does not appear to be fundamental to the model [27]. We shall touch on this again in the conclusion.

General questions of safety aside, questions of access control often require specific answers in specific settings; for example, given some explicit set of operations, can one user access a file belonging to another user? This question has been discussed for many types of systems, for many sets of rules, and in contexts other than formal modelling (for example, see [9][19][22]). The advantage of not using formal models is that the discussion focuses on what access control policies are realistic, both in implementation and in enforcement; the disadvantage is that the effects of the rules implementing those policies depend on a host of factors that are often overlooked or not understood thoroughly.

The Take-Grant Protection Model presents features of both types of models, namely those used to analyze questions of safety and those used to analyze the effects of specific access control policies. It represents systems as graphs to be altered by specific operations and, although the model was developed to test the limits of the results in [13], in this model, safety is not merely decidable even if the number of objects which can be created is unbounded, but it is decidable in time linear in the size of the graph. Further, the focus of most studies of this model have been on characterizing conditions necessary and sufficient for the transfer of rights, on the number of active entities that must cooperate for such transfers to take place, and on the complexity of testing for those conditions in a representation of a system. For this reason it is in some sense of more "practical" use than other formal systems, in that the safety question is decidable and the study of the complexity of conditions allowing compromise is emphasized.

Early work on the Take-Grant Protection Model [15][18] dealt with the transfer of rights assuming all active agents in the system would cooperate. Snyder extended these characterizations to include conditions under which rights could be stolen [28]; Bishop and Snyder introduced the notion of information flow and formulated necessary and sufficient conditions for information sharing [5].

Distinct from other access control models, the Take Grant Protection Model formalizes the notions of conspiracy in the context of transfer of rights [28] and information flow (this work). One contribution of this paper is to combine the notion of "theft" with the notion of "information flow," thereby providing an information flow analogue to the *can•snoop* predicate; the other is to extend the idea of conspiracy to the theft of information. This line of analysis has not been attempted with

other models, and may serve as a guide to extending access control models to handle information flow, thus beginning to bridge the gap between access control models and information flow models.

Applications of the model to various systems have been explored [4][14][26][30]. This paper also characterizes some very simple systems using the new results, but does not attempt to analyze them at length because the focus of this paper is on information and the conspiracies needed to obtain it. Using the new results to analyze current models of disclosure and integrity (for example, those described in [1][7][8][20][21][29]) is itself a separate paper; it is beyond the scope of the issues addressed here.

In the next two sections, we review the rules governing transfer of rights and information within the model, as well as some of the consequences of those rules. Next, we define, and present necessary and sufficient conditions for, the theft of information; following that, we present bounds on the number of actors needed for information to be shared (or stolen). We then briefly compare our results to similar ones for theft of rights. To demonstrate the usefulness of the concepts, we express the security policies of total isolation, owners being allowed to transfer rights, owners being allowed to transfer information but not rights, and then analyze the concept of a reference monitor. Finally, we suggest areas for future research.

2. Transfers of Authority

Let a finite, directed graph called a *protection graph* represent a system to be modelled. A protection graph has two distinct kinds of vertices, called *subjects* and *objects*. Subjects are active vertices, and (for example) can represent users; they can pass authority by invoking *graph rewriting rules*. Objects, on the other hand, are completely passive; they can (for example) represent files, and do nothing.

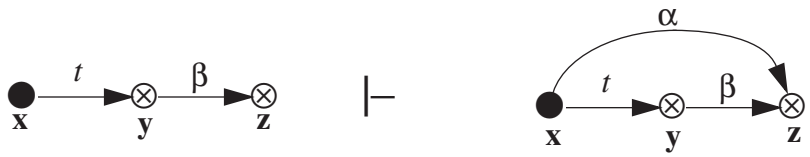
In protection graphs, the subjects are represented by ● and objects by ❍. Vertices which may be either subjects or objects are represented by ⊗. Pictures are very often used to show the effects of applying a graph rewriting rule on the graph; the symbol |– is used to mean that the graph following it is produced by the action of a graph rewriting rule on the graph preceding it. The symbol |–* represents several successive 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 edges of a protection graph are labelled with subsets of a finite set R of rights. Suppose that $R = \{ r, w, t, g \}$, where *r*, *w*, *t*, and *g* represent *r*ead, *w*rite, *t*ake, and *g*rant rights, respectively.

When written as labels on a graph, or when the set of rights under discussion contains only one element, the set braces are normally omitted.

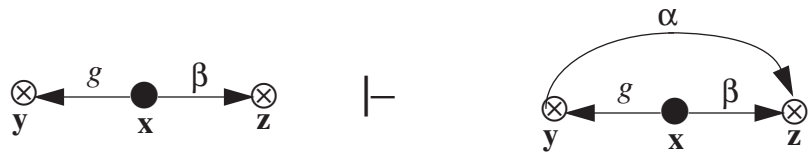
The Take-Grant Protection Model permits vertices with certain rights to transfer rights from one vertex to another. The rules governing the transfer of rights are called *de jure* rules and are as follows:

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 **y** to **z** labelled α. Graphically,



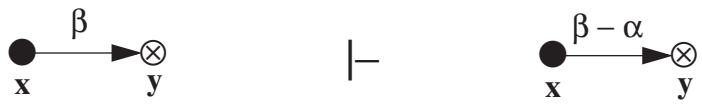
The rule is written "**x** grants (α to **z**) to **y**."

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. Further, and more formally, a vertex is an *actor* relative to the application of a graph rewriting rule if that rule requires that vertex to be a subject.

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 [15] 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* }.

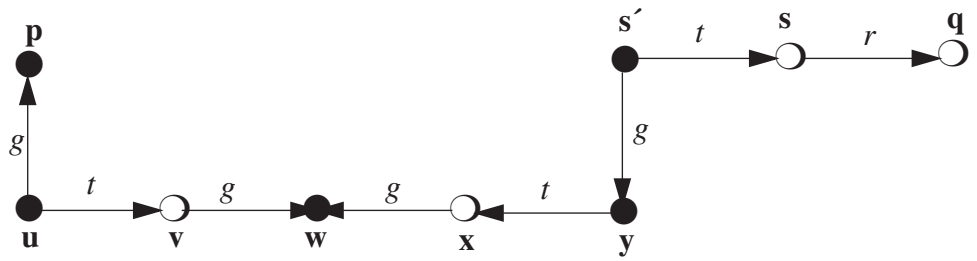
Under what conditions can rights be shared? To answer this question, we first need some definitions; for more exposition, the reader is referred to [15].

Definition. A *tg-path* is a nonempty sequence $\mathbf{v}_0, \ldots, \mathbf{v}_n$ of distinct vertices such that for all i, $0 \le i < n$, $\mathbf{v}_i$ is connected to $\mathbf{v}_{i+1}$ by an edge (in either direction) with a label containing *t* or *g*.

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

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

With each tg-path, associate one or more words over the alphabet in the obvious way. If the $\{ \vec{t}, \overleftarrow{t}, \vec{g}, \overleftarrow{g} \}$ path has length 0, then the associated word is the null word ν. The notation *t**



islands { **p**, **u** }, { **w** }, { **y**, **s´** } bridges: **u**, **v**, **w** and **w**, **x**, **y**

initial span: **p** (associated word ν) terminal span: **s´**, **s** (associated word ν)

Figure 1. Some examples of Take-Grant Terms. The reader is encouraged to find others.

means zero or more occurences of the character *t*, so for example *t*g* represents the sequence *g*, *tg*, *ttg*, ….

Definition. A vertex $\mathbf{v}_0$ *initially spans* to $\mathbf{v}_n$ if $\mathbf{v}_0$ is a subject and there is a tg-path between $\mathbf{v}_0$ and $\mathbf{v}_n$ with associated word in $\{ \vec{t}^*\vec{g} \} \cup \{ \nu \}$.

Definition. A vertex $\mathbf{v}_0$ *terminally spans* to $\mathbf{v}_n$ if $\mathbf{v}_0$ is a subject and there is a tg-path between $\mathbf{v}_0$ and $\mathbf{v}_n$ with associated word in $\{ \vec{t}^* \}$.

Definition. A *bridge* is a tg-path with endpoints $\mathbf{v}_0$ and $\mathbf{v}_n$ both subjects and the path's associated word in $\{ \vec{t}^* , \overleftarrow{t}^*, \vec{t}^*\vec{g}\overleftarrow{t}^*, \vec{t}^*\vec{g}\overleftarrow{t}^* \}$.

Figure 1 illustrates these terms. 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.

Theorem 3. [18] 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 **x´** 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, …, 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 in a witness has the form "**s** grants (α to **q**) to **z**" for any $\mathbf{z} \in G_j$ $(1 \leq j < n)$.

Theorem 4. [28] The predicate *can•steal*(α, **x**, **y**, G_0) is true if and only if the following hold simultaneously:

(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.

These rules apply only to the transfer of *rights*. But information may be transferred without any transfer of rights. Let us now examine this question.

3. Transfers of Information

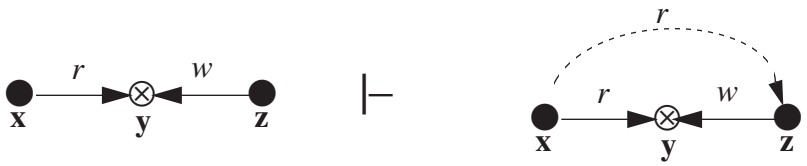
The *de jure* rules control the transfer of authority only; they say nothing about the transfer of information. Instead, we use a different set of rules, called *de facto* rules, to derive paths along which information may flow.

In order to describe transfers of information, we cannot use explicit edges because no change in authority occurs. Still, some indication of the paths along which information can be passed is necessary. Hence, 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 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 [5] to model transfers of information:

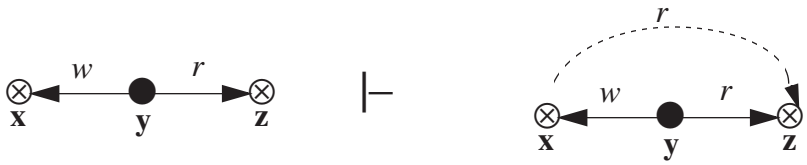
post: Let **x**, **y**, and **z** be three distinct vertices in a protection graph G_0, and let **x** and **z** be subjects. Let there be an edge from **x** to **y** labelled α with $r \in \alpha$ and an edge from **z** to **y** labelled β,

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,



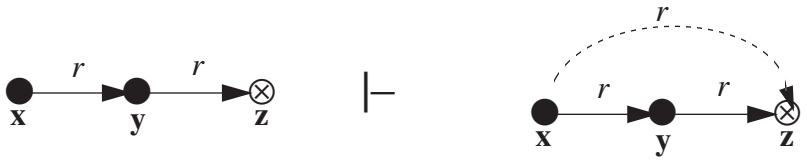
The rule is written "**z** posts to **x** through **y**," and is so named because it is reminiscent of **y** being a mailbox to which **z** posts a letter that **x** reads.

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 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**,for example by copying a file.

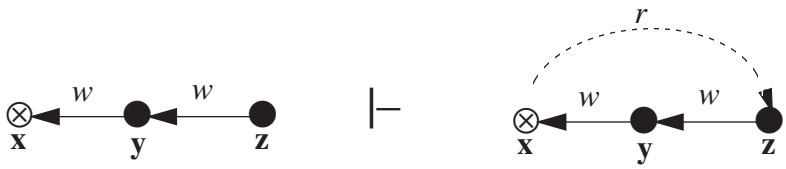
spy: Let **x**, **y**, and **z** be three distinct vertices in a protection graph G_0, and let **x** and **y** be subjects. Let there be an edge from **x** to **y** labelled α with $r \in \alpha$ and an edge from **y** to **z** labelled β, where $r \in \beta$. Then the *spy* rule defines a new graph 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**. For example, **x** reads **y**'s protected file **z** by reading the unprotected memory of a process with which **y** is displaying the contents of **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 *find*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," such as information sent by a user **z** to the mail server **y**, which then inserts it into **x**'s mailbox.

Note that these rules add implicit and not explicit edges. Further, as these rules model information flow, they *can* be used when either (or both) of the edges between **x** and **y**, or **y** and **z**, are implicit.

When can information flow from one vertex to another?

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.

Definition. An *rwtg-path* is a nonempty sequence $\mathbf{v}_0, \ldots, \mathbf{v}_n$ of distinct vertices such that for all *i*, $0 \le i < n$, $\mathbf{v}_i$ is connected to $\mathbf{v}_{i+1}$ 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 $\{ \vec{t}, \overleftarrow{t}, \vec{g}, \overleftarrow{g}, \vec{r}, \overleftarrow{r}, \vec{w}, \overleftarrow{w} \}$ in the obvious way.

Definition. A vertex $\mathbf{v}_0$ *rw-initially spans* to $\mathbf{v}_n$ if $\mathbf{v}_0$ is a subject and there is an rwtg-path between $\mathbf{v}_0$ and $\mathbf{v}_n$ with associated word in $\{ \vec{t}^*\vec{w} \} \cup \{ \nu \}$.

Definition. A vertex $\mathbf{v}_0$ *rw-terminally spans* to $\mathbf{v}_n$ if $\mathbf{v}_0$ is a subject and there is an rwtg-path between $\mathbf{v}_0$ and $\mathbf{v}_n$ with associated word in $\{ \vec{t}^*\vec{r} \}$.

Definition. A *bridge* is an rwtg-path with $\mathbf{v}_0$ and $\mathbf{v}_n$ both subjects and the path's associated word in the regular language $B = \{ \vec{t}^*, \overleftarrow{t}^*, \vec{t}^*\vec{g}\overleftarrow{t}^*, \vec{t}^*\overleftarrow{g}\overleftarrow{t}^* \}$.

Note this is the same as the definition given earlier.

Definition. A *connection* is an rwtg-path with $\mathbf{v}_0$ and $\mathbf{v}_n$ both subjects and the path's associated word in $C = \{ \vec{t}^*\vec{r}, \overleftarrow{w}\overleftarrow{t}^*, \vec{t}^*\vec{r}, \overleftarrow{w}\overleftarrow{t}^* \}$.

The next result [5] characterizes the set of graphs for which *can•know* is true:

Theorem 5. The predicate *can•know*(**x**, **y**, G_0) is true if and only if there exists a sequence of subjects $\mathbf{u}_1, \ldots, \mathbf{u}_n$ in G_0 ($n \geq 1$) such that the following conditions hold:

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

(5.2) $\mathbf{u}_n = \mathbf{y}$ or $\mathbf{u}_n$ rw-terminally spans to **y**;

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

Corollary 6. There is an algorithm for testing *can•know* that operates in linear time in the size of the graph.

Corollary 7. There is a witness to *can•know*(**x**, **y**, G_0) in which the vertices $\mathbf{u}_1, \ldots, \mathbf{u}_n$ in G_0 ($n \geq 1$) in Theorem 5 are actors, and no other vertex in G_0 is an actor.

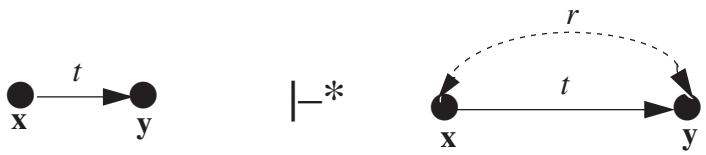
Proof: Immediate from the proof of Theorem 5 (see [5]). Note that other vertices may be created and *then* act, but these are not present in G_0.

In order to appreciate these results, let us now look at some examples of the uses of the rules and theorems; these will be useful in deriving our later results.

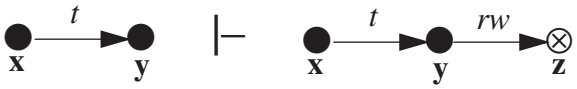
4. Some Examples of Combined *de jure* and *de facto* Rule Applications

In this section we present new results which are not only good examples of how the graph rewriting rules and the theorems in the previous sections are used, but also which will be quite useful in our later work. The first two results are quite basic, and state that if one subject has take or grant rights over another subject, either can (with the cooperation of the other) read information from the other. More formally:

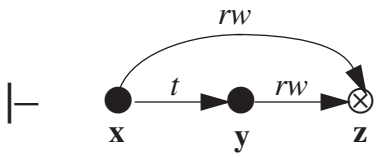
Lemma 8.



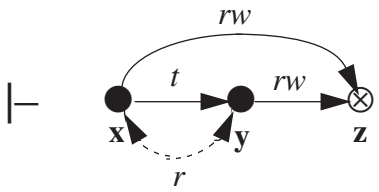
Proof: First, **y** creates (*rw* to new vertex) **z**:



Next, **x** takes (*rw* to **z**) from **y**:



Finally, **x** and **y** use the post rule through **z**:



Note that the direction of the implicit read edge depends on which rights **x** and **y** use. If **x** writes to **z** and **y** reads from **z**, the implicit edge (information flow) goes from **x** to **y**. If, on the other hand, **y** writes to **z** and **x** reads from **z**, the implicit edge goes from **y** to **x**. ■

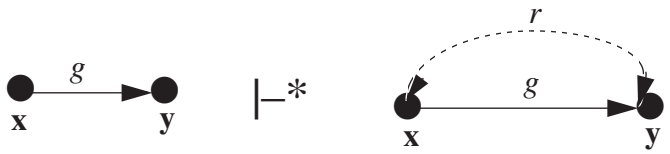
The next lemma shows that both **x** and **y** must act, or no information can be transferred:

Lemma 9. Let a vertex **x** have take rights over another vertex **y**, and assume the graph contains no subjects except (possibly) for **x** and **y**. If either **x** or **y** does not act, no sequence of graph transformations can add an implicit or explicit edge from **x** to **y**.

Proof: Assume first that **x** does not act. All *de jure* rules which add an outgoing edge require the source of that edge to be an actor (take, create) or have an incoming grant edge (grant). All the *de facto* rules which add an outgoing implicit edge require the source of that edge to be a subject (post) or have an incoming read (spy) or write (pass, post) edge. As the vertex **x** meets none of these requirements, it can never be given an implicit or explicit read edge to **y**.

Now assume **y** does not act. As there is no edge labelled *r* with **y** as its target, examination of the *de jure* rules shows no edge labelled *r* with **y** as its target can be added. Examination of the *de facto* rules shows that, in order to add an implicit edge labelled *r* into **y**, there must either be an incoming (explicit or implicit) edge labelled *r* (pass and spy rules), or **y** must be an actor (postand find rules). As neither of these conditions is met, no sequence of graph transformations can add an implicit or explicit edge from **x** to **y**. ■

Lemma 10.



Proof: An exercise for the reader.

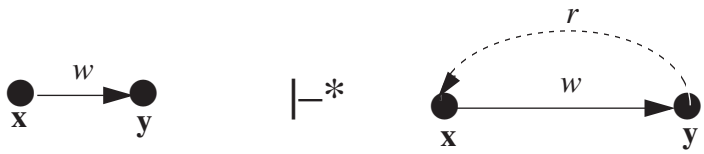
As in Lemma 9, both **x** and **y** must act, or no information can be transferred:

Lemma 11. Let a vertex **x** have take rights over another vertex **y**, and assume the graph contains no subjects except (possibly) for **x** and **y**. If either **x** or **y** does not act, no sequence of graph transformations can add an implicit or explicit edge from **x** to **y**.

Proof: An exercise for the reader.

The next case shows that information can flow backwards along a write edge:

Lemma 12.



Proof: An exercise for the reader.

Not surprisingly, both **x** and **y** must act, or no information can be transferred:

Lemma 13. Let a vertex **x** have write rights over another vertex **y**. If either **x** or **y** does not act, no sequence of graph transformations can add an implicit or explicit edge from **x** to **y**.

Proof: An exercise for the reader.

Later we shall consider the circumstances under which information can flow from a subject vertex **y** to another subject vertex **x**. To provide a basis for the results, let us consider the case of a protection graph G_0 with at least three vertices, namely **x** and **y**, and another vertex **z** which may be either a subject or an object. There is a path from **x** to **z** and a path from **y** to **z**; these are the only paths in the graph. (Note that these paths may include vertices other than their endpoints. However, we are assuming that except for such internal vertices, there are no vertices other than **x**, **y**, and **z**.) Furthermore, these paths may be initial, terminal, rw-initial, or rw-terminal (any combination is possible). Our problem is to derive witnesses to *can•know*(**x**, **y**, G_0) for those combinations of paths for which that predicate is true, and to prove that predicate is false for the others.

	x-to-**z** path			
y-to-**z** path	initial	terminal	rw-initial	rw-terminal
initial	maybe	true	false	maybe
terminal	true	maybe	false	maybe
rw-initial	maybe	maybe	false	true
rw-terminal	false	false	false	false

Table 1. Summary of values of *can•know*(**x**, **y**, G_0) in a 3-vertex graph G_0. The cases marked "maybe" are true if and only if **z** is a subject (and an actor).

Without loss of generality, we can assume that initial, terminal, rw-initial, and rw-terminal spans are all of length 1, because if the path is longer, all edges but the first are take edges and so by repeated applications of the take rule the vertex at the source of the directed path may obtain an edge with the rights of the last edge in the path.

First, if the path from **y** to **z** is rw-terminal and **z** is a subject, then the word associated with the **x** to **y** path is not in the set $B \cup C$ and so *can•know*(**x**, **y**, G_0) is false by condition (5.3) of Theorem 5; similarly, if **z** is an object, the word associated with the **x** to **y** path will not be in the set $B \cup C$ and, again, *can•know*(**x**, **y**, G_0) is false. If the path from **x** to **z** is rw-initial, similar reasoning shows again that *can•know*(**x**, **y**, G_0) is false whether or not **z** is a subject.

The other 15 combinations are presented in Table 1 (these can be determined by Theorem 5 and Lemma 11). For later reference, we state:

Lemma 14. If two subjects **x** and **y** in G_0 are connected by a bridge, then *can•know*(**x**, **y**, G_0) and *can•know*(**y**, **x**, G_0) are true.

Proof: Immediate from the definition of bridge, Lemma 8, Lemma 10, and the entries at the intersection of the "terminal" and "initial" rows and columns in Table 1. ■

Lemma 15. Let a subject **x** be connected by a bridge to another subject **y** in a graph containing exactly two subjects. If either **x** or **y** does not act, no sequence of graph transformations can add an implicit or explicit edge from **x** to **y**.

Proof: Immediate from the definition of bridge, Lemma 9, and Lemma 11. ■

Lemma 16. If two subjects **x** and **y** in G_0 are connected by a connection, then *can•know*(**x**, **y**, G_0) is true.

Proof: Immediate from the definition of connection and the entries in Table 1. ■

Now, let us consider the nature of the *can•know* predicate further:

Lemma 17. Let **x** and **y** be vertices in a protection graph G_0, and let $\mathbf{w}_1, \ldots, \mathbf{w}_n$ ($n \geq 1$) be a sequence of subjects in G_0. If *can•know*(**x**, $\mathbf{w}_1$, G_0) holds, *can•know*($\mathbf{w}_i$, $\mathbf{w}_{i+1}$, G_0) holds for $i = 1, \ldots, n-1$, and *can•know*($\mathbf{w}_n$, **y**, G_0) holds, then *can•know*(**x**, **y**, G_0) holds.

Proof: In Theorem 5, take $\mathbf{u}_1$ to be **x** or the subject $\mathbf{x}'$ in G_0 that rw-initially spans to **x** (such a subject exists because *can•know*(**x**, $\mathbf{w}_1$, G_0) is true); this gives condition (5.1). Also in Theorem 5, take $\mathbf{u}_n$ to be **y** or the subject $\mathbf{y}'$ that rw-terminally spans to **y** (as *can•know*($\mathbf{w}_n$, **y**, G_0) is true, such a exists). Finally, as the $\mathbf{w}_i$'s are all subjects and for $i = 1, \ldots, n-1$, *can•know*($\mathbf{w}_i$, $\mathbf{w}_{i+1}$, G_0) holds, there is an (implicit or explicit) path labelled *r* from $\mathbf{w}_i$ to $\mathbf{w}_{i+1}$ (by the definition of *can•know*). Hence $\mathbf{u}_{i+1} = \mathbf{w}_i$, $\mathbf{u}_1 = \mathbf{x}$ or $\mathbf{x}'$, and $\mathbf{u}_{i+2} = \mathbf{y}$ or $\mathbf{y}'$ in Theorem 5, establishing the result. ■

Lemma 18. Let *can•know*(**x**, **y**, G_0) be true. Then there exists a sequence of subjects $\mathbf{u}_1, \ldots, \mathbf{u}_n$ (with $n \geq 1$) such that $\mathbf{x} = \mathbf{u}_1$ or $\mathbf{u}_1$ rw-initially spans to **x**, $\mathbf{y} = \mathbf{u}_n$ or $\mathbf{u}_n$ rw-terminally spans to **y**, and *can•know*($\mathbf{u}_i$, $\mathbf{u}_j$, G_0) is true for all $i < j$.

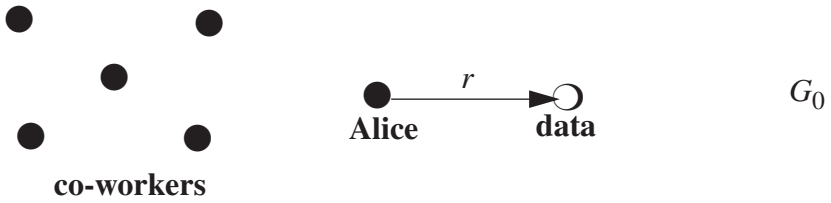
Proof: Conditions (5.1) and (5.2) of Theorem 5 give the subjects $\mathbf{u}_1$ and $\mathbf{u}_n$. By Theorem 5, there exist other subjects $\mathbf{u}_2, \ldots, \mathbf{u}_{n-1}$ such that there is a bridge or connection between $\mathbf{u}_i$ and $\mathbf{u}_{i+1}$ (by condition (5.3)). If a bridge, by Lemma 14 *can•know*($\mathbf{u}_i$, $\mathbf{u}_{i+1}$, G_0) holds; if a connection, by Lemma 15 *can•know*($\mathbf{u}_i$, $\mathbf{u}_{i+1}$, G_0) holds. The result follows immediately from Lemma 17.

5. Snooping, of the Theft of Information

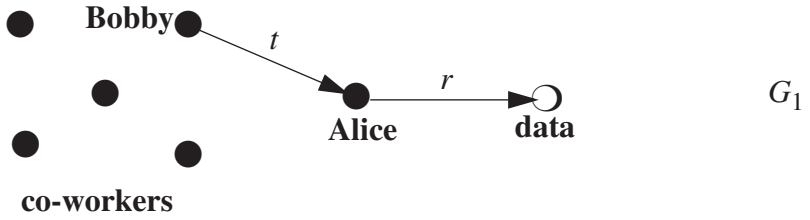
Up to this point, we have been considering cases where all vertices cooperate in sharing information, so all *de facto* rules may be applied with impunity. Suppose this is not true; suppose a vertex which has a right to read information from another vertex flatly refuses to pass this information along. Under what conditions can a vertex which does not have read rights over a second vertex obtain information from the second vertex?

An example will help show what the problem is. Suppose Alice works for a firm which has proprietary information that its competitors need desperately to see. Alice, who works with this information quite a bit, has the authority to read the documents containing the proprietary informa-

tion whenever she likes, with the understanding she is not to pass this sensitive data to anyone else, including co-workers and superiors. The situation, in Take-Grant terms, is:

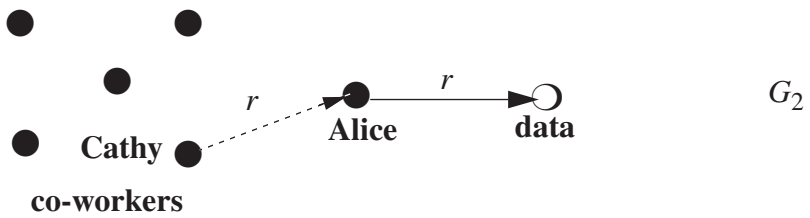


Any documents as sensitive as those which Alice consults must be kept under lock and key. Alice's company has several large vaults, and Alice has a key to one. To prevent employees with the keys from making copies, when an employee leaves the office, her key must be placed in her desk and the desk locked. To handle emergencies (such as Alice misplacing the key which unlocks her desk), a set of master desk keys is kept in the office of the chief of building security. The only person with a key to that office is the chief of building security, Bobby. Now, Bobby is not cleared to read any of Alice's documents, but he can "take" Alice's right to do so by opening her desk with his master key and copying her vault key. He can then pass the information on to someone else. This is an example of Alice's sharing (albeit unknowingly) her right to read the documents



(To make the analogy precise, the key to the vault is the "read right" and the key to the desk is the "take right," because with the latter key one can "take," or copy, the former key.)

Alice takes a sensitive document out of the vault, goes back to her desk, and begins to read the document. Normally, Alice is shielded from Cathy (who sits directly behind Alice in her office) by a partition which for this day only has been removed for repairs; hence Cathy can see what Alice is reading by looking over Alice's shoulder. In Take Grant terms, this situation is:



By the spy rule, Cathy can read the information Alice is reading (the Cathy-to-Alice edge, being unauthorized, is implicit); hence, *can•know*(**Cathy**, **data**, G_2) is true so long as Cathy can look over Alice's shoulder; if Alice read the document elsewhere, such as in the vault, Cathy would no longer be able to read the document over Alice's shoulder (so, in Take-Grant terms, the spy rule would not be applicable as there is no implicit or explicit Cathy-to-Alice read edge). Notice the difference between this case and the previous one: here, Alice must "cooperate" in some sense of the word by reading the data where Cathy could see it (such as at Alice's desk); were Alice to read it elsewhere, for example in the vault, Cathy could not see it. But so long as data is stored in the vault (a company requirement), Alice need not cooperate in any way with the building security chief in order for him to obtain the data, because by using his master key for the desks, not only does he have a copy of Alice's right to read, but also he can exercise that right whenever he wishes, regardless of Alice's presence. And the key to his office controls access to his copies of those other keys.

The *can•know* predicate fails to capture the distinction between these two situations. To embody the notion of "cooperation," we define a new predicate, called *can•snoop*. This predicate will be true if *can•know* is true and no-one who has any rights over the information being snooped for cooperates with the snooper. For example, *can•snoop*(**Cathy**, **data**, G_2) is false, since Alice must cooperate by passing the information to Cathy (in this case, by reading in such a way that Cathy can look over her shoulder). But *can•snoop*(**Bobby**, **data**, G_1) is true, since Bobby could see the documents whether or not Alice cooperated, once Bobby had "taken" the key to the vault.

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 exists a sequence of graphs and rule applications $G_0 \vdash_{\rho 1} \ldots \vdash_{\rho n} G_n$ for which all of the following conditions 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** in G_0 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.

Let us examine the definition more closely. The predicate is rather clearly the *de facto* analogue to *can•steal*, just as *can•know* is the *de facto* analogue to *can•share*. If **x** can steal read rights to **y**, clearly no-one who owns those rights over **y** can prevent **x** from obtaining information from **y**. Similarly, if **x** has authority to read **y**, it would strain the meaning of what we are trying to define to say that the predicate *can•snoop*(**x**, **y**, G_0) is true. If *can•steal*(r, **x**, **y**, G_0) is false, note

	cooperative	non-cooperative
rights	*can•share*	*can•steal*
information	*can•know*	*can•snoop*

Table 2. Summary of the relationship between predicates, cooperation, and information.

that any read edge from **x** to **y** in G_n must be implicit. And for the purposes of this discussion, we will assume that **y** will not cooperate (either wittingly or unwittingly) with any snooping; it would be equally reasonable to assume that **y** would cooperate, in which case what follows must be modified somewhat. (We shall return to this point later.)

Table 2 shows the relationship between information, cooperation, and the four predicates. We leave it as an exercise to the reader to show that *can•snoop*(**x**, **y**, G_0) , *can•steal*(r, **x**, **y**, G_0), and *can•share*(r, **x**, **y**, G_0) each imply *can•know*(**x**, **y**, G_0) .

Theorem 19. 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:

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

(19.2) there is a subject vertex $\mathbf{w}_1$ such that $\mathbf{w}_1 = \mathbf{x}$ or $\mathbf{w}_1$ rw-initially spans to **x** in G_0;

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

(19.4) *can•know*($\mathbf{w}_1$, $\mathbf{w}_n$, G_0) is true.

Informal argument: If *can•snoop* is true and *can•steal* false, we have to show all the conditions hold. Condition (19.1) follows from the definition. By part (b) of the definition, the predicate *can•know*(**x**, **y**, G_0) is true, giving condition (19.2). Also, by condition (5.2) of Theorem 5, there exists such a vertex $\mathbf{w}_n$. Combining this with the definition, it becomes clear that although $\mathbf{w}_n$ rw-terminally spans to **y**, $\mathbf{w}_n \neq \mathbf{y}$, and there is no edge labelled r from $\mathbf{w}_n$ to **y** in G_0. The proof that *can•know*($\mathbf{w}_1$, $\mathbf{w}_n$, G_0) is true involves proving that the first rule to add a read edge with target **y** is a take rule, and working backwards. (We should note that the latter technique is similar to the one used to prove the similar theorem for *can•steal* in [28].) Going from the conditions to *can•snoop* is trivial.

Proof: ($\Rightarrow$) Let *can•snoop*(**x**, **y**, G_0) be true. If *can•steal*(r, **x**, **y**, G_0) holds, we are done. So assume *can•steal*(r, **x**, **y**, G_0) is false.

Part (a) of the definition gives condition (19.1) of the theorem.

By part (b) of the definition, there is an implicit read edge from **x** to **y** in G_n, whence by definition *can•know*(**x**, **y**, G_0) is true. Consider the sequence of subjects $\mathbf{u}_1, \ldots, \mathbf{u}_n$ ($n \geq 1$) in Theorem 5. By Corollary 7, there is a witness W to *can•know*(**x**, **y**, G_0) in which the $\mathbf{u}_i$'s are the only actors present in G_0. So, taking $\mathbf{w}_1$ in condition (19.2) to be $\mathbf{u}_1$, that condition is established by condition (5.1). Note that by Corollary 7 we may without loss of generality take all vertices other than the $\mathbf{u}_i$'s to be objects.

As *can•snoop*(**x**, **y**, G_0) is true, the witness establishing that predicate is also a witness establishing *can•know*(**x**, **y**, G_0). Further, by part (c) of the definition, in this witness neither **y** nor any subject directly connected to **y** is an actor in a grant rule or a *de facto* rule resulting in a read edge with **y** as its target. Two cases now arise.

First, suppose **y** is an object, and look at the subject $\mathbf{u}_n$ in the witness W, above. This is the subject that rw-terminally spans to **y** in condition (5.2). Consider the nature of this rw-terminal span. If it is simply a read edge from $\mathbf{u}_n$ to **y**, by inspection of the *de facto* rules it will need to act to pass along information, thereby violating part (c) of the definition of *can•snoop*. Hence the rw-terminal span must involve one or more take edges followed by a read edge incident upon **y**; in this case, choose $\mathbf{w}_n$ in (19.3) to be $\mathbf{u}_n$.

Now, suppose **y** is a subject. Without loss of generality, we may take $\mathbf{u}_n = \mathbf{y}$ in (5.2); by theorem 8, this means **y** and $\mathbf{u}_{n-1}$ are connected by a path with associated word in $B \cup C$. If the word is in B, then by Lemma 15, $\mathbf{u}_{n-1}$ and **y** must both have been an actor in a rule application in W, contradicting part (c) of the definition. But if the word is in C, Lemma 11 says that **y** must be an actor (again contradicting part (c) of the definition) unless the connection is of the form $\bar{t}^* \bar{r}$. Thus, there is an rw-terminal span from $\mathbf{u}_{n-1}$ to $\mathbf{u}_n = \mathbf{y}$. This span must be used in our witness W, as any other mode of passing rights or information violates part (c) of the definition of *can•snoop*, so our witness would not be a witness to *can•snoop*, violating an assumption. By an argument similar to that when **y** is an object, this span must be of length at least 2; then $\mathbf{w}_n = \mathbf{u}_{n-1}$ in (19.3).

In either case, a $\mathbf{w}_n$ meeting the conditions in (19.3) exists.

It remains to be shown that *can•know*($\mathbf{w}_1$, $\mathbf{w}_n$, G_0) is true. Let $G_0 \vdash_{\rho 1} \ldots \vdash_{\rho n} G_n$ be a minimum length derivation sequence, and let i be the least index such that $G_{i-1} \vdash_{\rho i} G_i$, there is no (explicit or implicit) read edge from any vertex **z** to **y** in G_{i-1} not in G_0, and there is an (explicit or implicit) read edge from **z** to **y** in G_i that is not in G_0. That is, G_i is the first graph in which an edge labelled r with target **y** is added. Consider the rule ρ_i which caused this edge to be added. It cannot

be a grant rule since, by part (c) of the definition of *can•snoop*, the owner of *r* rights to **y** will not grant them to anyone else. The rule ρ_i cannot be a pass or spy rule, since by part (c) of the definition of *can•snoop*, the owner of *r* rights to **y** will not pass information from **y** to anyone else. Again by part (c), **y** will not write information from itself to anyone else, so ρ_i cannot be a post or find rule. As neither the create nor the remove rule adds edges to existing vertices, ρ_i cannot be either. Hence, ρ_i must be a take rule.

Let **z** and **z'** be the two other vertices involved in this take rule. We therefore have ρ_i: **z** takes (*r* to **y**) from **z´**, where **z´** is a vertex in G_{i-1}. By Theorem 5 we see that *can•know*($\mathbf{w}_1$, **y**, G_0) is true. Apply Theorem 5 again. By this theorem, there is a subject $\mathbf{u}_n$ such that $\mathbf{u}_n = \mathbf{y}$ or $\mathbf{u}_n$ rw-terminally spans to **y**. As there is no direct edge labelled *r* from $\mathbf{u}_n$ to **y** in G_0, then **z** must be the $\mathbf{u}_n$ in Theorem 5. As condition (19.3) holds, and an rw-terminal span has subjects only at the end, **z** is also the $\mathbf{w}_n$ in that condition. Hence $\mathbf{z} = \mathbf{u}_n = \mathbf{w}_n$, from which *can•know*($\mathbf{w}_1$, $\mathbf{w}_n$, G_0) immediately follows.

(⇐) If *can•steal*(*r*, **x**, **y**, G_0) holds, *can•snoop*(**x**, **y**, G_0) is true.

So, assume the other four conditions hold. Part (a) of the definition is the same as condition (19.1) of the theorem. By Theorem 5, conditions (19.2), (19.3), and (19.4) establish part (b) of the definition of *can•snoop*.

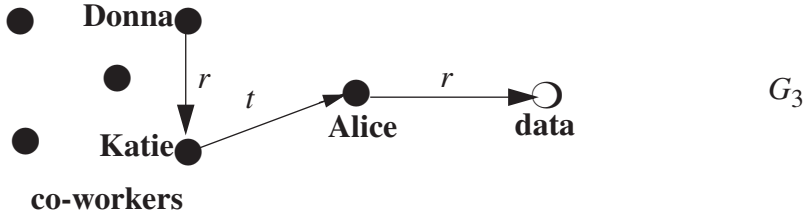
To prove part (c), we must show that there is at least one witness to *can•know*($\mathbf{w}_1$, $\mathbf{w}_n$, G_0) that does not require **y**, or any vertex directly connected to **y** in G_0, to be an actor. Let $\mathbf{w}_1$, …, $\mathbf{w}_n$ be the sequence of actors in Theorem 5; there is at least one such witness to *can•know*($\mathbf{w}_1$, $\mathbf{w}_n$, G_0) by Corollary 7. If $\mathbf{y} = \mathbf{w}_i$, or **y** is an object and $\mathbf{w}_i$ has read rights over **y**, for $i \neq n$, then simply apply Lemma 18 to derive an (implicit or explicit) edge from $\mathbf{w}_{i-1}$ to $\mathbf{w}_{i+1}$; clearly, this will not produce an (explicit or implicit) read edge with **y** as its target. By Lemma 18 again, *can•know*($\mathbf{w}_1$, $\mathbf{w}_{i-1}$, G_0) and *can•know*($\mathbf{w}_{i+1}$, $\mathbf{w}_n$, G_0) also hold, and by Corollary 7, there is at least one witness to each that does not involve $\mathbf{w}_i$; so *can•know*($\mathbf{w}_1$, $\mathbf{w}_n$, G_0) holds, and there is at least one witness to it that does not violate condition (19.3). So part (c) of the definition holds. ■

The proof technique used above is very similar to the one used to prove theorem 6, the main difference being the consideration of *de facto* rule applications. This emphasizes the similarity of the two predicates.

As an example, let us return to the office described at the beginning of this section. Consider G_1. By Theorem 4, *can•steal*(*r*, **Bobby**, **data**, G_1) is true, so *can•snoop*(**Bobby**, **data**, G_1) is also true by the above theorem. This conforms to our intuition; as noted earlier, it doesn't matter wheth-

er or not Alice cooperates with Bobby. However, in G_2, even though *can•know*(**Cathy**, **y**, G_0) is true, *can•steal*(*r*, **Cathy**, **data**, G_0) is false (specifically, in Theorem 4, taking **x´ = Cathy**, **s = Alice**, and **y = data**, condition (4.4) fails; taking **x´ = x = Cathy**, **y´ = Alice**, and **y = data**, condition (19.3) in Theorem 19 fails). So the predicate *can•snoop* captures the distinction between Alice's assisting in Cathy's seeing the information, and Bobby's seeing the information whether or not Alice cooperates.

However, *can•snoop* may be true, yet *can•steal* false. Consider the following situation:



G_3

Here, Alice is authorized to read the data, and her supervisor, Katie, is authorized to acquire any right Alice has. Katie's friend Donna is another manager on the same project, so has the right to read whatever Katie may pass to her. But Donna cannot acquire rights over Katie's subordinates. As condition (4.4) fails, *can•steal*({*r*}, **Donna**, **data**, G_3) is false; but if we take **x** = $\mathbf{w}_1$ = **Donna**, $\mathbf{w}_n$ = **Katie**, and **y** = **data** in Theorem 19, we see *can•snoop*(**Donna**, **data**, G_3) is true.

6. Applications of the Theory

Before we discuss security breaches, we should say a few words about the notion of a security policy. Breaches of security are defined in terms of a set of rules governing the use (and abuse) of a computer system. These rules, called the *security policy*, define what is allowed and what is not allowed. For example, a security policy may require that rights to a file be given only by the owner; in this case, if the owner mailed a copy of a protected file to another user, no breach of security has occurred even if the recipient had no right to see the file. The test of a system's security is how well it implements the policy.

In this section we consider three different security policies. The first is that of complete isolation of subjects; this technique is quite effective for solving the problem of transferring information and rights, but in most cases is far too restrictive. The second is that the owner of an object controls the dissemination of rights over that object; this is the policy exhibited by some database systems such as System R. The third is that no subject other than the owner may have any rights to

an object, and any information flowing to or from that object may do so only with the cooperation of the owner; resource managers often exhibit this property.

We consider first the statement of each policy in terms of the four predicates, then describe the set of graphs that satisfy that policy; finally, we construct protocols which preserve membership of the graphs in the set when the protocols are used.

6.1. 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 [17].

To prevent any information or rights transfer (illicit or otherwise) it suffices to make all four predicates always be false.The set of graphs satisfying this policy contains exactly those graphs h satisfying

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

for all pairs of vertices $\mathbf{x}$ and $\mathbf{y}$ in h, and all subsets of rights $\alpha \subseteq R$.

To characterize this requirement in terms of the rules, note that by Theorem 3 and Theorem 5, it suffices to prevent any bridges or connections between any two subjects to enforce this condition; in that case, both conditions (3.4) and (5.3) can never hold. Because the $\mathbf{x}$ and $\mathbf{y}$ referred to in the theorems may be subjects, it is not possible to prevent conditions (3.2), (3.3), (5.1), and (5.2) *a priori* from occurring; hence the above constraint is also necessary. This may be formalized to prove the following Take-Grant characterization of the set of graphs satisfying the policy:

Theorem 20. 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. 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:

Corollary 21. Testing a graph for violation of the restriction may be done in time linear in the number of edges of the graph. 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 (for example, when a new user is added to the system), 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. We do so in the manner of Snyder [26] by requiring all subjects to create other subjects using the following *subject creation protocol*:

subject creation protocol: A subject **x** desiring to create a subject **y** over which **x** has the set of rights α, **x** must execute the following rules atomically:

x creates (α to new subject **y**)

x removes ({ t, g, r, w } to) **y**

By enforcing this protocol, and preventing any node from applying the create rule directly to create a subject, complete isolation can be enforced.

We should note though that in practise such systems would be useless, since no process could communicate with another in any fashion. Even in systems where isolation is desired, it is typically used to prevent specific sets of processes from communicating; other processes may communicate freely. (For example, a system enforcing security levels would allow two processes at the same level to communicate freely, whereas one at a higher level could not transmit information to a process at a lower level.) So, most likely the "complete isolation" would be enforced between (for example) children of different subjects, and a different statement of the security policy would be needed. Hence, we turn from this somewhat uninteresting policy to consider one in which the holder of a right over an object determines to what other subjects it shall pass that right.

6.2. Transfer on Possession of a Right

This policy creates a system in which mere possession of a right enables a subject to propagate that right. Hence the set of graphs meeting this policy 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, 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 the other conditions of Theorem 4 negates one of of conditions (3.1), (3.2) or (3.3), making *can*•*share*(α,**x**,**y**,*h*) false. For (4.4) to be false, either there must be no bridges between any two subjects (by the definition of "island") or no take edges incident upon any subject. We note that the former renders *can*•*share*(α,**x**,**y**,*h*) false. But the latter renders *can*•*snoop*(**x**,**y**,*h*) false (as all rw-terminal spans will have length 1, so condition (19.3) fails) without necessarily negating *can*•*know*(**x**,**y**,*h*) or *can*•*share*(α,**x**,**y**,*h*). Clearly, this characterization is best, and from the above it can be shown:

Theorem 22. If no subject has any incident take edges, then the security policy allowing the possessor of a right to transfer that right to another is implemented.

Testing for this condition is simple:

Corollary 23. 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 subject creation protocol above. However, the lack of take edges also inhibits sharing; 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 requires the use of a take rule). Now consider a graph *h* meeting this security policy. As there are no take edges, the only take edges that could be added using the rules 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 for subjects by the modified subject creation protocol.

To overcome this problem, we can require that in the initial graph, all subjects have grant rights to one another, and modify the subject creation protocol to be:

granting subject creation protocol: A subject **x** desiring to create a subject **y** over which **x** has the set of rights α, **x** must execute the following rules atomically:

x creates (α to new subject **y**)
x removes ({ *t* } to) **y**
x grants ({ *g* } to all other subjects) to **y**
x grants ({ *g* } to **y**) to all other subjects

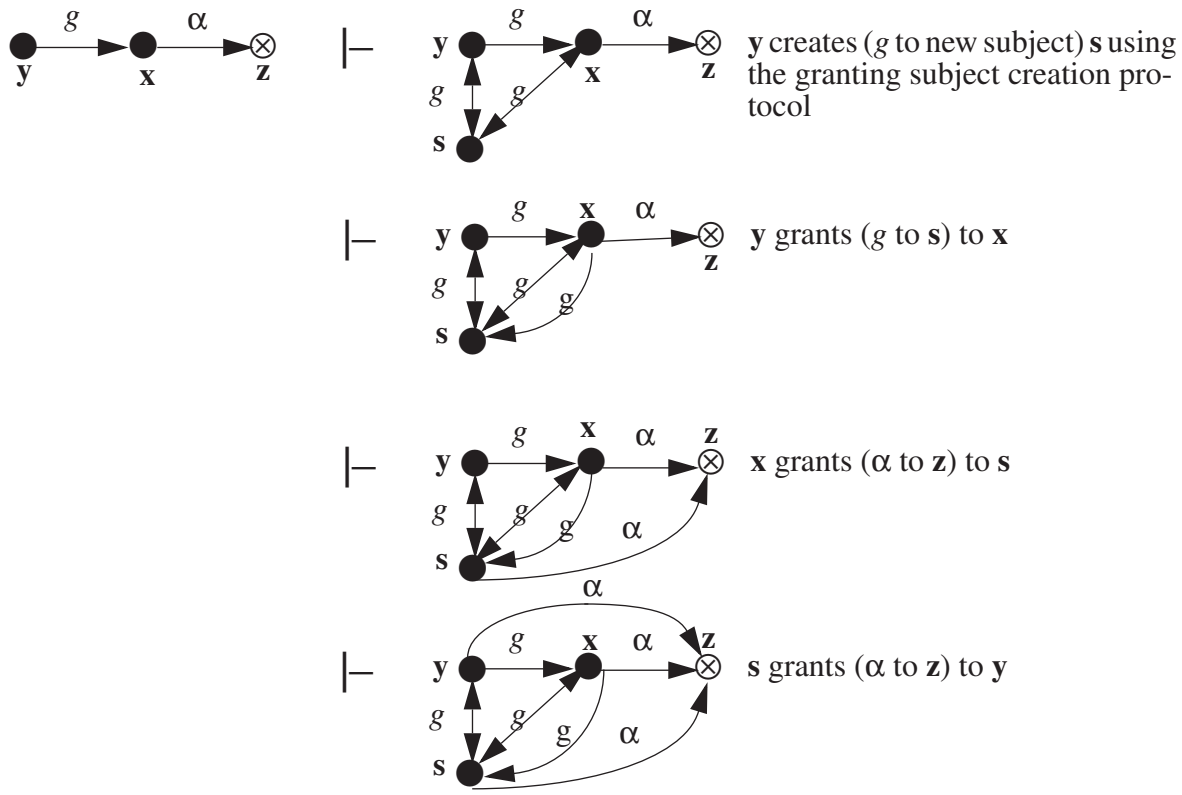


Figure 2. Proof of Lemma 2 using the granting subject creation protocol

If this protocol is used to create subjects, creating a new subject would add an incoming grant edge to the parent, in which case the proof of Lemma 2 is straightforward (see figure 2). Then 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. Hence the security policy will hold.

6.3. Owner Propagates Information but Not Rights

This policy creates a system in which only one subject is to have rights over a designated object. Those rights may not be transferred (with or without the consent of the subject). However, the subject may allow information to flow out of the object. An example of this policy would be a network server which responds to queries by giving information about the users active on a computer; the server has the right to obtain that information from the system and pass that data to an-

other process, but it cannot give any other process the right to obtain that information directly. The set of graphs meeting this policy is the set of graphs with elements satisfying

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

for all protection graphs *h,* any element **r** of the set of distinguished objects (called "resources"), any vertex **x** *h*, and all subsets of rights $\alpha \subseteq R$. The reasoning is that no rights to the resource may be transferred (hence the first two predicates must be false). To satisfy the latter two predicates, the owner must cooperate in any sharing of information.

If $can\bullet share(\alpha,\mathbf{x},\mathbf{r},h)$ is false, so is $can\bullet steal(\alpha,\mathbf{x},\mathbf{r},h)$, and without loss of generality we can consider only the former. Given that **s** in Theorem 3 is the unique subject (or monitor) **m** which possesses rights to **r**, (3.1) and (3.3) both hold; as **x** may be a subject, (3.2) holds. Hence (3.4) must be made to fail; the obvious way is to prevent there being bridges between **m** and any other subject. This means that no take or grant rights may be incident on **m** (unless they are between **m** and **r**).

The ability to transfer information is required; however, such transfers require **m** be an actor. So we must prevent the predicate $can\bullet snoop(\mathbf{x},\mathbf{r},h)$ from holding. As $can\bullet steal(\alpha,\mathbf{x},\mathbf{r},h)$ is false for all $\alpha \subseteq R$, we need only consider the conditions in Theorem 19. As only **m** has *r* rights over **r**, condition (19.1) holds; as **x** may be a subject, (19.2) may hold; but as no take edges may be incident on **m**, condition (19.3) always fails. So $can\bullet snoop(\mathbf{x},\mathbf{r},h)$ also fails. But all of the conditions in Theorem 5 hold, so $can\bullet know(\mathbf{x},\mathbf{r},h)$ holds, as required.

Finally, we note that as **m** rw-terminally spans to **r**, $\mathbf{r} \in A(\mathbf{m})$. Further, as no other vertex terminally, initially, rw-terminally, or rw-initially spans to **r**, the only way for information to flow out of **r** to some vertex **x** is through **m**. By the nature of the *de facto* rules, **m** must be an actor, as required. So the above formula accurately captures the security policy, and in Take-Grant terms, this discussion may be formalized to show:

Theorem 24. Let a subject **m** possess exclusive rights to a resource **r**. If there are no take or grant edges incident upon **m** (except possibly between **m** and **r** only), then no other vertex can acquire rights over **r**. Further, information may flow out of **r** only with **m**'s being an actor in a *de facto* rule.

Finally, we note that the same problems for complete isolation arise if **m** can create subjects; but if the following creation protocol is always used by **m**, the security policy will be enforced:

monitoring creation protocol: A monitor **m** desiring to create a vertex **y** over which **m** has the set of rights α, **m** must execute the following rules atomically:

m creates (α to new vertex **y**)

m removes ({ *t*, *g*} to) **y**

This preserves the condition that no take or grant edges be incident upon **m**.

6.4. Reference Monitors

The concept of a *reference monitor* was first described in [2] as a subject or process meeting three requirements:

1. the process must be *isolated* (that is, tamper-proof);
2. the process must be *complete* (that is, always invoked when the resource it controls is accessed); and
3. the process must be *verifiable* (that is, small enough to be subject to analyses and tests the completeness of which can be ensured).

We now restate these three conditions in terms of the Take-Grant Protection Model. Let the reference monitor be **m** and the resource it protects be **r**. Isolation means that no-one can write over the monitor; this may be stated as $\neg can\bullet share(\{w\},\mathbf{x},\mathbf{m},G_0)$ for all $\mathbf{x} \in G_0$. Completeness means that whenever any $\mathbf{x} \in G_0$ obtains access to the resource, then **m** is an actor in the witness to the access. Verifiability is a bit more tricky, since it consists of two parts. The first, an implementation-level task of verifying that the monitor is indeed implemented correctly, is beyond the scope of this paper; however, the second, which is a verification that no information will leak or rights be given away when the monitor is implemented correctly, is merely the condition in the preceding section.

Using that analysis, we may characterize what protection graphs containing reference monitors look like. Let G_0 be a protection graph with a single reference monitor **m** protecting a single resource **r**. Then no edge with a right in { *w* } may have **m** as its target and no edge with a right in { *t*, *g* } may be incident on **m** (with the possible exception of such an edge between **m** and **r**). As demonstrated in the previous section, the latter ensures information or rights may not be stolen from **m**, that **m** may not transfer rights, and that **m** must be an actor in any witness to a transfer of information. The former simply ensures no subject will ever be able to write to **m**, since no rule adds an edge labelled α unless there is already an edge labelled α already incident upon that vertex.

One general observation about modelling a reference monitor should be made. A reference monitor would be used to enforce rights indicated by the model; so the proof that the monitor is "secure" requires the assumption that the model is correctly implemented by the thing being mod-

elled. However, this tautology holds for *any* method used to model a system; if there is a mechanism that enforces rights within the system based upon the model, the model cannot be used to examine that mechanism for security. Also, a reference monitor is simply a special case of a resource manager, the differences being that reference monitors do more than simply control access rights because they also control access, and they control access to *all* objects in a system; so given a mechanism to enforce those rights, the above applies to resource managers guarding resources other than the access control graph (such as printers, the CPU, and so forth).

7. Conclusion

This paper has explored several aspects of information transfer in the Take-Grant protection model. The notion of information theft was developed, and then necessary and sufficient conditions to steal information were determined. Finally, as an application of this theory, the theoretical characteristics of reference monitors were expressed in terms of those predicates.

This has several ramifications. The first is that the Take-Grant protection model, while primarily a theoretical tool, can be used to model very practical concepts. In [4], the protection model was used to represent hierarchies and derive results paralleling the work done earlier by Bell and LaPadula [3]; now, the model can be used to capture the key notions of reference monitors and of several security policies. In short, the model can no longer be held to be a purely theoretical tool.

Some aspects of the model are particularly enticing in terms of applicability to real systems. For example, many models allow reflexive rights. The Take-Grant protection model does not. Correctly representing a real system in a theoretical model requires that all parts of the entity being modelled be represented as dictated by their attributes. Thus, a process would not be represented by a single subject; it would be a set of objects, some of which represent those parts of the process instructions and data resident in memory, others of which represent the resources held by the process, and so forth. The subject would represent the execution of the process. As the execution controls all the objects representing the process, it would have various rights over each. So if a process needed to access a part of its memory (say, to alter a variable which is shared with other processes), the subject would not need write permission on itself but rather write permission on the object representing that variable. Similarly, if certain portions of the process' memory were not writable (for example, the instructions making up the program being executed), the subject representing the process execution might have read rights, but not write rights, over that object. Hence the transfer of information does not appear to require reflexivity when a system is appropriately represented in a

theoretical model. Similarly, as no subject should be able to confer upon itself rights which it does not already have, reflexivity adds nothing to the modelling of transfer of authority. This intuition indicates the model does not suffer from being irreflexive; indeed, it forces the modeler to break the system being modelled into all component parts. Perhaps this would reveal unrealized assumptions or dependencies.

As an aside, we note that formulating a reflexive version of the Take-Grant Protection Model would alter its nature radically. Even though this reflexive version would be biconditional (and hence in a class in which, in the general case, safety is undecidable [12]), questions of safety would be decidable, as we pointed out in the introduction. In fact, if a subject had *take* rights over any other node, that subject would have all defined rights over that node. Further, if one subject had *grant* rights over another, that subject could give the target of the *grant* rights any authority over itself. While this would most likely not change the statement of the theorems giving necessary and sufficient conditions for sharing and theft, it would certainly change their interpretation (for example, the conditions in the theorems requiring the existence of a vertex **s** with an edge from **s** to another vertex **y** would be trivially true; just take **s** to be **y**). The precise effects of reflexivity, as well as its necessity, is an area in which further work is needed.

The rules for information transfer focus on reading, but by interchanging the read and write rights one could construct an equally logical set of *de facto* rules [5]. This would change the theorems involving the *de facto* rules considerably. The practical interpretation of those rules would also need to be addressed; in particular, given a real system, how does one know if the set of *de facto* rules completely captures all information flows in the system? The answer is not obvious and requires considerable research; suffice it to say that, for now, the rules used in this paper capture four of the most common techniques of transferring information without transferring rights.

In the definition of *can•snoop*, the target of the snooping was not to cooperate with the snooping. An alternate definition would be to allow the target to cooperate, but not the owners of the target; in this case, one could treat the target as a Trojan horse designed to "leak" information. Under this assumption, the proof presented for Theorem 19 does not apply (specifically, the rule ρ_i could be a *post* or *find* rule); this is not surprising, since condition (19.2) of that theorem is overly restrictive if **q** is a subject and allowed to act in a rule application. Another area for future work lies in the precise reformulation of the necessary and sufficient conditions for *can•snoop* to be true if the target of the snooping is allowed to act.

A related area is to incorporate a notion of "group" into the model. Changes of protection state in most computers do not affect a single process (subject) or resource (object); they affect several. However, within the Take-Grant protection model, each rule affects only one subject and one object (the source and target of the added implicit or explicit edge). How these rules might be modified to take these situations into account is another open area.

This leads to the following question: when the rules are changed to these "group rules," new theorems stating necessary and sufficient conditions for the predicates *can•share*, *can•steal*, *can•know*, and *can•snoop* to be true will have to be derived. It would be most useful if one could derive "metatheorems" instead, so that given a set of rules, one could use the metatheorems to state necessary and sufficient conditions for each of the predicates instead of having to rederive those results. This is yet another area for research.

Acknowledgments: This work owes much to Larry Snyder, who first interested me in the Take-Grant protection model and whose work on it has been the basis for many of the results presented here. Thanks also to Dorothy Denning, who suggested a line of research that caused me to consider the particular problem leading to the results presented her, and to the anonymous referees, Jonathan Millen, and Ravi Sandhu, whose diligence greatly improved the quality of this paper.

References

[1] P. Ammann and R. Sandhu, "Safety Analysis for the Extended Schematic Protection Model," *Proc. of the 1991 IEEE Symp. on Security and Privacy* (May 1991), 87-97.

[2] J. Anderson, "Computer SecurityTechnology Planning Study," Technical Report ESD-TR-73-51, USAF Electronics Systems Division, Bedford, MA (Oct. 1972).

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

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

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

[6] M. Bishop and L. Snyder, "The Transfer of Information and Authority in a Protection System," Technical Report #166, Deparement of Computer Science, Yale University (July 1979).

[7] D. Brewer and M. Nash, "The Chinese Wall Security Policy," *Proc. 1989 IEEE Symp. on Security and Privacy* (May 1989) 206-214.

[8] D. Clark and D. Wilson, "A Comparison of Commercial and Military Computer Security Policies," *Proc. of the 1987 IEEE Symp. on Security and Privacy* (Apr. 1987), 184-194.

[9] R. Feiertag and P. Neumann, "The Foundations of a Provably Secure Operating System (PSOS)," *Proc. of the National Computer Conference 48* (1979), 329-334.

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

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

[12] M. Harrison and W. Ruzzo, "Monotonic Protection Systems," in *Foundations of Secure Computing*, Academic Press, New York City, NY (1978), 337-366.

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

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

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

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

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

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

[19] E. McCauley and P. Drongowski, "KSOS – the Design of a Secure Operating System," *Proc. of the National Computer Conference 48* (1979), 345-353.

[20] J. McLean, "The Algebra of Security," *Proc. of the 1988 IEEE Symp. on Security and Privacy* (Apr. 1988), 2-8.

[21] J. McLean, "Security Models and Information Flow," *Proc. of the 1990 IEEE Symp. on Security and Privacy* (May 1990), 180-187.

[22] G. Popek, M. Kampe, C. Kline, S. Stoughton, M. Urban, and E. Walton, "UCLA Secure UNIX," *Proc. of the National Computer Conference 48* (1979), 355-364.

[23] R. Sandhu, "Expressive Power of the Schematic Protection Model (extended abstract)," *Proc. of the Computer Security Foundations Workshop*, Technical Report M88-37, The MITRE Corp., Bedford, MA (Jun. 1988), 188-193.

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

[25] R. Sandhu, "The Typed Access Matrix Model," *Proc. of the 1993 IEEE Symp. on Research in Security and Privacy* (May 1992) 122-136.

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

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

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

[29] J. Wittbold and D. Johnson, "Information Flow in Nondeterministic Systems," *Proc. of the 1990 IEEE Symp. on Security and Privacy* (May 1990), 144-161.

[30] M. Wu, "Hierarchical Protection Systems," *Proc. 1981 Symp. on Security and Privacy* (Apr. 1981), 113-123.