ON THE SYNTHESIS AND ANALYSIS OF PROTECTION SYSTEMS

Lawrence Snyder

Research Report #119

This work was supported in part by Office of Naval Research Grant N00014-75-C-0752.

On the synthesis and analysis of protection systems

Lawrence Snyder
Department of Computer Science
Yale University
10 Hillhouse Avenue
New Haven, Connecticut 06520

Abstract:

The design of a protection system for an operating system is seen to involve satisfying the competing properties of richness and integrity. Achieving both requires the interplay of analysis and synthesis. Using a formal model from the literature, three designs are developed whose integrity (with the help of the model) can be shown.

1. *Introduction*

In an enumeration of the many properties that a protection system should have, two distinguish themselves as being especially important:

richness - the property of admitting a complex variety of sharing relationships,

integrity - the property of guaranteeing that the protection system cannot be compromised even in the most hazardous of circumstances.

Both properties are crucial - a rich system with dubious integrity is just as unacceptable as a system of unassailable integrity with only meager sharing facilities.

The task of achieving both richness and integrity in a protection system is difficult because the two properties are contradictory. For every feature, restriction, exception, etc., added to achieve richness during the *synthesis* phase of design, a complication is usually introduced into the *analysis* phase of validation. Traditionally, there seems to have been too much emphasis on synthesis at the expense of analysis. This partly explains why clever systems are so often compromised. It is one purpose of this report to demonstrate that *richness and integrity can be achieved by allowing the analysis to guide the synthesis.*

That analysis should play a leadership role may at first seem curious, since to have anything to analyze something must first have been synthesized. But this is the *ad hoc* view of the problem - where each completed design is analyzed separately, from scratch and without the benefit of general guiding principles. What is being proposed here is a more general view -- where analysis establishes general properties of a whole class of designs based on a formal model of protection while synthesis amounts to selecting among the designs. This view will be exhibited by performing both analysis and synthesis on the Take-Grant Model [1,2]. Specifically, the analysis (started in [1,2]) will be extended to establish new, stronger conditions on when sharing can be performed. Then three designs will be selected in order to exhibit the synthesis activity.

A second objective of this report is to demonstrate that a particular Model, the Take-Grant Model [1,2], is a suitable model of capability protection in that it admits designs that have both richness and integrity. Most of the work to establish integrity appears elsewhere [1,2] and will only be mentioned here. It is *not* obvious that the Take-Grant system has richness -- indeed, from the earlier papers [1,2] one might conclude that it is *impoverished*. It will be shown (section 4) that the Take-Grant system has rich instances.

The remainder of the paper is structured as follows: section 2 gives an introduction to the Take-Grant Model that parallels earlier work (and can be skipped by those familiar with [1,2]). Section 3 discusses the apparent short-comings of the Take-Grant model and establishes a new condition for sharing. Section 4 presents three different protection system designs -- each with a different cost assumption -- in order to exhibit the synthesis of rich systems. The final section is reserved for discussion and suggestions for future research.

2. *Graphical Model of the Take-Grant System*

In this section the Take-Grant protection model of [1] is described (with some minor modifications*). In order to focus on the role that the model plays in synthesizing and analyzing protection systems, the Take-Grant model will be initially presented in purely formal (though quite intuitive) terms. The *interpretation* as a protection model will be postponed until section 2.3.

2.1 *Constituents of the Take-Grant Model*

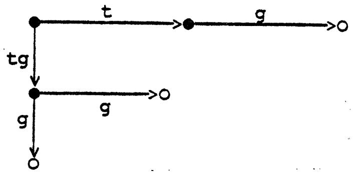
The *state* of a Take-Grant Protection system is a finite, directed, edge labeled graph called a *protection graph*. There are two types of vertices in the protection graph, *subjects* and *objects*. (Notationally, filled circles, ●, will

* For those familiar with the earlier version of the model, "t" and "g" labels are used instead of "r" and "w", respectively. The "call" operation has been dropped from consideration and "remove" has been weakened. None of these changes substantially effects the earlier work.

This work was supported in part by Office of Naval Research Grant N00014-75-C-0752.

denote subjects, unfilled circles, ○, will denote objects, and crossed circles, ⊗, will denote either subjects or objects.) The labels on the edges are called *rights* and are either {t}, {g}, {t,g} where "t" and "g" are mnemonic for "take" and "grant."[†]

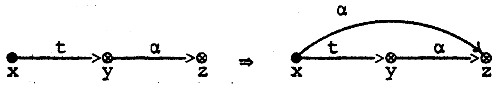
Example 2.1: A protection graph with three subjects and three objects.



A protection graph G is modified to G' by means of rewriting rules. Rules have the form $\alpha \Rightarrow \beta$. When α matches some subgraph of G, the rule can be *applied* to G, producing a new graph G' (the operation of applying a rule r is written $G \vdash_r G'$).

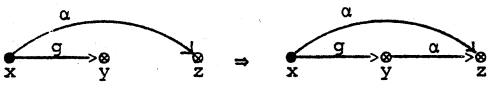
There are four rewriting rules in the Take-Grant Model:

Take: Let x, y, and z be three distinct vertices in a protection graph G such that x is a subject. Let there be an edge from x to y labeled γ such that "t" $\in \gamma$, and an edge from y to z labeled α.[††] Then the *take* rule defines a new graph G' by adding an edge to the protection graph from x to z labeled α. Graphically,



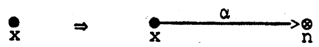
The rule can be read: "x takes (α to z) from y."

Grant: Let x, y, and z be three distinct vertices in a protection graph G such that x is a subject. Let there be an edge from x to y labeled γ such that "g" $\in \gamma$, and an edge from x to z labeled α. The *grant* rule defines a new graph G' by adding an edge from y to z labeled α. Graphically,



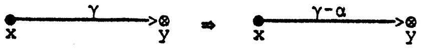
The rule can be read: "x grants (α to z) to y."

Create: Let x be any subject vertex in a protection graph G and let α be a subset of rights. *Create* defines a new graph G' by adding a new vertex n to the graph and an edge from x to n labeled α. Graphically,



The rule can be read: "x creates (α to) new $\left\{\begin{smallmatrix}\text{subject}\\ \text{object}\end{smallmatrix}\right\}$ n."

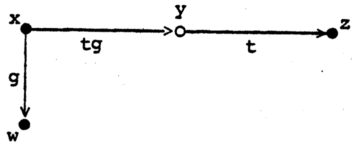
Remove: Let x and y be any distinct vertices in a protection graph G such that x is a subject. Let there be an edge from x to y labeled γ, and let α be any subset of rights. Then *remove* defines a new graph G' by deleting the α labels from γ. If γ becomes empty as a result, the edge itself is deleted. Graphically,



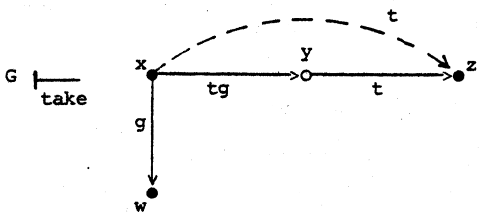
The rule can be read: "x removes (α to) y."

Notice that in the case of *take* and *grant* if the edge which is to be added already exists, the label α is simply unioned with the label presently assigned to the edge.

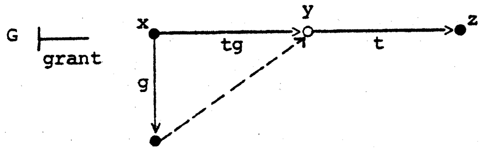
Example 2.2: Let G be the protection graph



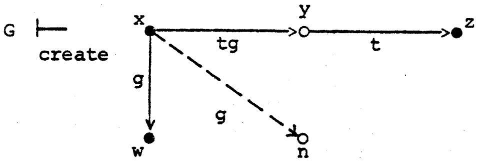
then the four rules can be exhibited[*] as follows:



x takes (t to z) from y;



x grants (tg to y) to w;

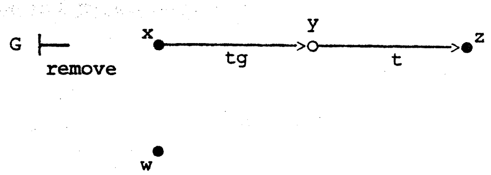


x creates (g to) new object n;

[†] We will generally elide the braces around sets.

[††] In the rules, α is a variable representing any of the three possible labels, t, g, and tg.

[*] The dashed lines have no special meaning, they are only a visual aid to indicate the added edge.



x removes (g to) w.

2.2 *Properties of Protection Graphs*

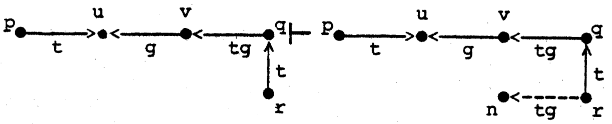
The primitive facilities provided by the model are uncomplicated, but their interactions are unexpectedly complex. Accordingly, the analysis of the model will be accomplished in two steps. First to be presented is a statement of the sharing relationships realizable in graphs containing only subjects (2.2.1). Then, the sharing relationships realizable in a general graph can be stated in terms of "subject islands" (2.2.2).

2.2.1 *Acquiring Rights in Subject-Only Graphs*

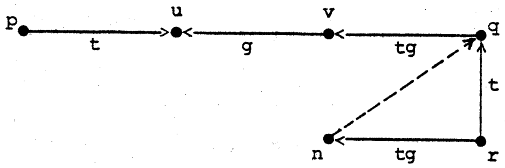
Two vertices, p and q, are said to be *connected* if there is a path between them without regard to directionality. The vertices p and q are *subject-connected* if they are connected by a path whose vertices are only subjects. If α is a label and p and q are vertices in a protection graph G_0 then *p can α q* means that there exists a sequence of graphs $G_0, G_1, \ldots, G_n$ each derived from its predecessor by one of the four rules (i.e., $G_0 \vdash G_1 \vdash G_2 \vdash \ldots \vdash G_n$) and in G_n there is an edge from p to q with label α.

Theorem 2.1: [2] Let p, q and r be subject vertices in a protection graph such that there is an edge from r to q labeled α. Then p can α q if p and q are subject-connected.

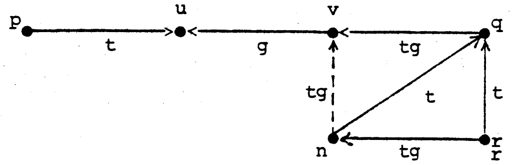
The proof of (a somewhat stronger version of) this theorem is given in [2]. The theorem can very easily be misinterpreted, so any discussion of the result is postponed to the next section. For the present, an example may help to suggest some of the unexpected consequences of the theorem.



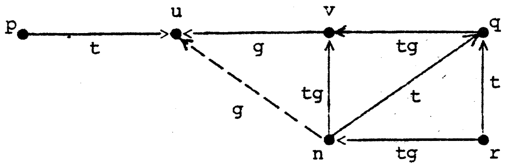
r creates (tg to) new subject n



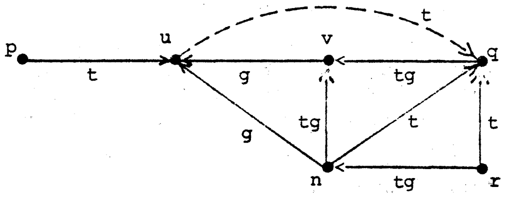
r grants (t to q) to n



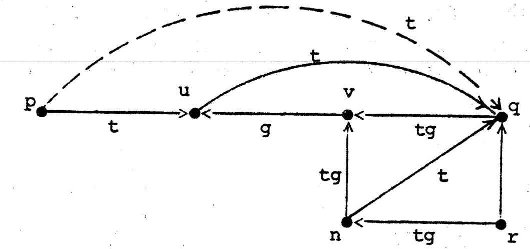
n takes (tg to v) from q



n takes (g to u) from v



n grants (t to q) to u



p takes (t to q) from u. □

2.2.2 *Acquiring Rights for the Subject-Object Case*

Complex as the example from the last section may appear, it only involves subjects -- the easy case! Objects create further complications to be dealt with in this section. (This section may be skipped on the first reading.)

A *block* in a protection graph G is any maximal subject-connected subgraph. Let p and q be subjects and $x_1, \ldots, x_n$ ($n \geq 1$) be objects such that

(2.1) p directly connected to x_1

x_i directly connected to x_{i+1} $\quad 1 \leq i \leq n-1$

x_n directly connected to q,

then $p, x_1, x_2, \ldots, x_n, q$ is a *path*. With each such path associate a *word* over the alphabet

$$\{ \vec{t}, \vec{g}, \overleftarrow{t}, \overleftarrow{g} \}$$

where letters correspond to edge labels in the obvious way, e.g., $\circ\xrightarrow{t}\circ$ is represented by $\vec{t}$, and $\circ\xrightarrow{t}\circ\xrightarrow{t}\circ\xleftarrow{g}\circ\xrightarrow{tg}\circ$ is a path associated with the two words $\vec{t}\,\vec{t}\,\overleftarrow{g}\,\vec{t}$ and $\vec{t}\,\vec{t}\,\overleftarrow{g}\,\vec{g}$.

Let E be the union of the regular languages defined by

(2.2) $\vec{t}(\vec{t})^+$

$\overleftarrow{t}(\overleftarrow{t})^+$

$$(\vec{t})^{*}\vec{g}(\overleftarrow{t})^{+}$$
$$(\vec{t})^{*}\overleftarrow{g}(\overleftarrow{t})^{+}$$
$$(\vec{t})^{+}\vec{g}(\overleftarrow{t})^{*}$$
$$(\vec{t})^{+}\overleftarrow{g}(\overleftarrow{t})^{*}$$

where $A^{+} = AA^{*}$ for any set A. A *bridge* between two blocks exists if from some subject p in one block there is a path with associated word in E to subject q in the other block.

Theorem 2.2: [1] Let G be a protection graph, p, q and r subjects such that there is an edge from r to q with label α. Then p can α q if and only if the there exists a sequence of blocks $B_1,\ldots,B_k$ with p in B_1 and q in B_k and for i=1,...,k-1 there is a bridge from B_i to B_{i+1}.

The proof of this theorem is also found in [1]. Notice that if the protection graph only contains subjects (and thus k=1), then Theorem 2.2 strengthens Theorem 2.1 to be "if and only if."

Example 2.4: Let a protection graph G have the schematic form of figure 2.1, where A and B are blocks, i.e., maximal subject-connected subgraphs. A path between p and q is shown, and since its associated word is in the "last" component set of E, (i.e., $\vec{t}\,\vec{t}\,\vec{t}\,\overleftarrow{g}\,\overleftarrow{t} \in (\vec{t})^{+}\overleftarrow{g}(\overleftarrow{t})^{*}$), the path is a bridge. By the theorem, p can t q as the following rule applications indicate:

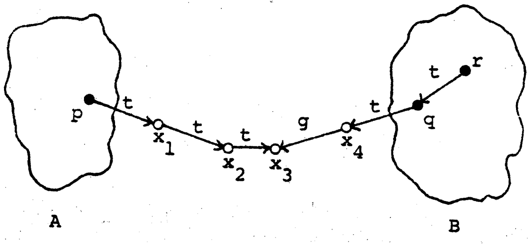
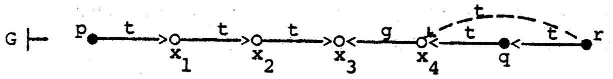
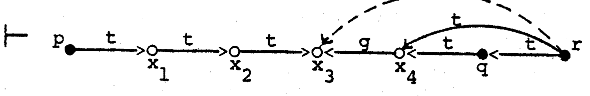


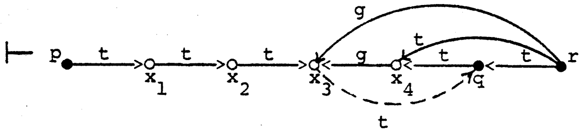
Figure 2.1: A protection graph with two blocks, A and B.



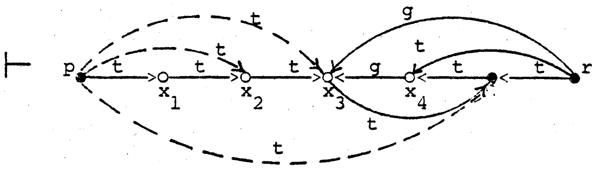
r takes (t to x_4) from q



r takes (g to x_3) from x_4



r grants (t to q) to x_3



p takes (t to x_2) from x_1

p takes (t to x_3) from x_2

p takes (t to q) from x_3. □

2.3 Interpretation of the Take-Grant Model

So far in this section, the only indication that the Take-Grant model is intended for use in studying protection has been some suggestive vocabulary. The objective now is to relate the graph-theoretic development to protection.

It is assumed that the protection system (which is to be modeled by the Take-Grant model) is a logically separate entity from the operating system "supervisor" (and thus the supervisor is subject to its limitations like any other process.)* In particular, the independence of the protection system allows the user to query the system himself for an *audit* to verify that certain protection conditions hold. The protection graph is a description of the currently extant protection relationships. Thus, the protection relationships among systems entities can be changed *only by the four rules*. The subjects are generally thought to be "user processes" or components that are "active" from a protection point of view, while the objects are thought of as files or processes "known" to be secure. When a subject "applies" a rule (notice that only subjects can "apply" the rules) it is requesting a modification of the protection state. *Take* causes a user to acquire a new right over some systems entity while *grant* gives some right away. *Create* enables new processes and files to have their protection configuration added to the system structure while *remove* eliminates rights.

There are several important characteristics that should be noted about the rules of the model.

(2.3) a. *Take* and *grant* do not create any new rights -- they merely disseminate existing rights.

b. Once all rights to a subject or object have been *removed* they can never be restored.

* Here operating system is the totality of the non-user programs while "supervisor" refers to the monitor program.

c. Rights once *granted* away can never be recovered, i.e., they can be distributed by the grantee without consultation with the grantor.

Finally, it is significant that it can be efficiently determined whether or not p can α q. In contrast to [3] where the "safety question" is either undecidable or prohibitively expensive, there is a linear time algorithm to decide if p can α q [1], i.e., to determine if the conditions of Theorem 2.2 are satisfied. Thus audits can be efficiently performed.

3. Evaluation of the Take-Grant Model

In the last section the Take-Grant model was introduced in purely formal terms and the constituent parts of the model were correlated with protection systems components (e.g., subject vertices represent user processes). What has not been done is to interpret the theorems from the protection viewpoint. That is the first undertaking of this section. After that, we discuss why the model appears to provide such meager facilities. Finally, conditions are set forth that permit richer sharing relationships.

3.1 What Does the Take-Grant Model Allow?

Consider the protection state schematically shown in figure 3.1 where the region represents the totality of all subjects and objects. Subjects A and B are to be thought of as users while objects w, x, y, and z are thought of as their private files.

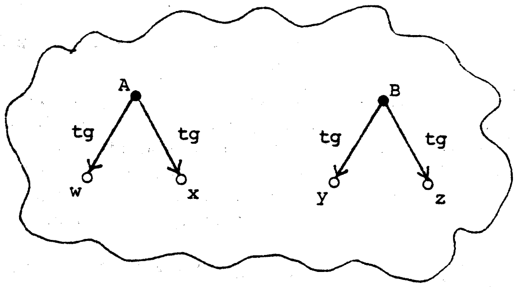


Figure 3.1: Schematic Protection State

Given the following list of objectives, Theorem 2.2 can be used* to specify the needed requirements to achieve these objectives and to infer the potential consequences.

(3.1) *A wants to prevent any subject from getting rights to* w. In order to achieve this, A either must not be connected to any subject or else if it is connected it must be connected by a path with associated word not in E. One consequence of such a relationship is that A cannot share with anyone else if it is so isolated. Apparently, protection prevents sharing.

(3.2) *A wants to have access to* B*'s file* y. A must be subject-connected to B (or connected by a path with associated word in E). If A and B are so connected then B can acquire rights to A's files w and x. It seems that sharing forfeits protection.

(3.3) *A wants to share its file* w *with* B *and also to prevent* B *(or any other subject) from acquiring rights to* x. If B can tg w then B can tg x. Thus, the only way that A can protect x is to remove its rights to x. But by observation (2.3b) A can never reacquire the rights to x! It appears as though rights cannot be selectively protected.

3.2 A Postmortem

Theorem 2.2 provides an exact characterization as to when a subject can acquire a right in the Take-Grant Model. When the characterization is applied to solve some typical protection problems (3.1-3.3), however, the model ***appears*** to be disappointingly weak. But appearances can be deceiving. So, before dismissing the model as analyzable but so poorly endowed with sharing facilities as to be useless, it is prudent to review the model and theorem checking that each feature is realistic and correct.

Our review has exposed two characteristics of the model worthy of revision -- the reader may discover others. In particular, the model permits two parameters to vary arbitrarily which in reality are far more constrained. Moreover, we will find that constraining these parameters enriches the model.

The first parameter that is underconstrained is the protection graph G in Theorem 2.2. Recall from section 2.2.2 that the Theorem states:

"Let G be a protection graph...."

This permits the protection configuration to be chosen arbitrarily. But in reality, the protection state of an operating system, which the graph G is supposed to model, will probably be derived from a specific initial state by a certain set of sharing protocols. For example, the protection state at system initialization time may only include the supervisor (probably a subject) and some service files and routines (probably objects) that the supervisor has tg rights to. Furthermore, the supervisor is likely to create new users (new subjects) by means of a fixed protocol, the compilers and editor will create files by fixed protocols, etc. Hence, the protection state will not be an arbitrary graph even if a particular user is permitted unlimited freedom to apply protection rules. (Section 4 is devoted to formulating a useful set of creation and sharing protocols.)

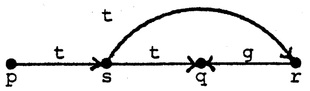
The second, and in our opinion the most important, underconstrained parameter of the model is found in the definition of *p can* α *q*. Recall (section 2.2.1) that p can α q means that

> *there exists* a sequence of graphs $G_0, G_1, \ldots, G_n$ each derived from its predecessor by one of the four rules and in G_n there is an edge from p to q with label α.

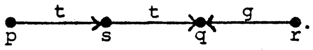
An example will help to distinguish between the

* Our use of Theorem 2.2 in (3.1-3.3) requires that the letter "q" in the statement of the theorem correspond to a file, however, it is required by the theorem that "q" be a subject. So, as stated Theorem 2.2 does not apply. The reader can easily prove, however, that "q" may be either a subject or an object in the theorem, and hence our conclusions here are valid.

provisions of this definition and the requirements of statements such as (3.1-3.3). Consider the graphs

(3.4) 

and

(3.5) 

In both cases p can g q, since in (3.4)

(3.6) p takes (t to r) from s
p takes (g to q) from r

is all that is required and in (3.5) the sequence

(3.7) r creates (tg to) subject n
r grants (g to q) to n
r grants (t to n) to q
p takes (t to q) from s
p takes (t to n) from q
p takes (g to q) from n

defines the appropriate graph.

Let us call a subject an *initiator* of a rule if that subject corresponds to the vertex "x" in the rule definition of section 2.1. Thus, in the second example r initiates the first three rules and p initiates the second three.

The notable distinction between (3.6) and (3.7) (besides length) is that r is not an initiator in (3.6) but it is in (3.7). Now, suppose r wishes to protect the g right to q. In (3.4) it cannot prevent p from acquiring the right since p can initiate the needed action without r's assistance. But in (3.5) *if r never initiates a rule then no matter what rules p, q, and s initiate, p can never acquire the g right to q.* The conclusion is that p can be *given* g to q in either case; p can *steal* g to q in only the first case (3.4).

The distinction, then, is between the possibility of a right being given away and the possibility of a right being stolen. Theorem 2.2 only addresses giving rights assuming everyone to be infinitely generous. Since it seems realistic to assume that users will not "knowingly"[*] give away what they wish to protect, a right that cannot be stolen is protected. In the last part of this section a theorem similar to Theorem 2.2 details the conditions under which p can steal α to q.

3.3 Stealing in the Take-Grant Model

The distinction to be made is between the case where p must acquire a right with the assistance of one of the "owners" of the right and the case where p is able to acquire that right without any owner's assistance. If the latter case applies p will be able to "steal" the right.

More precisely,[†] let G be a protection graph, p, q be subjects and α a label such that $p \not- \alpha \rightarrow q$ in G. Then *p can steal α to q in G* if there exists a sequence of graphs

$$G \vdash_{r_1} G_1 \vdash_{r_2} G_2 \vdash_{r_3} \cdots \vdash_{r_n} G_n$$

such that

(i) $p - \alpha \rightarrow q$ in G_n

and

(ii) for all i, x_i initiates rule r_i implies

$x_i \not- \alpha \rightarrow q$ in G.

Thus, p can steal α to q if it doesn't already have α rights to q, if p can α q (i) and if it can do so without any owner of the α right to q initiating any rules, i.e., cooperating with the dissemination of its right. The intuition we seek is if p can α q but p cannot steal α to q then the α right to q must necessarily be protected and p can only acquire it as a *gift.*

The circumstances under which rights can be stolen may now be characterized.[*]

Theorem 3.1: Let G be a protection graph, p, q are subjects and α is an edge label. Then p can steal α to q if and only if

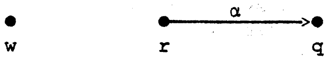
(i) p can α q

and

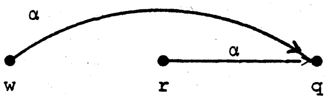
(ii) there exist subjects r and s such that $s - t \rightarrow r$ and $r - \alpha \rightarrow q$.

Proof: (=>) Suppose p can steal α to q, then by definition of "can steal," p can α q, so condition (i) is satisfied. Moreover, by definition of p can α q, there is some r such that $r - \alpha \rightarrow q$. So suppose (for purposes of contradiction) that for all r such that $r - \alpha \rightarrow q$ that there is no s such that $s - t \rightarrow r$.

Let $G \vdash_{r_1} G_1 \vdash_{r_2} \cdots \vdash_{r_n} G_n$ be a sequence that implements p can steal α to q. Let i be the smallest integer such that some vertex w receives the α right to q (i.e., this is the first graph in which the α right to q is disseminated -- this exists by definition of "can steal"). Thus, in G_{i-1} the situation is



and in G_i the situation is



By examining the rules, it is clear that r_i is either "take" and w corresponds to "x" in the rule definition, or "grant" and r corresponds to "x" in

[*] The word "knowingly" is used here only in the limited sense that by requesting an audit, the user can find (and thus "know") whether the initiation of a particular rule will materially contribute to a theft.

[†] The notation $p - \alpha \rightarrow q$ (resp. $p \not- \alpha \rightarrow q$) in G means that there is (resp., is not) an edge in G from p to q labeled α. The "in G" will be elided when it is clear from context.

[*] As usual, we treat only the case where the "agents" of the theorem are subjects. The cases where some nonempty subset of p, q, r and s are objects has not been addressed.

the rule definition. The second case is eliminated since r is the initiator which isn't allowed in a "steal." So r_i must be a take initiated by w and hence w must have take rights to r. By Theorem 2.2, however, for w to get t rights to r, some subject must have t rights to r, contradicting the assumption.

(<=) Suppose that p can α q, $s \xrightarrow{t} r$ and $r \xrightarrow{\alpha} q$. If p can α q without r initiating a rule, then the theorem is proved. So suppose that p can α q and r initiates some rules. We construct a surrogate subject, a new subject n, identical to r. Then the sequence of rule applications implementing p can α q can be rewritten with n replacing all occurrences of r.

The process to be defined will begin with a graph such as that schematically shown in figure 3.2 and will produce the graph schematically shown in figure 3.3.

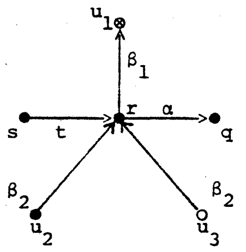


Figure 3.2: Given configuration.

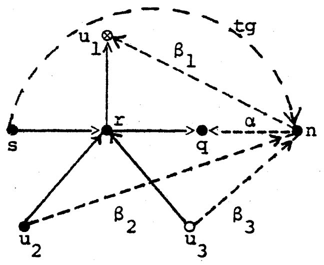


Figure 3.3: Constructed configuration.

Step 1. (Create surrogate)

s creates (tg to) new subject n

Step 2. (Give surrogate β_1 type edge)

s takes (β_1 to u_1) from r

s grants (β_1 to u_1) to n $\forall\ r \xrightarrow{\beta_1} u_1$

Step 3. (Give surrogate β_2 type edges)

Case a. $\beta_2 = g$

u_2 creates (tg to) new subject n'
u_2 grants (g to n') to r
s takes (g to n') from r
s grants (g to n) to n'
u_2 takes (g to n) from n'

Case b. $\beta_2 = t \wedge \alpha = t$

q creates (tg to) new subject n'
u_2 takes (t to q) from r
u_2 takes (t to n') to q
s takes (t to q) from r
s takes (g to n') from q
s grants (t to n) to n'
u_2 takes (t to n) from n'

Case c. $\beta_2 = t \wedge \alpha = g$

u_2 creates (tg to) new subject n'
u_2 takes (g to q) from r
u_2 grants (g to n') to q
s takes (g to q) from r
s grants (t to n) to q
q grants (t to n) to n'
u_2 takes (t to n) from n'

Step 4. (Give surrogate β_3 type edges)

By Theorem 2.2, the only way that objects such as u_3 can participate in "p can α q" transformation is if they have associated word in E (see section 2.2.2). Let v be a subject such that there is a path from v to r with associated word w in E. Then

(i) $w \in \vec{t}(\vec{t})^{+}$

or (ii) $w \in (\vec{t})^{+}\vec{g}(\vec{t})^{*}$

and in the second case the "starred" set is empty. Thus, v can, by a sequence of "takes," get either $v \xrightarrow{t} r$ (i) or $v \xrightarrow{g} r$ (ii) and we apply step 3, where now $u_2 = v$. □

Since the conditions characterizing p can α q can be checked in linear time, an immediate consequence of the theorem is:

Corollary 3.2: There is a linear time algorithm to decide if p can steal α to q.

4. *The Synthesis of the Take-Grant Systems*

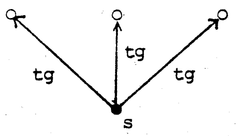
The analysis of the model is now completed and it is time to "synthesize" some systems. As indicated in the introduction, our view of the synthesis activity is that of the *selection* of a particular subset of protection states definable in the model. The subset will include the protection configurations achievable in a particular system. The analysis enables us to make an informed choice as to which configurations to include and which to exclude, i.e., we can determine what sharing can be achieved. This selection process is exhibited by specification of initial configurations (4.1) and by three protection systems designs (4.2-4.4). The designs represent only a sampling of the potential protection structures. They have been selected to illustrate how the model can be used to achieve goals in addition to simple security -- in this case achieving efficiency by balancing the number of users sharing against the total amount of sharing taking place.

It may be helpful before presenting the designs to underscore the distinction between a model and a design. A model provides a means of specifying protection system states, a means of specifying state changes, and a means of stating and proving theorems about these states and state changes. *The model does not provide a design.* A "design" of a protection system -- the structures and operations that the program must effect -- will be embodied in the *assumptions* made in the model. Thus, a design only exists implicitly in the present development. Making it explicit requires writing system specifications that realize and/or enforce the assumptions. Thus, a designer must first formulate a suitable abstract model of protection, such as the Take-Grant Model, and then convert the assumptions of the model into explicit specifications.

4.1 *Initial State and User Processes*

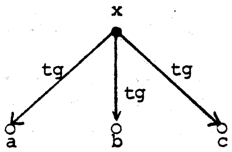
As indicated in the last section, the operating system supervisor is distinct from the protection system and is thus treated just like any other subject in the system. Of course, it does have a special role of joining new users to the system, managing library programs, etc., so considerable interest will be directed toward understanding how it might perform these functions. Accordingly, the initial configuration and the protocol followed by the operating system will be of crucial importance.

In each design the operating supervisor is the initial subject in the system together with its "service objects," i.e., library files, etc. Thus each of the following designs have as initial configurations



where s is the operating system supervisor and the objects are the "service objects." Notice that no edges are incoming to s, so by our theorems none will ever be introduced by remark 2.3a and so no user will be able to steal from the supervisor.

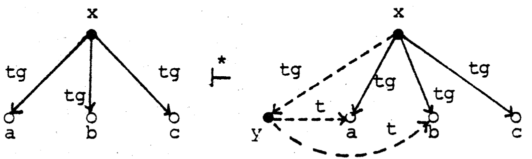
Normally, a user x will be described by the protection subgraph



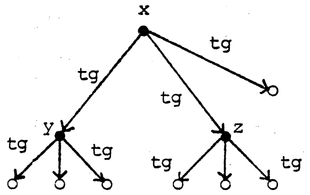
where x is the user and the objects are files. To create a subprocess y to operate on two files, apply the operations

x creates (tg to) new subject y
x grants (t to a) to y
x grants (t to b) to y

which are shown graphically

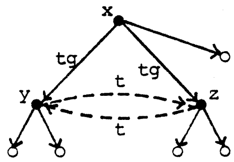


A second general user form achievable in the model are the *project* users, used, for example, by a group jointly writing a compiler. Here x is the project leader (created by the system) while y and z are project workers (created by the project leader) and the graphical representation is



where y and z have created their own files, as does x.

Notice that the project supervisor (x) is allowed to acquire rights to the workers (y,z) without their knowledge. Although x can "steal" objects from y (with our definition of "can steal") this may be permitted as a justifiable access, since the file was created on the supervisor's behalf. If, in addition, the workers' files are to be generally available, then the project leader can grant take rights to the individual workers resulting in



With a take, y can access z's files and vice versa. Other general user structures can obviously be envisioned, e.g., instructor - teaching assistant - students, and the reader is invited to design them.

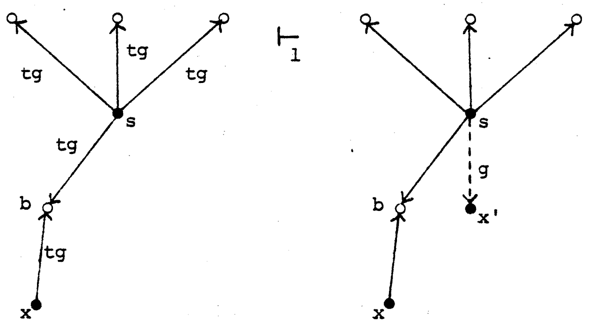
4.2 *Design 1* - *Operating System As Communications Agent*

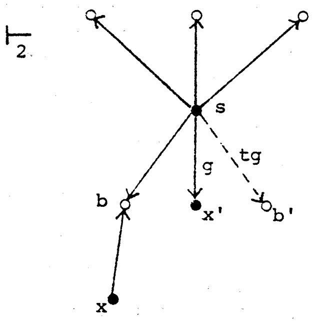
In this design the supervisor communicates with the systems users by means of an object (thought of, possibly, as a buffer). The users communicate with one another by requesting the operating system supervisor to act as intermediary.

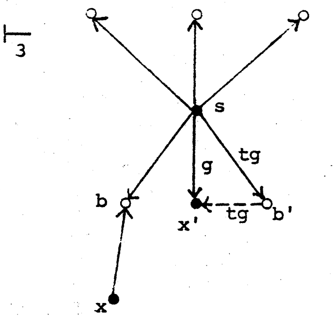
The protocol for introducing a new user x to the system is:

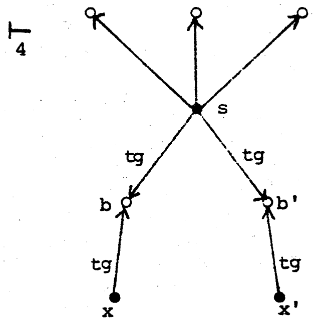
1. s creates (g to) new subject x
2. s creates (tg to) new object b
3. s grants (tg to b) to x
4. s removes (g to) x.

Graphically, a system with one user, x, can have a new user x' added as follows:



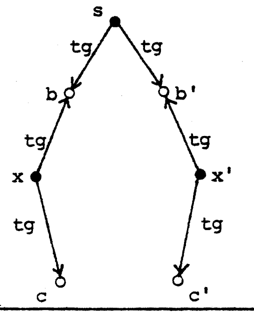






Notice that the user must *trust* the supervisor not to perform step (1) with grant *and* take and then to retain the take right since this would enable the supervisor to take anything created by the user. But if the user requests an audit from the protection system as its first act of business, it can be verified that no such right exists. Also, no arrows are incoming to the user so it can establish a subsystem with the same features as the overall system -- i.e., the user acts as supervisor to its subordinates.

Given the configuration*



x can be given rights to c' using the following protocol.

1. x creates (tg to) new object d
2. x grants (tg to d) to b
3. s takes (tg to d) from b
4. s grants (tg to d) to b'
5. s removes (tg to) d
6. x' takes (tg to d) from b'
7. x' grants (t to c') to d
8. x takes (t to c') from d

Here d acts as a receptical for the data.

In step 5 the operating system yields its right to possibly taking the data and prior to step 7, a paranoid x' could request an audit to verify that s yields its rights *and* that the others have followed the protocol.

Whether or not this design is adequate is dependent on the system's requirements -- a question that cannot be answered here. However, it should be noted that with the supervisor as intermediary there could be a lot of traffic. Thus, in an effort to reduce this, a second design is considered.

4.3 *Design 2* - *No Agent*

Here the operating system supervisor sets up a buffer (such as b in Design 1) between each user pair. Then the sharing responsibilities are placed on the users rather than the supervisor. In addition, the supervisor must retain grant rights over all of the users in order to establish the communication.

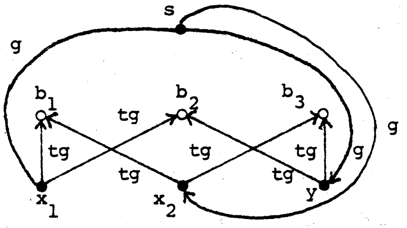
The protocol for introducing new users assuming $x_1,\ldots,x_n$ already exists is:

1. s creates (g to) new subject y
2. s creates (tg to) new object b_1
3. s grants (tg to b_1) to y
4. s grants (tg to b_1) to x_1
5. s removes (tg to) b_1

. . .

k+2. s creates (tg to) new object b_n
k+3. s grants (tg to b_n) to y
k+4. s grants (tg to b_n) to x_n
k+5. s removes (tg to) b_n

The following configuration results when y is added and x_1 and x_2 exist.



Communication among users is a simple task and is left as an exercise.

The design may reduce the variable cost by eliminating communication traffic, but it raises the supervisor's overhead of joining a new user to the system to be proportional to the number of users presently in the system. Moreover, the protection system is swamped with information. If

*The service objects have been elided.

modest sharing among processes is anticipated, Design 3 might be preferred.

4.4 *Design 3 - The Supervisor As Communications Linkage Agent*

The obvious solution to the shortcomings of Designs 1 and 2 is to combine the features -- i.e. the supervisor sets up communication buffers on demand. Thus, the supervisor's work is proportional to the number of users sharing rather than the amount of sharing or the number of users. Also, only those links that are needed are created

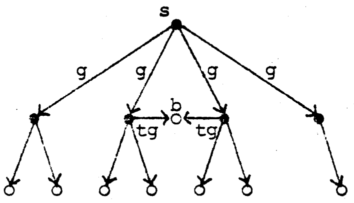
The user creation protocol for user x is simply

s creates (g to) new subject x.

When sharing between subjects x and y is required, the protocol for the supervisor is

1. s creates (tg to) new object b
   -- this is the buffer
2. s grants (tg to b) to x
3. s grants (tg to b) to y
4. s removes (tg to) b.

A sample configuration among four users with two of them sharing might be:



The communication protocol for the users is obvious. Notice also that the users might request an audit once the object b has been created. Moreover, in this scheme (and in the other designs as well) any user can decide to isolate himself from others with whom he has been communicating simply by performing a remove. But by (2.3b), he does so in Design 1 at the risk of perpetual isolation.

4.5 *Summary*

The point to be emphasized is that the formal Take-Grant Model provides a means of guiding the synthesis by permitting informed selection among a multitude of choices. For example, in the foregoing designs the operating system supervisor never allows a user that it creates to have an incoming edge labeled by t since this would allow the *potential* of having rights taken without the user's knowledge. Should a user decide that it desires such rights over its own subsystems (i.e., the ability to steal), then it can create them in this manner. If it is less interventionist than that it could create subsystems after designs 1-3. In any case, the fact that the system had been analyzed and characterized enables everyone to know the potential consequences of their actions. More specifically, the objectives lists in 3.1-3.3 can be solved in these systems.

5. *Discussion and Suggested Future Research*

In summary, the Take-Grant Model has been presented, analyzed and used to produce three protection systems designs. The designs are rich enough to solve typical protection problems, and it is probably a simple matter for a clever designer to improve upon these. Theorem 3.1 states which rights can be given away, and which can be stolen. Consequently, each design derives integrity (at least in the abstract) from this analysis of the model.

The point to be emphasized is that the analysis of a formal model of protection can provide both integrity as well as guidance during synthesis.

In the category of future research, several avenues can be suggested. First, the designs 1-3 of section 4 should be evaluated in the context of practical protection system requirements in order to discover what shortcomings exist. Presuming that some are found, synthesis of additional systems will be indicated. It may be that the analysis is not yet complete and so a second search for any other "uninstantiated parameters" as was done in section 3 can be suggested. Although the Take-Grant Model has probably not yet been exhausted as a source of interesting and challenging problems, a third research direction points to formulating a different basis protection model with a different set of primitive rules. (For example, new edge labels might be introduced to incorporate other capabilities.)

A somewhat different path for future research is to make explicit the assumptions of designs 1-3 (or some other design) and perhaps implement them. This is actually a proposal to do the explicit *design* (in the sense of the implicit *vs* explicit distinction mentioned at the beginning of section 4) by formulating the specifications and program structures that actually enforce the assumptions.

Acknowledgements

It is a pleasure to thank A. K. Jones for the hours generously given to discussing these topics. Thank yous are also due to T. Budd, D. Denning and D. Hanson for their thoughtful comments on an earlier draft, and to G. Spalinger, whose patience is genuinely appreciated, for expertly preparing the manuscript.

References

1. A. K. Jones, R. J. Lipton, and L. Snyder.
   A linear time algorithm for deciding security.
   Proceedings of the 17th FOCS (1976).
2. R. J. Lipton and L. Snyder.
   A linear time algorithm for deciding subject security.
   JACM 24:3 (1977).
3. M. A. Harrison, W. L. Ruzzo, and J. D. Ullman.
   Protection in operating systems.
   CACM 19:8 (1976).