THEORETICAL ISSUES CONCERNING PROTECTION IN OPERATING SYSTEMS

Michael A. Harrison

Computer Science Division
University of California
Berkeley, CA 94720

The research reported in this paper was sponsored by the National Science Foundation under Grant MCS-8311781.

Introduction

Modern computer systems contain important information and unauthorized access can result in significant problems. One need only think of certain examples like electronic funds transfer systems, internal revenue service applications, as well as command and control computers used in military applications to realize the significance and scope of the problem of guaranteeing secure computation.

In the real world, there are many techniques used for penetrating systems which involve subverting people (or machines), tapping communication lines, and breaching physical security. There is no way in which such complex interactions can be modeled in their entirety even with the aid of computers. Instead, one must focus on the most important aspects of the systems and create specialized models in order to understand these parts of the system. Moreover the modeler must decide how specific to be. For example, we could invent a simple model to abstract the function of a particular circuit in the control unit of a computer. A more grandiose scheme might be to model all computer systems at once by studying a uniform model of computation such as a RAM or a Turing machine.

In this chapter, we are interested in protection and security from an operating systems point of view. That forces us to make a number of simplifying assumptions. The remarkable part of our treatment will be that even with severe assumptions, the security question remains "hard" in a technical sense.

In our first model, somewhat akin to the example of modeling a circuit, we shall ignore the computer and operating system and concentrate on the access to an object by a subject. Our first goal is to arrive at a model which is rich enough to describe actual systems but is sufficiently restricted so that one can utilize simple and efficient techniques.

1. A Specialized Model

Introduction

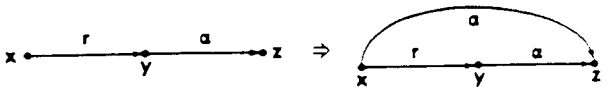
We shall begin our study of models for operating systems by taking a dramatic simplification of a real operating system. The one we have chosen comes from the Ph.D. thesis of Cohen, 1976. Also see Denning and Graham, 1972 and Lipton and Snyder 1977. We have abstracted everything except the relationships between various processes or users in the system. We do model the dynamic changes of access which can occur in the system. This is done by having a sequence of transformation rules on a directed graph which indicates these relationships. We shall explain the various transformations and see that the system has more power than we first realized to access information.

We shall be able to give an algorithm for telling if an arbitrary process p can obtain an arbitrary right αq to an arbitrary object q. The algorithm has the advantage of being very efficient to compute. Then we shall interpret the theorem that we have proven to give a statement of the protection policy which was implemented in this system. The policy will turn out to be more discriminating than one might have expected.

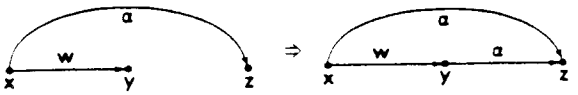
1.1 The Take-Grant Model

The basic idea is to use a type of dynamically changing labeled directed graph as the model. The nodes of the graph represent "users" and the labels on a directed arc are some nonempty subset of $\{r, w, c\}$ where r stands for "read", w for "write" and c for "call". Formally, the model consists of a finite directed graph with no self-loops and each branch labeled as above. Each graph may be transformed by any of the five rewriting rules.

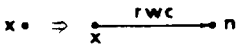
1. **Take:** Let x, y, and z be three distinct vertices in a protection graph, and let there be an arc from x to y with label γ such that $r \in \gamma$ and an arc from y to z with some label $\alpha \subseteq \{r, w, c\}$. The take rule allows one to add the arc from x to z with label α, yielding a new graph G'. Intuitively, x *takes* the ability to do α to z from y. We represent this as shown below: †



2. **Grant:** Let x, y, and z be three distinct vertices in a protection graph G, and let there be an arc from x to y with label γ such that $w \in \gamma$ and an arc from x to z with label $\gamma \subseteq \{r, w, c\}$. The grant rule allows one to add an arc from y to z with label α, yielding a new graph G'. Intuitively, x *grants* y the ability to do α to z. In our representation:

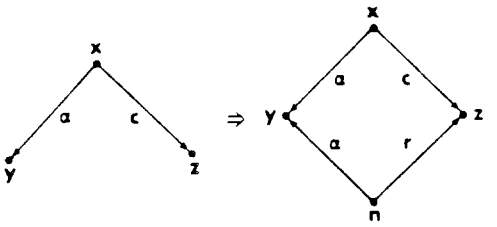


3. **Create:** Let x be any vertex in a protection graph; the create operation allows one to add a new vertex n and an arc from x to n with label $\{r, w, c\}$, yielding a new graph G'. Intuitively, x creates a new user that it can *read, write* and *call*. In our representation:

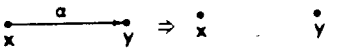


† Here we write an explicit right as an arc label (x $\xrightarrow{r}$ y) to mean the arc label contains that right (i. e. x $\xrightarrow{\gamma}$ y such that $r \in \gamma$.)

4. **Call:** Let x,y and z be distinct vertices in a protection graph G, and let $\alpha \subseteq \{r, w, c\}$ be the label on an arc from x to y and γ the label on an arc from x to z such that $c \in \gamma$. The call rule allows one to add a new vertex n, an arc from n to y with label α, and an arc from n to z with label r, yielding a new graph G'. Intuitively x is *calling* a program z and passing parameters y. The "process" is created to effect the call: n can *read* the program z and can α the parameters. In our representation:



5. **Remove:** Let x and y be distinct vertices in a protection graph G with an arc from x to y with label α. The remove rule allows one to remove the arc from x to y, yielding a new graph G'. Intuitively, x *removes* its rights to y. In our representation

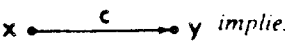


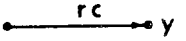
The remove rule is defined mainly for completeness, since real protection systems tend to have such a rule.

The operation of applying one of the rules to a protection graph G yielding a new protection graph G' is written $G \vdash G'$. As usual, $\vdash^*$ denotes the reflexive, transitive closure of $\vdash$.

An important technical point is that these rules are monotone in the sense that if a rule can be applied, then adding arcs cannot change this.

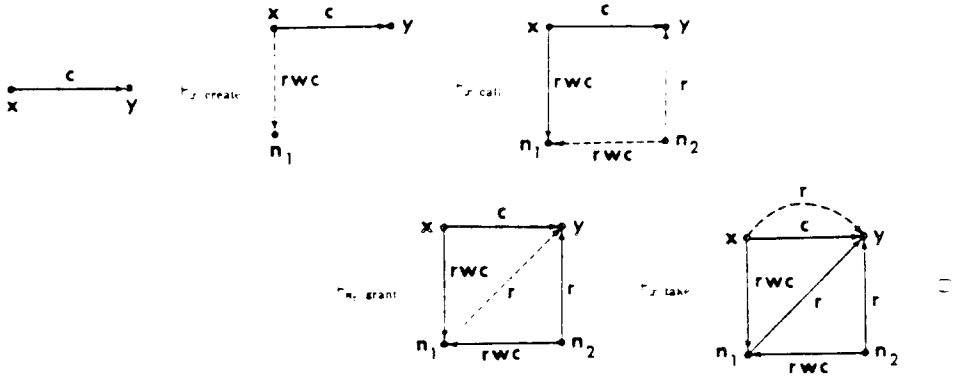
The basic application of this model is to answer questions of the form:" Can $p\alpha q$?" where $\alpha \in \{r, w, c\}$. As an example, we shall show that if x can call y then x can read y. This is just the kind of property we wish to deduce from such a model since the fact that x can read y may have been an unintentional consequence of allowing x to call y.

Example 1. *In a protection graph* G 

implies 

implies

Proof. Apply the following rules:



It can be shown that there are two simple conditions that are necessary and sufficient to determine if vertex p can α vertex q. Let G be a protection graph and $\alpha \in \{r, w, c\}$. Call p and q *connected* if there

exists a path between p and q independent of orientation or labels of the arcs. Define the predicates:

Condition 1: p and q are connected in G.

Condition 2: There exists a vertex x in G and an arc from x to q with label β such that $\alpha = r$ implies $\{r, c\} \cap \beta \neq \emptyset$, or $\alpha = w$ implies $w \in \beta$, or $\alpha = c$ implies $c \in \beta$.

Informally these conditions state that p can αq if and only if there is an undirected path between p and q (condition 1) and some vertex x α's q (condition 2).

It is not hard to see that conditions 1 and 2 are necessary. By a sequence of lemmas, quite similar to the previous example, sufficiency can be established. One can draw the following inferences.

Theorem 1.1.1. *Let p and q be distinct vertices in a protection graph and α a label. Conditions 1 and 2 are necessary and sufficient to imply p can αq.*

The consequence of the main theorem is that the protection policy for this take-grant system can be precisely stated.

Policy: If p can read (write) [call] q, then any user in the connected component containing p and q can attain the right to read (write and call) q.

This policy may appear to be more undiscriminating than one might have expected. A primary reason for this is that the take-grant system treats all elements of the system the same whereas most protection models recognize two different entities: subjects and objects. If we dichotomize the vertices of our model into subject and object sets, by coloring the vertices, and require (as is usually the case) that only subjects can initiate the application of our rules, then the system becomes much more difficult to analyze. Such an analysis has been reported in [JLS]. It should be noted that in the dichtomized model there are protection graphs that satisfy conditions 1 and 2 for which p can αq is false.

Another generalization would involve additional labels which could be passed around. These would represent more complex types of interaction between the subjects. Again the main theorem proven in this section would no longer be true.

Additional work has been done in which "conspiracy" has been treated. We can consider some of the vertices to be conspirators. Then one can ask questions like, if p can αq, can it do so with a bounded number of conspirators in the sub-graph? The reader is referred to [BiS] [Sny].

This simple system gives us an idea of what to expect of a simple model. We learned something and were able to sharpen our notion of the policy being implemented. On the other hand, stating a more general result about the security of an overall system would be unjustified by this model alone.

2. A Uniform Approach to Modeling Protection

Introduction

We are now about to take a very high level and ambitious approach to modeling security. In this extreme approach, the goal is to find a uniform model which is sufficiently general to be able to model all types of protection mechanisms. Continuing the analogy of the introduction, this model is like a Turing machine because of its uniformity. This model has become known as the HRU model in the literature because the model was originally introduced by Harrison, Ruzzo, and Ullman in 1976.

2.1 The HRU Model

Our first task is to arrive at a model which is general but captures the essence of "safety" from unauthorized access. Towards this end, let us assume that a computer contains a collection of abstract objects whose security is important. In practice, these objects would be interpreted as files containing important data. Let us furthermore postulate the existence of a "reference monitor" which is to be interposed between the world and the protected objects as in Figure. 1 which is intended to represent a modern computer system with multiple users or even multiple processors. By assuming that all accesses to the protected objects go through the reference monitor, and further that all the hardware is infallible, one can model these systems by examining the dynamic behavior of the monitor. One should note that the effect of the users and of the operating system itself, are assumed to affect access to the objects only through the "commands" which enter the monitor. The perfect hardware implements each access.

It is now time to get more precise concerning the model.

Definition. A *protection system* consists of the following parts: a finite set of *generic rights* R, a finite set of *commands* of the form:

command $\alpha(X_1, X_2, \ldots, X_k)$
  if $r_1 \in (X_{s_1}, X_{o_1})$ and
    $r_2 \in (X_{s_2}, X_{o_2})$ and
      $\vdots$
    $r_m \in (X_{s_m}, X_{o_m})$
  then
    op_1
    op_2
    $\vdots$
    op_n
end

or if m is zero, simply

command $\alpha(X_1, X_2, \ldots, X_k)$
    op_1
    op_2
    $\ldots$
    op_n
end

In our definition α is a name and $X_1, \ldots, X_k$ are formal parameters. Each op_i is one of the six following primitive operations.

enter r **into** (X_s, X_o)	**delete** r **from** (X_s, X_o)
create subject X_s	**destroy subject** X_s
create object X_o	**destroy object** X_o

By convention $r, r_1, r_2, \ldots, r_k$ denote generic rights. We use $s, s_1, s_2, \ldots, s_m$ and $o, o_1, o_2, \ldots, o_m$ to denote states and objects respectively. We also need to discuss the "configurations" of a protection system. Intuitively, these correspond to the instantaneous configurations used in the usual definition of automata. See Harrison 1978.

Definition. A *configuration* of a protection system is a triple (S, O, P), where S is the set of "current subjects", O is the set of "current objects", $S \subseteq O$, and P is an *access matrix*, which has a row for each subject in S and a column for each object in O. $P[s, o]$ is a subset of R, the set of generic rights and gives the rights that s enjoys with respect to o.

Before proceeding further, let us consider a simple example which exposes the most common interpretation of the model.

Example 1. We assume that each subject is a process and that the objects other than the subjects are files. Each file is owned by a process, and we shall model this notion by saying that the owner of the file

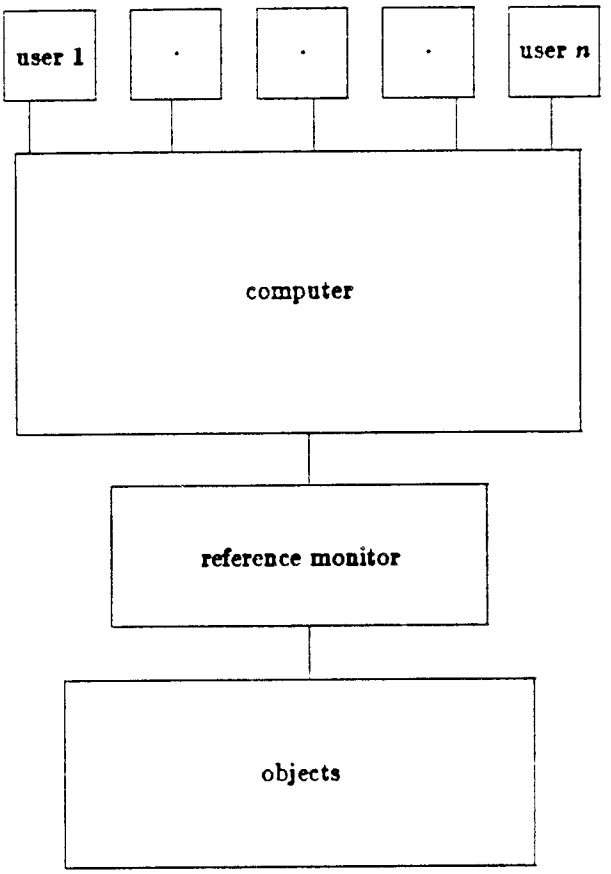


Figure 1
A computer system with a reference monitor.

has the right **own** to that file. The other generic rights are **read, write,** and **execute.** The allowable operations are as follows.

(1) A process may create a new file. The process which creates the file has ownership of it. This may be represented by a procedure:

command $CREATE$ (process,file)
  create object file
  enter own into (process, file)
end

(2) The owner of a file may confer any right to that file, other than **own**, on any subject including the owner himself). We thus have three commands of the form:

command $CONFER_r$(owner, friend, file)
  if own in (owner,file)
    enter *r* **into** (friend,file)
end

where *r* is **read, write,** or **execute.** Technically the *r* here is not a parameter but is used as an abbreviation for three similar procedures, one for each value of *r*.

(3) Similarly, we have three commands by which the ownership of a file may revoke another subject's access rights to the file.

command $REMOVE_r$(owner, exfriend, file)
  if own in (owner,file) and
    r in (exfriend,file)
  then delete *r* **from** (exfriend,file)

end

where r is **read**, **write**, or **execute**.

This completes the specification of most of the example protection system.

To formally describe the effect of the commands, we must give the rules for changing the state of the access matrix.

A typical access matrix is shown in Figure 2.

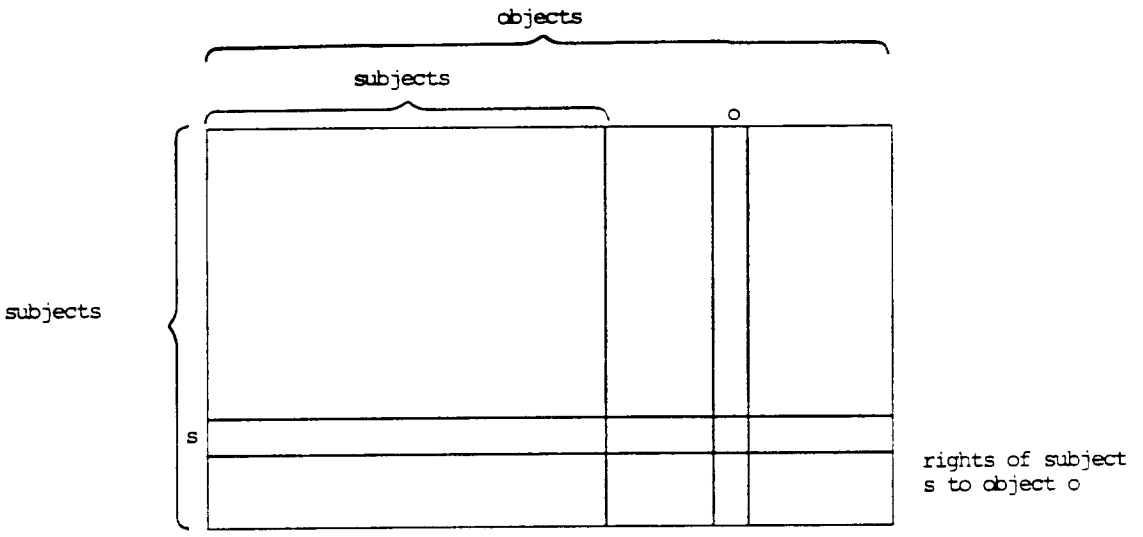


Figure 2
An Access Matrix

Note that the s-th row may be thought of as a "capability list" while the o-th column is an "access-control list".

Next, we need the rules for changing configurations in a protection system.

Definition. Let (S, O, P) and (S', O', P') be configurations of a protection system, and let op be one of the six primitive operations. We shall say that:

$$(S, O, P) \underset{op}{\Rightarrow} (S', O', P')$$

[which is read (S, O, P) yields (S', O', P') under op] if either:

1. op = **enter** r **into** (s, o) and $S = S'$, $O = O'$, $s \in S$, $o \in O$, $P'[s_1, o_1] = P[s_1, o_1]$ if $(s_1, o_1) \neq (s, o)$ and $P'[s, o] = P[s, o] \cup \{r\}$, or
2. op = **delete** r **from** (s, o) and $S = S', O = O'$, $s \in S$, $o \in O$, $P'[s_1, o_1] = P[s_1, o_1]$ if $(s_1, o_1) \neq (s, o)$ and $P'[s, o] = P[s, o] - \{r\}$,
3. op = **create subject** s',where s' is a new symbol not in O , $S' = S \cup \{s'\}$, $O' = O \cup \{s'\}$, $P'[s, o] = P[s, o]$ for all $(s, o) \in S \times O$, $P'[s, o] = \emptyset$ for all $o \in O'$, and $P'[s, s'] = \emptyset$ for all $s \in S'$, or
4. op = **create object** o',where o' is a new symbol not in O , $S' = S$, $O' = O \cup \{o'\}$, $P'[s, o] = P[s, o]$ for all $(s, o) \in S \times O$, and $P'[s, o'] = \emptyset$ for all $s \in S$, or
5. op = **destroy subject** s', where $s' \in S$, $S' = S - \{s'\}$, $O' = O - \{s'\}$, and $P'[s, o] = P[s, o]$ for all $(s, o) \in S' \times O'$, or
6. op = **destroy object** o', where $o' \in O - S$, $S' = S$, $O' = O - \{o'\}$, and $P'[s, o] = P[s, o]$ for all $(s, o) \in S' \times O'$.

Next we indicate how a protection system can execute a command.

Definition. Let $Q = (S, O, P)$ be a configuration of a protection system containing:

command $\alpha(X_1, X_2, \ldots, X_k)$

if $r_1 \in (X_{s_1}, X_{o_1})$ and
...
$r_m \in (X_{s_m}, X_{o_m})$
then
op_1
op_2
...
op_n
end

Then we can say that

$$Q \overset{*}{\underset{\alpha(x_1,\ldots,x_k)}{\vdash}} Q'$$

where Q' is the configuration defined as follows:

1. If α's conditions are not satisfied, i.e. if there is some $1 \leq i \leq m$ such that r_i is not in $P[x_{s_i}, x_{s_i}]$, then $Q = Q'$.
2. Otherwise, i.e. if for all i between 1 and m, $r_i \in P[x_{s_i}, x_{s_i}]$, then let there exist configurations $Q_0, Q_1, \ldots, Q_n$ such that

$$Q = Q_0 \underset{op_1^*}{\Rightarrow} Q_1 \underset{op_2^*}{\Rightarrow} \cdots \underset{op_n^*}{\Rightarrow} Q_n = Q'$$

where op_i^* denotes the primitive operation op_i with the actual parameters $x_1, \ldots, x_k$ replacing all occurrences of the formal parameters $X_1, \ldots, X_k$, respectively. Then Q' is Q_n.

Similarly, $Q \vdash_\alpha Q'$ if there are parameters $x_1, \ldots, x_k$ such that $Q \vdash_{\alpha(x_1,\ldots,x_k)} Q'$. Also $Q \vdash Q'$ if there exists a command α such that $Q \vdash_\alpha Q'$.

It is also convenient to write $Q \vdash^* Q'$, where $\vdash^*$ is the reflexive and transitive closure of $\vdash$. That is, $\vdash^*$ represents zero or more applications of $\vdash$.

Each command is given in terms of formal parameters. At execution time, the formal parameters are replaced by actual parameters which are object names. Although the same symbols are often used in this exposition for formal and actual parameters, this should not cause confusion. The "type checking" involved in determining that a command may be executed takes place with respect to actual parameters.

These protection systems compute like nondeterministic devices. This makes intuitive sense as the sequence of accesses of the protected objects may come in an unpredictable fashion.

Example: These techniques can be used to model the protection aspects of the UNIX * operating system.

In this simple system, each file has an owner who may specify whether he or she can **read**, **write** or **execute** a file. Also the privileges of other members of the same group, or other users can be specified. Thus, we will need an **own** right. Normally, the rights of s to f would go into the (s, f) entry. But when f is created, how could we give all subjects the right to read f since there are a potentially unbounded number of subjects? One trick is to put the rights in the (f, f) entry and to treat a file as a subject. Thus s can read f if either

1. **own** $\in (s, f)$ and **owner_can_read** $\in (f, f)$ or
2. **anyone_can_read** $\in (f, f)$.

This solution creates a new problem however. How do we encode this disjunctive condition into the limited formalism of this model? The trick is to use commands which have identical bodies but with different conditions. The detailed procedures follow:

command *CREATE_FILE* (u, f)
create subject f
enter own into (u, f)
end

* UNIX is a trademark of Bell Laboratories.

```
command LET_OWNER_READ (u, f)
  if own ∈ (u, f)
  then enter owner_can_read into (f, f)
end
command LET_ANYONE_READ (u, f)
  if own ∈ (u, f)
  then enter anyone_can_read into (f, f)
end
command READ (u, f)
  if either
    own ∈ (u, f) and
    owner_can_read ∈ (f, f)
  or
    anyone_can_read ∈ (f, f)
  then
    enter read into (u, f)
    delete read from (u, f)
end
```

The procedures that we have displayed are just for **read**. Similar procedures must be written for **write** and **execute**. Note that the use of **read** here is purely symbolic. It will never exist in the matrix between commands.

The ability of the model to describe the policies used in real systems has been demonstrated by Harrison, Ruzzo, and Ullman 1976.

2.2. Safety

It is important to be able to discuss safe (and hence unsafe) systems precisely. We shall approach the notion of "safety" by attempting to characterize "unsafety". That, in turn, requires a definition of what it means for a command to "leak" a right.

Definition. Given a protection system, we say command $\alpha(X_1, \ldots, X_k)$ *leaks generic right* r *from configuration* $Q = (S, O, P)$ if α, when run on Q, can execute a primitive operation which enters r into a cell of the access matrix which did not previously contain r. More formally, there is some assignment of actual parameters $x_1, \ldots, x_k$ such that

1. $\alpha(x_1, \ldots, x_k)$ has its conditions satisfied in Q, i. e. for each clause "r in (X_i, X_j)" in α's conditions we have $r \in P[x_i, x_j]$, and
2. if α's body is $op_1, \ldots, op_n$, then there is an m, $1 \leq m \leq n$, and configurations $Q = Q_0, Q_1, \ldots, Q_{m-1} = (S', O', P')$, and $Q_m = (S'', O'', P'')$, such that

$$Q_0 \underset{op_1^*}{\Rightarrow} Q_1 \underset{op_2^*}{\Rightarrow} \cdots \underset{op_m^*}{\Rightarrow} Q_m$$

where op_i^* denotes op_i after the substitution of $x_1, \ldots, x_k$ for $X_1, \ldots, X_k$ and there exists some s and o such that $r \notin P'[s, o]$ but $r \in P''[s, o]$. (Of course, op_m must be **enter** r **into** (s, o)).

Notice that given Q,α and r, it is easy to check whether α leaks r from$Q$ even if α deletes r after entering it. This latter condition may seem unnatural but even having the r in the matrix for one unit of time is enough to cause a leak. We could arrange for a system to "block" in the middle of a command and to interrupt a procedure.

It is important to note that leaks are not necessarily bad. The judgement about whether or not a leak is unfortunate depends on whether or not the subjects are trusted.

Definition. Given a particular protection system and generic right r, we say that the initial configuration Q_0 is *unsafe* for r (or *leaks* r) if there is a configuration Q and a command α such that

1. $Q_0 \vdash^* Q$, and
2. α leaks r from Q.

We say that Q_0 is *safe* for r if Q_0 is not unsafe for r.

Now, let us pause a moment to examine what we have accomplished with the HRU model. We started with a concern about real security issues. A model was introduced which attempted to capture the ways in which objects might be accessed by processes. The model was progressively simplified until we now have a somewhat limited object of study. We have found a technically reasonable "safety question" which we would like to solve. A solution could mean several different things. It would be nice to have a uniform procedure for solving any safety question for any protection system. If this is not possible, we could consider settling for a less general result.

If we were to derive an efficient algorithm for solving the safety question, it could be challenged on the grounds of the simplicity of the model, e.g. "Has the problem been defined away?". In fact, the results are somewhat surprising at first glance. It will be shown that there is no algorithm which can solve the safety question. Even in our restricted model, the problem is unsolvable which means that in any more realistic version, the same argument will carry over and the result will hold. In the next section, we shall summarize what is known about safety questions.

2.3 Some Mathematical Results

Now that we have a model, we will mention some of the results that are known about the safety question. First, the "good news".

Definition. A protection system is *mono-operational* if each command's interpretation is a single primitive operation.

It is possible to find an algorithm to test for safety in such systems.

Theorem 2.3.1. *There is an algorithm which decides whether a given mono-operational protection system and given initial configuration is unsafe for a given generic right r.*

Proof. The proof hinges on two simple observations. First, commands can test for the presence of right, but not for the absence of rights or objects. This allows delete and destroy commands † to be removed from computations leading to a leak. Second, a command can only identify objects by the rights in their row and column of the access matrix. No mono-operational command can both create an object and enter rights, so multiple creates can be removed from computations, leaving the creation of only one subject. This allows the length of the shortest "leaky" computation to be bounded.

Suppose

$$Q_0 \underset{C_1}{\vdash} Q_1 \underset{C_2}{\vdash} \cdots \underset{C_m}{\vdash} Q_m \tag{2.3.1}$$

is a minimal length computation reaching some configuration Q_m for which there is a command α leaking r. Let $Q_i = (S_i, O_i, P_i)$.

Claim. $C_i, 2 \leq i \leq m$, *is an* **enter** *command. Moreover,* C_1 *is either an* enter *or* **create subject** *command.*

Proof of the claim. Suppose not, and let C_n be the last non-enter command in the sequence (2.3.1). Then we could form a shorter computation

$$Q_0 \underset{C_1}{\vdash} Q_1 \underset{C_2}{\vdash} \cdots Q_{n-1} \underset{C'_{n+1}}{\vdash} Q'_{n+1} \underset{C'_{n+2}}{\vdash} \cdots \underset{C'_m}{\vdash} Q'_m$$

as follows:

a. if C_n is a delete or destroy command, let $C'_i = C_i$ and $Q'_i = Q_i$ plus the right, subject or object which would have been deleted or destroyed by C_n. By the first observation above, C_i cannot distinguish Q_{i-1} from Q'_{i-1}, so $Q'_{i-1} \vdash_{C'_i} Q'_i$ holds. Likewise, α leaks r from Q'_m since it did so from Q_m.

b. Suppose that 1) C_n is a create subject command and $| S_{n-1} | \geq 1$ or 2) that C_n is a create object command. Note that α leaks r from Q_m by assumption, so α is an enter command. Further, we must have $| S_m | \geq 1$ and

$$| S_m | = | S_{m-1} | = \cdots = | S_n | \geq 1$$

because $C_m, \ldots, C_{n+1}$ are enter commands by assumption and hence do not change the number of subjects. Thus $| S_{n-1} | \geq 1$ even if C_n is a create object command. Let $s \in S_{n-1}$. Let o be the name of the object created by C_n. Now we can let $C'_i = C_i$ with s replacing all occurrences of o, and $Q'_i = Q_i$ with s and o merged. For example, if $o \in O_n - S_n$ we would have

$$S'_i = S_i \quad \text{and} \quad O'_i = O_i - \{o\}$$

and

$$P'_i[x, y] = \begin{cases} P_i[x, y] & \text{if } y \neq s; \\ P_i[x, s] \cup P_i[x, o] & \text{if } y = s. \end{cases}$$

Clearly,

$$P_i[x, o] \subseteq P'_i[x, s],$$

so for any condition in C_i satisfied by o, the corresponding condition in C'_i is satisfied by s. Likewise for the conditions of α.

† Since the system is mono-operational, we can identify the command by the type of primitive operation.

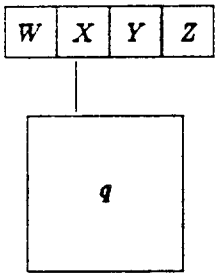


Figure 3
A Turing machine in state q reading a_i.

c. Otherwise, we have $| S_{n-1} | = 0$, C_n is a create subject command. If $n \leq 2$ then there is nothing to prove so we can assume $n \geq 2$. The construction in this case is slightly different - the create subject command cannot be deleted (subsequent "enters" would have no place to enter into). However, the commands preceding C_n can be skipped (provided that the names of objects created by them are replaced), giving

$$Q_0 \underset{C_n}{\vdash} Q_n \underset{C'_{n+1}}{\vdash} Q'_{n+1} \underset{C'_{n+2}}{\vdash} \cdots \underset{C'_m}{\vdash} Q'_m$$

where, if $S_n = \{s\}$, we have that C'_i is C_i with s replacing the names of all objects in O_{n-1}, and Q'_i is Q_i with s merged with all $o \in O_{n-1}$.

In each of these cases we have created a shorter "leaky" computation, contradicting the supposed minimality of (2.3.1). Now we note that no C_i enters a right r into a cell of the access matrix already containing r, else we could get a shorter sequence by deleting C_i. Thus we have the following upper bound on m:

$$m \leq g(| S_0 | +1)(| O_o | +1) + 1 \qquad (2.3.2)$$

where g is the number of generic rights. This follows because the size of the original matrix can increase by at most one. An obvious decision procedure now presents itself. Try all sequences of enter commands, of length up to the bound given in equation (2.3.2). The sequences to be tested can be constrained by the claims established in order to save work. ∎

The algorithm given in the previous proof is exponential in the matrix size. However, by using dynamic programming, an algorithm can be derived for any given protection system which is polynomial in the size of the initial matrix. It is worth noting that if we wish a decision procedure for all mono-operational systems, where the commands are a parameter of the problem, then the decision problem is **NP** -complete. Thus the above problem is almost certainly of exponential time complexity in the size of the matrix. An argument will be sketched by reducing the k-clique problem to the problem: given a mono-operational system, a right r and an initial access matrix, determine if that matrix is safe for r. Given a graph and an integer k, we can produce a protection system whose initial access matrix is the adjacency matrix for the graph and which has one command. This command's conditions test its k parameters to see if they form a k-clique, and its body enters some right r somewhere. The matrix will be unsafe for r in this system if and only if the graph has a k-clique. The argument given above is a polynomial reduction of the clique problem, which is **NP** -complete, to our problem. Thus our problem is at best **NP** -complete. It is easy to find a nondeterministic polynomial time algorithm to test safety, so our problem is in fact **NP** -complete and no worse.

One obvious corollary of the above is that any family of protection systems which includes the mono-operational systems must have a general decision problem which is at least as difficult as the **NP** -complete problems, although individual members of the family could have easier decision problems.

Now let us consider general protection systems. Is it possible to solve the safety problem in a uniform manner for all protection systems? We are about to prove a negative result which becomes all the more significant because of the weakness of the model.

Theorem 2.3.2. *It is undecidable whether a given configuration of a given protection system is safe for a given generic right.*

	s_1	s_2	s_3	s_4
s_1	$\{W\}$	{own}		
s_2		$\{X, q\}$	{own}	
s_3			$\{Y\}$	{own}
s_4				$\{Z$, end$\}$

Figure 4
Matrix which simulates machine in Figure. 3.

	s_1	s_2	s_3	s_4
s_1	$\{W, p\}$	{own}		
s_2		$\{Y\}$	{own}	
s_3			$\{Y\}$	{own}
s_4				$\{Z$, end$\}$

Figure 5. Representing a tape.

Proof. There are a number of ways to prove this result. The access control matrix can be used to encode a string of potentially unbounded length on the main diagonal. If we have a Turing machine as shown in Figure 3, we encode it into the matrix as shown in Figure 4.

We will now formalize this idea.

We are now going to prove that the general safety problem is not decidable. We assume the reader is familiar with the notion of a Turing machine. Cf. Harrison 1978. Each Turing machine T consists of a finite set of *states* K and a distinct finite set of *tape symbols* Γ. One of the tape symbols is the *blank* B, which initially appears on each cell of a tape which is infinite to the right only (that is, the tape cells are numbered $1, 2, \ldots, i, \ldots$). There is a *tape head* which is always *scanning* (located at) some cell of the tape.

The *moves* of T are specified by a function δ from $K \times \Gamma$ to $K \times \Gamma \times \{L, R\}$. If $\delta(q, X) = (p, Y, R)$ for states p and q and tape symbols X and Y, then should the Turing machine T find itself in state q, with its tape head scanning a cell holding symbol X, then T enters state p, erases X and prints Y on the tape cell scanned and moves its tape head one cell to the right. If $\delta(q, X) = (p, Y, L)$, the same things happens, but the tape head moves one cell left (but never off the left end of the tape at cell 1).

Initially, T is in state q_0, the *initial state* , with its head at cell 1. Each tape cell holds the blank. There is a particular state q_f, known as the *final state* , and it is a fact that it is undecidable whether started as above, an arbitrary Turing machine T will eventually enter state q_f.

Theorem 2.3.3. *It is undecidable whether a given configuration of a given protection system is safe for a given generic right.*

Proof. We shall show that safety is undecidable by showing that a protection system, as we have defined the term, can simulate the behavior of an arbitrary Turing machine, with leakage of a right corresponding to the Turing machine entering a final state, a condition we know to be undecidable. The set of generic rights of our protection system will include the states and tape symbols of the Turing machine. At any time, the Turing machine will have some finite initial prefix of its tape cells, say $1, 2, \ldots, k$, which it has ever scanned. This situation will be represented by a sequence of k subjects, $s_1, s_2, \ldots, s_k$, such that s_i "owns" s_{i+1} for $1 \leq i < k$. Thus we use the ownership relation to order subjects into a linear list representing the tape of the Turing machine. Subject s_i represents cell i, and the fact that cell i now holds tape symbol X is represented by giving s_i generic right X to itself. The fact that q is the current state and that the tape head is scanning the jth cell is represented by giving s_j generic right q to itself. Note that we have assumed the states distinct from the tape symbols, so no confusion can result.

There is a special generic right **end** , which marks the last subject s_k. That is, s_k has generic right **end** to itself, indicating that we have not yet created the subject s_{k+1} which s_k is to own. The generic right **own** completes the set of generic rights. An example showing how a tape whose first four cells hold $WXYZ$, with the tape head at the second cell and the machine in state q, is shown in Figure 5.

The moves of the Turing machine are reflected in commands as follows. First, if

$$\delta(q, X) = (p, Y, L)$$

then there is

command $C_{qX}(s, s')$
  if
  own in (s, s') and
  q in (s', s') and
  X in (s', s')
then
  delete q **from** (s', s')
  delete X **from** (s', s')
  enter p **into** (s, s)
  enter Y **into** (s', s')
end

That is, s and s' must represent two consecutive cells of the tape, with the machine in state q, scanning the cell represented by s'. The body of the command changes X to Y and moves the head left, changing state from q to p. For example, Figure 4 becomes Figure 5 when command C_{qX} is applied.

If

$$\delta(q, X) = (p, Y, R)$$

that is, the tape head moves right, then we have two commands, depending whether or not the head passes the current end of the tape, that is, the **end** right. There is

command $C_{qX}(s, s')$
  if own $\in (s, s')$ and
    q in (s, s) and
    X in (s, s)
  then
    delete q **from** (s, s)
    delete X **from** (s, s)
    enter p **into** (s', s')
    enter Y **into** (s, s)
end

To handle the case where the Turing machine moves into new territory, there is also

command $D_{qX}(s, s')$
  if end $\in (s, s)$ and
    q in (s, s) and
    X in (s, s)
  then
    delete q **from** (s, s)
    delete X **from** (s, s)
    create subject s'
    enter B **into** (s', s')
    enter p **into** (s', s')
    enter Y **into** (s, s)
    delete end from (s, s)
    enter end into (s', s')
    enter own into (s, s')
end

If we begin with the initial matrix having one subject s_1, with rights q_0, B (blank) and end to itself, then the access matrix will always have exactly one generic right that is a state. This follows because each

command deletes a state known by the conditions of that command to exist. Each command also enters one state into the matrix. Also, no entry in the access matrix can have more than one generic right that is a tape symbol by a similar argument. Likewise, **end** appears in only one entry of the matrix, the diagonal entry for the last created subject.

Thus, in each configuration of the protection system reachable from the initial configuration, there is at most one command applicable. This follows from the fact that the Turing machine has at most one applicable move in any situation, and the fact that C_{qX} and D_{qX} can never be simultaneously applicable. The protection system must therefore exactly simulate the Turing machine using the representation we have described. If the Turing machine enters state q_f, then the protection system can leak generic right q_f, otherwise, it is safe for q_f. Since it is undecidable whether the Turing machine enters q_f, it must be undecidable whether the protection system is safe for q_f. ∎

While this result is discouraging from the point of view of guaranteeing safety, perhaps there are less general results to be obtained which are still valuable. Although Theorem 2.3.2. says that there is no single algorithm which can decide safety for all protection systems, one might hope that for each protection system, one could find a particular algorithm to decide safety. It can easily be seen that this is not possible. The simulation technique which was used previously can be applied to a universal Turing machine on an arbitrary input. This leads to a particular protection system for which it is undecidable whether a given initial configuration is safe for a given right. While we could give different algorithms to decide safety for different classes of systems there is no hope of covering all systems with a finite or even an infinite class of algorithms.

It might be the case that the power of these systems is caused by only one or two of the operations. It is natural to investigate the power of the fundamental operations. The first idea would be to limit the growth of such systems. While such a limitation of resources does make safety decidable, we can show the following.

Theorem 2.3.3. *The question of safety for protection systems without create commands is complete in polynomial space.*

Proof. A construction similar to Theorem 2.3.2 proves that any polynomial space bounded Turing machine can be reduced in polynomial time to an initial access matrix whose size is polynomial in the length of the Turing machine input.

The proof techniques which were employed in Harrison, Ruzzo, and Ullman, 1976 and also in Harrison and Ruzzo, 1978 all made use of the diagonal of the access matrix in an essential way. What would happen of there were only a finite number of subjects and the number of objects which are not subjects was still unconstrained? Would the safety problem become "tractable"? Lipton and Snyder, 1977 have provided the following answer.

Theorem 2.3.4. *The safety problem for protection system with a finite number of subjects is decidable.*

Moreover, it is shown that such protection systems are recursively equivalent to" vector addition systems" and a connection between the safety question for the former and the covering problem for the latter is obtained. Although the safety question is decidable, it is again not something one would care to compute.

† *This suggests that it probably takes exponential time.*

2.4 Monotonicity

In an attempt to better understand wherein lies the computational power of protection systems, we shall now consider systems which can only increase in both size and in the entries in the matrix.

Definition. A protection system is *monotonic* if no command contains a primitive operation of the form

destroy subject s
destroy object o
delete r **from** (s, o)

A number of our colleagues who are familiar with operating systems constructs conjectured that monotonicity would reduce the computing power of protection systems. We shall show that it does not do so. It merely requires a different kind of proof which is more intricate and hence more interesting.

Theorem 2.4.1. *It is undecidable whether a given configuration of a given monotonic protection system is safe for a given generic right.*

Proof. The idea of the proof is to encode an instance of the Post correspondence problem. See Post 1946. on the main diagonal of the access matrix. We would like to be able to grow an x-list and a y-list and at a suitable point in time, to compare them. Because of the monotonic restriction, the x and y lists must be "interlaced" and the check for equality is done by "pointer chasing".

Proof. The idea of the proof would be to encode an instance of the Post Correspondence Problem, Post 1946. on the main diagonal of the access matrix. We would like to be able to grow an x-list and a y-list and at a suitable point in time, to compare them. Because of the monotonic restriction, the x and y lists must be "interlaced" and the check for equality is done by "pointer-chasing."

Formally, suppose we have an instance of the Post Correspondence Problem given by

$$\mathbf{x} = (x_1, \ldots, x_n) \quad \text{and} \quad \mathbf{y} = (y_1, \ldots, y_n)$$

where $x_i, y_i \in \{0,1\}^+$. It is convenient to define

$$x_i = x_{i1} \cdots x_{il_i} \quad \text{and} \quad y_i = y_{i1} \cdots y_{im_i}$$

where $x_{ij}, y_{ik} \in \{0,1\}$ for all i, j, k such that $1 \leq i \leq n$, $1 \leq j \leq l_i$, and $1 \leq k \leq m_i$.

We shall construct a protection system which has the following set of generic rights

$$R = \{0, 1, \textbf{link}, \textbf{start}, \textbf{match}, \textbf{yx-end}, \text{leak}\}$$

and the following commands: For each $i, 1 \leq i \leq n$, we have a procedure:

command $START_i(X_1, \ldots, X_{l_i}, Y_1, \ldots, Y_{m_i})$
  for $j := 1$ **to** l_i **do create subject** X_j †
  for $j := 1$ **to** l_i **do enter** x_{ij} **into** (X_j, X_j)
  for $j := 1$ **to** l_{i-1} **do enter link into** (X_j, X_{j+1})
  for $j := 1$ **to** m_i **do create subject** Y_j
  for $j := 1$ **to** m_i **do enter** y_{ij} **into** (Y_j, Y_j)
  for $j := 1$ **to** m_{i-1} **do enter link into** (Y_j, Y_{j+1})
  enter yx-end into (Y_{m_i}, X_{l_i})
  enter match into (Y_{m_i}, X_{l_i}) **if** $y_{m_i} = x_{l_i}$ ‡
  enter start into (Y_1, X_1)
end

For each $i, 1 \leq i \leq n$, we have a procedure

command $GROW_i(YEND, XEND. X_1, \ldots, X_{l_i}, Y_1, \ldots, Y_{m_i})$

† This notation is a shorthand for **create subject** $X_1 \ldots$ **create subject** X_{l_i}.

‡ The notation means that the primitive operation is included in the command if $y_{m_i} = x_{l_i}$.

if yx-end $\in (YEND, XEND)$
then
for $j := 1$ **to** l_i **do create subject** X_j
for $j := 1$ **to** l_i **do enter** x_{ij} **into** (X_j, X_j)
for $j := 1$ **to** l_{i-1} **do enter link into** (X_j, X_{j+1})
for $j := 1$ **to** m_i **do create subject** Y_j
for $j := 1$ **to** m_i **do enter** y_{ij} **into** (Y_j, Y_j)
for $j := 1$ **to** m_{i-1} **do enter link into** (Y_j, Y_{j+1})
enter yx-end into (Y_{m_i}, X_{l_i})
enter match into (Y_{m_i}, X_{l_i}) **if** $y_{m_i} = x_{l_i}$
enter link into $(XEND, X_1)$
enter link into $(YEND, Y_1)$
end

For each $b \in \{0, 1\}$, we have a procedure

command $MATCH_b(Y, X, AY, AX)$
if match $\in (Y, X)$ and
link $\in (AY, Y)$ and
link $\in (AX, AX)$ and
b $\in (AY, AY)$
then
enter match into (AY, AX)
end

Lastly,

command $LEAK(Y, X)$
if start $\in (Y, X)$ and
match $\in (Y, X)$
then
enter leak into (Y, X)
end

Intuitively, this protection system "computes", starting with an empty configuration, as follows: Each command $START_i$ encodes strings o x_i and y_i into the protection matrix. The location of the first pair of symbols, (x_{i1}, y_{i1}), is marked by **start** while the last pair, (x_{il_i}, y_{im_i}), is marked by **yx-end** .

Each command $GROW_i$ adds x_i and y_i to the end of some sequence of x's and y's which have been previously entered into the matrix. The locations of the ends of such a sequence are indicated by the **yx-end** right. Similarly, GROW$_i$ marks the end of the new sequence with **yx-end** .

Notice that GROW$_i$ is conditional only upon some **yx-end** , which is never deleted. Thus, several different GROW$_i$ commands may be applied to the same **yx-end** . Each GROW$_i$ may then be thought of as growing a new branch on each of two trees - one in which paths from the root represent sequences of x's, the other representing corresponding sequences of y's. The **start** right associates the roots of the two trees while the **link** rights associate ancestors and descendents, and finally the **yx-end** rights indicate ends of corresponding paths. Moreover, the $START_i$ commands are unconditional so that we may actually get a forest of these pairs of trees.

Before starting the formal proof, an intuitive example will be worked. Suppose

$$\mathbf{x} = (01, 1) \quad \text{and} \quad \mathbf{y} = (0, 11)$$

	X_1	X_2	Y_1	X_3	X_4	Y_2	X_5	Y_3	Y_4
X_1	0	link							
X_2		1		link			link		
Y_1	start match leak	yx-end	0			link		link	
X_3				0	link				
X_4					1				
Y_2					yx-end	0			
X_5							1		
Y_3		match						1	link
Y_4							yx-end		1

Figure 6

Imagine that the following sequence of commands is executed.

$START_1(X_1, X_2, Y_1)$
$GROW_1(Y_1, X_2, X_3, X_4, Y_2)$
$GROW_2(Y_1, X_2, X_5, Y_3, Y_4)$
$MATCH_1(Y_4, X_5, Y_3, X_2)$
$MATCH_0(Y_3, X_2, Y_1, X_1)$
$LEAK(Y_1, X_1)$

Figure 6 displays the matrix after this sequence has been executed.

We attempt to match corresponding x and y sequences by working from the bottom of the tree to the top. This seems easier than working down from the root, since there is a unique chain of links to follow from any node to the root in each tree, whereas working down from the root, it is not clear how to arrange to follow corresponding paths through the two trees. The $START_i$ and $GROW_i$ commands start matching two corresponding sequences by matching their last symbols. The $MATCH_b$ commands then compare the two predecessors (i.e., ancestors in the tree) of any pair of matched nodes.

The **leak** right can be entered if and only if matching proceeds all the way up to the root nodes. Next, we show that this can happen if and only if the Post Correspondence Problem has a solution; this is known to be a recursively unsolvable problem, Post 1946. Thus, we will have shown that is recursively unsolvable whether or not this protection system is safe for the right **leak** and the empty initial configuration.

Notation . Let $\emptyset$ be the empty configuration $(\emptyset, \emptyset, \emptyset)$. For any configuration (S, O, P), and any $X \in S$, let $A(X) = \{Y \in S \mid \textbf{link} \in P[Y, X]\}$. (In our tree interpretation, $A(x)$ is the parent of node (X, X).) We may extend this notation by defining $A^{i+1}(X) = A(A^i(X))$. Let $C(X)$ be the contents of $P[X, X]$.

Lemma 2.4.1. *If $\emptyset \vdash^* Q = (S, O, P)$, then for all $X \in S$ we have*

1 $P[X, X] = \{\textbf{0}\}$ *or* $\{\textbf{1}\}$
2 $| A(X) | \leq 1$
3 $A(X) = 0$ *if and only if there is a Y such that* **start** $\in P(X, Y)$ *or* **start** $\in P(Y, X)$. *Furthermore, any such Y is unique.*
4 *For each $Y \in S$, if* **yx-end** $\in P(Y, X)$ *then there exist $m \geq 1$, $i_1, \ldots, i_m$ each i_j between 1 and n such that*

$$x = x_{i_1} \ldots x_{i_m} = C(A^{lg(x)-1}(X)) \ldots C(A(X))C(X).$$

5 *For each $Y \in S$, if* **match** *is in $P(Y, X)$ then there exist $m \geq 1$, $X', Y' \in S$ such that* **yx-end**

$\in (Y', X')$, $Y = A^{m-1}(Y')$, $X = A^{m-1}(X')$, and for each $j, 0 \leq j < m$, we have $C(A^j(X')) = C(A^j(Y'))$. † Also $y = y_{i_1} \ldots y_{i_m} = C(A^{lg(y)-1}(Y)) \ldots C(A(Y))C(Y)$, and **start** $\in P(A^{lg(y)-1}(Y), A^{lg(x)-1}(X))$.

This claim formalizes the discussion above. In (1), it is shown how strings are encoded on the diagonal. In parts (2) and (3) every node has a unique parent, except the root. In part (3), the root and only the root of every tree is paired with some other tree, and that tree is uniquely determined. **yx-end** joins the ends of corresponding sequences in paired trees according to part (1). Finally, in (5), **matching** proceeds along corresponding sequences.

We are now ready to do the formal argument.

The argument is an induction on the length of the computation of $\emptyset \vdash^* Q$.

Basis: The argument is trivial for

$$\emptyset \overset{0}{\vdash} \phi$$

Induction Step: Assume that $Q = (S, O, P)$ satisfies the conditions. We will show that

$$Q \underset{\alpha}{\vdash} Q'$$

where $Q' = (S', O', P')$ implies that Q' does also.

If $\alpha = START_i$, it is clear that all the rights entered are placed into created entries so that the "old portion" of P' is unchanged. ‡ Moreover, the "new portion" of P' satisfies conditions (1) through (5) by construction. There is no possible connection between the old and new portions of P' because

$$P'[X, X'] = P'[X', X] = 0$$

with $X \in S, X' \in S' - S$. Thus $\vdash_\alpha$ where $\alpha = START_i$ preserves (1) through (5).

If $\alpha = GROW_i$, it is clear that one of **0** and **1** is entered in each new diagonal element, and all other entries are made off the diagonal, so condition (1),

$$P[X, X] = \{0\} \quad \text{or} \quad \{1\}$$

will still hold. The **link** right is never entered in an old object, and only entered once in each new object, so condition (2) ($| A(X) | \leq 1$) still holds. The **start** right is not entered so (3) still holds. If **yx-end** $\in P'[Y, X]$ with † $Y, X \in S$ (not $S' - S$), then (4) holds in $P'[Y, X]$ with $Y, X \in S' - S$, then it is easy to see that (4) holds with x_i and y_i continuing the sequence ending at $(YEND, XEND)$. That is, we have **yx-end** $\in (YEND, XEND)$ and there exist $m \geq 1, i_1, \ldots, i_m$ such that

(i) $x = x_{i_1} \ldots x_{i_m} = C(A^{lg(x)-1}(XEND)) \ldots C(XEND)$

(ii) $y = y_{i_1} \ldots y_{i_m} = C(A^{lg(y)-1}(YEND)) \ldots C(YEND)$ and

(iii) **start** $\in (A^{lg(y)-1}(YEND), A^{lg(x)-1}(XEND))$.

Finally, (5) is not affected at all in the old portion of P' and moreover it holds vacuously in the new portion of P' except possibly for (Y_{m_i}, X_{l_i}) in the case where $y_{m_i} = x_{l_i}$. In that case it holds with $m = 1$. Thus, $\vdash_\alpha$ where $\alpha = \text{GROW}_i$ preserves properties (1) through (5).

If α is $MATCH_b$, we see that the **link** right is not entered anywhere so conditions (2) through (4) are not affected. The other rights are entered by this command off the main diagonal so property (1) is also unaffected. If α is

$$MATCH_b(Y, X, AY, AX)$$

then we must have had **match** in $P[Y, X]$. Then, by property (5), there must have been $m \geq 1, X', Y' \in S$ such that

---

† *There is a natural identification taking place here. If we concatenate the contents of the appropriate cells of the matrix, this line becomes something like* $x = x_1 x_2 = 011 =$ "0" "1" "1".

‡ We prefer to say "old portion" of P' rather than $P' \cap (S \times O \times 2^R)$ and "new portion" of P' instead of $P' \cap (S' - S) \times (O' - O) \times 2^R$.

† Assume that command $GROW_i$ is called with actual parameters $(YEND, XEND)$. The Y and X here are formal parameters.

$$\textbf{yx-end} \in P[Y', X']$$
$$Y = A^{m-1}(Y'), X = A^{m-1}(X')$$

and

$$C(A^j(X')) = C(A^j(Y'))$$

for each $j, 0 \leq j < m$. It is clear that after the $MATCH_b$ command is executed, similar conditions hold in P', since

$$AY = A(Y), AX = A(X),$$
$$C(AY) = C(AX),$$

and the other entries are unchanged, so (5) is satisfied by $m+1, X'$, and Y'.

If α is $LEAK$, then no rights are entered on the diagonal so property (1) still holds and **link**, **start** and **march** are not entered, so properties (2) - (5) are unaffected.

Thus, the induction is extended, and we see that Lemma 2.4.1 is true.

We are now ready to prove Theorem 2.4.1. Suppose the Post Correspondence Problem has a solution, say $(i_1, i_2, \ldots, i_m)$. Then commands

$$START_{i_1}, GROW_{i_2}, \ldots, GROW_{i_m}$$

could be executed with appropriate parameters so that the indicated solution is constructed. Since a solution ends with i_m, we certainly must have the "**enter match** ..." command in "$GROW_{i_m}$", so execution of several "$MATCH_b$" commands with appropriate parameters would result in placing the **match** right in the same position as the **start** right, thus allowing the $LEAK$ command to enter the **leak** right. Conversely, if **leak** is ever entered, it must be because **start** and **match** appear in the same position of the matrix. By property (5) of Lemma 2.4.1, we see that there must be some Y', X' such that **yx-end** $\in P[Y', X']$, and their predecessors match. But then by property (4) for Y', X', we see that their predecessors must be corresponding sequences of x_i's and y_i's, i.e., the Post Correspondence Problem must have a solution. Thus, the protection system is safe for the right **leak** and the initial configuration $\emptyset$ if and only if the Post Correspondence Problem has no solution, and hence safety is recursively unsolvable. ∎

A study of the proof reveals that most of the commands have one or two conditions attached to them. It is necessary to use one command which requires five conditions. By using some coding tricks, these commands may be replaced by six commands each of which needs two conditions. This leads to the following result.

Theorem 2.4.2. *The safety question for monotonic protection systems is undecidable even when each command has at most two conditions.*

Proof. The construction is similar to the one used in the proof of the preceding theorem, except that a more complex sequence of commands must be used for the matching. The set of generic rights is

$$R = \{0, 1, \textbf{link}, \textbf{start}, \textbf{match}, \textbf{yx-end}, \textbf{leak}, \textbf{my}, \textbf{myx}, \textbf{myx0}, \textbf{myx1}\}$$

The set of commands includes the $START_i, GROW_i$, and $LEAK$ commands of the previous proof. Note that these commands all have only one or two conditions. The $MATCH_b$ commands, which had five conditions, are replaced by the following six commands having two conditions each.

command $FOLLOWY(Y, X, AY)$
  if match $\in (Y, X)$ and **link** $\in (AY, Y)$
  then
    enter my into (AY,X)
end
command $FOLLOW(AY, X, AX)$
  if my $\in (AY, X)$ and **link** $\in (AX, X)$
  then

enter myx into (AY, AX)

end

For each $b \in \{0,1\}$, we have

command $GETY_b(AY, AX)$

if myx $\in$ (AY,AX) and $b \in$ (AY,AY)

then

enter myxb into (AY, AX) †

end

for each $b \in \{0,1\}$, we have

command $MATCHX_b(AY, AX)$

if myxb $\in (AY, AX)$ and $b \in (AX, AX)$

then

enter match into (AY, AX)

end

Next, we need a result which characterizes computation in the new system.

Lemma 2.4.2. *If* $\emptyset \vdash^* Q = (S, O, P)$, *then for each* $X \in S$, *we have (1) - (5) of Lemma 2.4.1 as well as the following conditions.*

(6) If **my** $\in P(Y, X)$ then there exists $Y' \in S$ such that $Y = A(Y')$ and **match** $\in P(Y', X)$.
(7) If **myx** $\in P(Y, X)$ then there exists $X' \in S$ such that $X = A(X')$ and **my** $\in P(Y, X')$.
(8) If **myxb** $\in P(Y, X)$ with $b = 0, 1$ then myx $\in P(Y, X)$ and $b \in P(Y, Y)$.

Proof. Since (1) through (4) of Lemma 2.4.1 were unaffected by the $MATCH_b$ command in the previous construction, the absence of that command does not matter. Similarly, the six new commands do not enter the **start, link** , or **yx-end** rights so (2) = (4) are not affected. Since these commands don't enter rights on the diagonal, they preserve property (1) also. Since the original commands do not use any of the rights **my, myx, myx0,** or **myx1** , they will not effect (6) - (8). Thus (6) - (8) just reflect the conditions and actions of the commands $FOLLOWY$, $FOLLOWX$, and $GETY_b$ respectively, so they will hold. Finally, looking at the $MATCHX_b$ commands, and combining its conditions with properties (8), (7) and (6) we see that $MATCHX_b$ enters the **match** right in (Y,X) just in case there exist $Y', X' \in S$ such that $Y = A(Y'), X = A(X'), C(Y) = C(X)$, and **match** $\in (Y', X')$. These are precisely the conditions which allow us to inductively extend property (5). Hence the claim is proven.

Now to complete the proof of Theorem 2.4.2.

Proof of the Theorem. The argument parallels the proof of Theorem 2.4.1 but uses Lemma 2.4.2 instead of Lemma 2.4.1.

Theorem 2.4.1 shows that the safety question for monotonic protection systems is undecidable, even if each command has at most two conditions. However, in many important practical situations, commands need only one condition. For example, a procedure for updating a file may only need to check that the user has the "update" right to the file. In contrast to the undecidability of the cases discussed in the preceding section, the safety question is decidable if each command of a *monotonic* protection system has at most one condition.

Definition. A *mono-conditional* protection system is one in which each command has at most condition.

Mono-conditional protection systems are much more complicated than one might anticipate. It is still not known whether or not the safety problem is solvable for such systems.

We state the best result known on this topic without proof. More details may be found in Harrison and Ruzzo, 1978.

Theorem 2.4.3. *Safety of mono-conditional protection systems with* **create, enter,** *and* **delete** *(but without* **destroy***) commands is decidable.*

† If $b = 0$, then **myx0** is to be entered.

3. Logic and Protection Systems

Introduction

The results of the previous sections are not encouraging. Can we do better by changing our perspective? Suppose we just want to know if a system is secure. Perhaps we can prove a system to be correct so that potential users can be guaranteed to be safe in using the system. Even partial results would be helpful because certain subsystems could be certified and then special considerations could be used in the rest of the system. For example, if the file system were provably secure and the mail system was not, it would give some guidance to a system administrator who might decide to shut off the mail system when sensitive work was to be done.

We shall now begin to investigate trying to prove that systems are safe.

3.1 Logical Theories and Safety

In slightly different but more mathematical language, Theorem 2.3.2 can be restated as follows.

Theorem 3.1.1. *The set of safe protection systems is not recursive.*

We can generate a list of all unsafe systems by systematically enumerating all protection systems and all sequences of commands in each system, outputting the description of any system for which there is a sequence of commands causing a leak. Hence, the following is true:

Theorem 3.1.2. *The set of unsafe protection systems is recursive enumerable.*

We cannot, of course, also enumerate all safe systems, for a set is recursive if and only if both it and its complement are recursively enumerable. The bounded case, discussed in Theorem 2.3.3, is recursive though not computationally attractive.

Could we avoid the problems inherent in these results by shifting our perspective towards proving properties of the system in some particular formal language rather than dealing with algorithms directly? The idea is similar to some of the work on program verification.

To pursue this idea we shall say that a *formal language* L is a recursive subset of the set of all strings over a given finite alphabet; the members of L are called *sentences*.

A *deductive theory* T over a formal language L consists of a set A of axioms, where $A \subseteq L$, and a finite set of *rules of inference*, which are recursive relations over L. The set of *theorems* of T is defined inductively by:

(a) if t is any axiom (i.e. if $t \in A$), then t is a theorem of T; and

(b) if $t_1, \ldots, t_k$ are theorems of T and $< t_1, \ldots, t_k, t > \in R$ for some rule of inference R, then t is a theorem of T.

Thus every theorem t of T has a *proof* which is a finite sequence $< t_1, \ldots, t_n >$ of sentences such that $t = t_n$ and each t_i is either an axiom or follows from some subset of $t_1, \ldots, t_{i-1}$ by a rule of inference. We write $T \vdash t$ to indicate that t is a theorem of T or is provable in T.

Two theories T and T' are said to be *equivalent* if they have the same set of theorems though not necessarily the same axioms or rules of inference.

A theory T is *recursively axiomatizable* if it has (or is equivalent to a theory with) a recursive set of axioms. The set of theorems of any recursively axiomatizable theory is recursively enumerable: we can generate all finite sequences of sentences, check each to see if it is a proof, and enter in the enumeration the final sentence of any sequence which is a proof.

A theory T is *decidable* if its theorems form a recursive set.

Since the set of safe protection systems is not recursively enumerable, it cannot be the set of theorems of a recursively axiomatizable theory. This means that the set of all safe protection systems cannot be generated effectively by rules of inference from a finite (or even recursive) set of safe systems. This does not rule out the possibility of effectively generating smaller, but still interesting classes of safe systems. This observation can be refined, as we proceed to do, to establish further limitations on any recursively axiomatizable theory of protection.

Definition A *representation of safety* over a formal language L is an effective mapping $p \rightarrow t_p$ from protection systems to sentences of L.

We wish to interpret t_p as a statement of the safety of the protection system p. The following definition captures what is necessary before such a definition is useful.

Definition A theory T is *adequate for proving safety* if there is a representation $p \rightarrow t_p$ of safety such that $T \vdash t_p$ if and only if p is safe.

3.2 Incompleteness of Protection Systems

The notions of the previous section are quite appealing but analogs of the classic Church and Gödel theorems for the undecidability and incompleteness of formal theories of arithmetic hold for formal theories of protection systems.

Theorem 3.2.1. *Any theory T adequate for proving safety must be undecidable.*

This theorem follows from Theorem 3.1.1 by noting that, were there an adequate decidable theory T, we could decide whether or not a protection system p was safe by checking whether or not $T \vdash t_p$.

Theorem 3.2.2. *There is no recursively axiomatizable theory T which is adequate for proving safety.*

Proof. This theorem follows from Theorems 3.1.1 and 3.1.2. If T were adequate and recursively axiomatizable, we could decide the safety of p by enumerating simultaneously the theorems of T and the set of unsafe systems; eventually, either t_p will appear in the list of theorems or p will appear in the list of unsafe systems, enabling us to decide the safety of p. ∎

Theorem 3.2.2 shows that, given any recursively axiomatizable theory T and any representation $p \rightarrow t_p$ of safety, there is some protection system whose safety either is established incorrectly by T or is not established when it should be. This result in itself is of limited interest for two reasons: it is not constructive (i.e., it does not show how to find such a p); and, in practice, we may be willing to settle for inadequate theories as long as they are sound, that is as long as they do not err by falsely establishing the safety of unsafe systems.

The next theorem overcomes the first limitation, showing how to construct a protection system p which is unsafe if and only if $T \vdash t_p$; the idea is to design the commands of p so that they can simulate a Turing machine that "hunts" for a proof of the safety of p; if and when a sequence of commands finds such a proof, it generates a leak. If the theory T is sound, then such a protection system p must be safe but its safety cannot be provable in T.

Definition. A theory T together with a representation $p \rightarrow t_p$ of safety is *sound* if and only if p is safe whenever $T \vdash t_p$.

Theorem 3.2.3. *Given any recursively axiomatizable theory T and any representation of safety in T, one can construct a protection system p for which $T \vdash t_p$ if and only if p is unsafe. Furthermore, if T is sound, then p must be safe, but its safety is not provable in T.*

Proof. The proof of Theorem 2.3.2 shows how to define, given an indexing $\{M_i\}$ of Turing machines and an indexing $\{p_i\}$ of protection systems, a recursive function f such that

(a) M_i halts if and only if $p_{f(i)}$ is unsafe.

Since T is recursively axiomatizable and the map $p \rightarrow t_p$ is computable, there is a recursive function g such that

(b) $T \vdash t_{p_i}$ if and only if $M_{g(i)}$ halts;

the Turing machine $M_{g(i)}$ simply enumerates all theorems of T, halting if t_{p_i} is found. By the recursion theorem, Rogers, 1967, one can effectively find an index j such that

(c) M_j halts if and only if $M_{g(f(j))}$ halts.

Combining (a),(b), and (c), and letting $p = p_{f(j)}$, we get

(d) $T \vdash t_p$ if and only if $M_{g(f(j))}$ halts
if and only if M_j halts
if and only if $p = p_{f(j)}$ is unsafe

as was to be shown.

Now suppose that T is sound. Then t_p cannot be a theorem of T lest p be simultaneously safe by soundness and unsafe by (d). Hence t_p is not a theorem of T and so p is safe by (d). ∎

The unprovability of the safety of a protection system p in a given sound theory T does not imply that the safety of P is unprovable in every theory. We can, for example, augment T by adding t_p to its axioms. However, Theorem 3.2.3 states that there will exist another safe p' whose safety is unprovable in the new theory T'. In other words, this abstract view shows that systems for proving safety are necessarily incomplete: no single effective deduction system can be used to settle all questions of safety.

The process of extending protection theories to encompass systems not provably safe in previous theories creates a progression of ever stronger deductive theories. With the stronger theories, proofs of safety can

be shortened by unbounded amounts relative to weaker theories. This phenomena is known in logic and complexity theory.

Theorems 3.2.2 and 3.2.3 force us to settle for attempting to construct sound, but necessarily inadequate, theories of protection. What goals might we seek to achieve in constructing such a theory T? At the least, T should be nontrivial; theories that were sound because they had no theorems would be singularly uninteresting. We might also hope that the systems whose safety was provable in T, when added to the recursively enumerable set of unsafe systems, would form a recursive set. If this were so, then we could at least determine whether T were of any use in attempting to establish the safety or unsafety of a particular protection system p before beginning a search for a proof or disproof of the safety of P. The next theorem shows that this hope cannot be fulfilled.

Theorem 3.2.4. *Given any recursively axiomatizable theory T and any sound representation of safety in T, the set*

$$X = \{ p \mid T \vdash t_p \text{ or } p \text{ unsafe} \}$$

is not recursive.

Proof. If X were recursive, then the safety of a protection system p could be decided as follows. First, we check to see if p is in X. If it is not, then it must be safe. If it is, then we enumerate simultaneously the theorems of T and the unsafe systems, stopping when we eventually find either a proof of p's safety or the fact that p is unsafe. ∎

3.3 Finiteness Conditions and Reducibilities

If we consider finite systems in which the number of objects * cannot grow beyond the number present in the initial configuration, then the safety question becomes decidable, although any decision procedure is likely to require enormous amounts of time (cf. Theorem 2.3.3.). This doubtless rules out practical mechanical safety test for these systems. However this does not rule out successful safety tests constructed by hand. Ingenious or lucky people might be able to find proofs faster than any mechanical method. We show now that even this hope is ill-founded.

Although we can always obtain shorter safety proofs by choosing a proof system in which the rules of inference are more complicated, it makes little sense to employ proof systems whose rules are so complex that it is difficult to decide whether an alleged proof is valid. We shall regard a logical system as "reasonable" if we can decide whether a given string of symbols constitutes a proof in the system in time which is a polynomial function of the string's length. Practical logical systems are reasonable by this definition. We show now that, corresponding to any reasonable proof system, there are protection systems which are bounded in size, but whose safety proofs or disproofs cannot be expected to have lengths bounded by polynomial functions of the size of the protection system.

Theorem 3.3.1. *For the class of protection systems in which the number of objects† is bounded, safety (or unsafety) is polynomially verifiable by some reasonable logical system if and only if* **PSPACE** $=$ **NP** , *that is, if and only if any problem solvable in polynomial space is solvable in polynomial time.* †

Proof. By Theorem 2.3.3, the safety and unsafety problems for systems of bounded size are both in **PSPACE** . Hence, if **PSPACE** $=$ **NP** , then there would be **NP**-time Turing machines to decide both safety and unsafety. Given such machines, we could define a reasonable logical system in which safety and unsafety were polynomially verifiable: the "axioms" would correspond to the initial configurations of the Turing machines and the "rules of inference" to the transition tables from the machines.

Also by Theorem 2.3.3, any problem in **PSPACE** is reducible to a question concerning the safety (or unsafety) of a protection system whose size is bounded by a polynomial function of the size of the original problem. Now if the safety (or unsafety) of protection systems with bounded size were polynomial verifiable, we could decide safety (or unsafety) in **NP**-time by first "guessing" a proof and then verifying that it was a proof (performing both tasks in polynomial time). By Theorem 2.3.3, we could then solve any problem in **PSPACE** in **NP**-time, showing that **PSPACE** $=$ **NP** .

Since the above result applies equally to proofs of safety and unsafety, one must expect that there are systems for which it will be just as difficult and costly to penetrate the system as to prove that it can (or cannot) be done. In mono-operational systems, however, the situation is quite different.

Theorem 3.3.2. *The safety of mono-operational systems is polynomially verifiable.*

Proof. This result follows from Theorem 2.3.1 whose proof shows that the unsafety question of mono-operational systems is solvable in **NP**-time. Although we simply observe that to demonstrate unsafety, one need only exhibit a command sequence leading to a leak. We know that there are short unsafe command sequences if any exists at all: an upper bound on the length of such sequences was given in equation (2). Thus an unsafe sequence (if it exists) has a length bounded by a some polynomial function of the system size. ∎

By Theorems 2.3.3 and 3.3.2, proofs of unsafety for mono-operational systems are short, but the time to find the proofs cannot be guaranteed to be short; at the worst we might have to enumerate each of the sequences of length at most $g(m+1)(n+1)$ that could produce a leak. However, while proofs of unsafety are short for mono-operational systems, proofs of safety are not.

Theorem 3.3.3. *For mono-operational systems, safety is polynomial verifiable if and only if* **NP** *is close under complementation.* ‡

---

* hence, subjects also

† **PSPACE** *is the class of all problems which can be solved in polynomial space. It is known that any problem which can be solved nondeterministically in polynomial space is in* **PSPACE** , *but it is widely believed that* **PSPACE** $\neq$ **NP** .

‡ *It is unlikely that* **NP** *is closed under complement.*

Proof. If **NP** were closed under complement, then safety would be in **NP** because unsafety is in **NP** by Theorem 3.3.1. Thus there would be a nondeterministic Turing machine for checking safety in polynomial time, which would demonstrate that safety is polynomial verifiable.

Conversely, suppose that safety were polynomially verifiable. We could then construct a nondeterministic Turing machine which would guess a proof of safety and then check it in polynomial time; hence safety would be in **NP** . But unsafety is in **NP** by Theorem 3.3.1 and if any **NP** complete problem has its complement in **NP** , then **NP** is closed under complement. ∎

These result imply that system penetrators have a slight advantage when challenging mono-operational systems: any system that can be penetrated has a short command sequence for doing so. However, it may still take enormous amounts of time to find such sequences, as no systematic method of finding an unsafe command sequence in polynomially bounded time is likely to be found.

4. Conclusions

Theoretical Conclusions

On the one hand, our desire to model operating systems as simply as possible led us to consider the take-grant system. This is a straightforward system to model and has a comparatively simple safety problem.. The only unattractive feature of this model is that it dealt with only one trivial operating system. In order to get something practically useful, it is necessary to extend this work to consider a family of such models, ever more complicated and expressive. This line of research has been investigated by Larry Snyder and his collaborators. The results are interesting but the models stop far short of real operating systems.

On the other hand, the HRU model is both expressive and very general. Unfortunately undecidable problems and unproven conjectures abound. The problem is that HRU systems are very complicated models and it is not clear that mathematical results are going to be useful for operating systems work. The techniques which are needed in order to express operating systems constructs in the model are quite ingenious but somewhat unnatural. It may be unrealistic to expect programmers to express operating system features in these formalisms.

The most important future theoretical research direction is to invent new models which combine natural modes of expression of operating system constructs with efficient or at least feasible algorithms.

Practical Conclusions

There are certain inescapable conclusions from the theoretical work reported here.

1. Do not try to be too general! The HRU results tell us that there can be no algorithm to tell if some right can even get into a certain cell of the access matrix. Any complicated schemes which depend on such information cannot be implemented in full generality.
2. Do not expect to be able to verify operating systems! A practical argument to justify this maxim is in the experience of a decade of research into program verification. Of course, the theoretical results of Section 3 are relevant also.
3. Be sceptical of the literature! Some of the recent material in the literature has suggested that secure operating systems are now here. This is particularly true of the 1983 special issue * of the *Computer* on secure computation. There are a number of techniques which are known which improve the security of operating systems. Unfortunately, relatively few of them are general and this area is an art more than a science.
4. Do not give up! While the theoretical results are discouraging, they are not fatal. There are many techniques which can be used to improve glaring weaknesses in present systems. One will be mentioned directly.

The present situation is not unlike the problem that one faces with respect to physical security. There is a constant tension between the abilities of a criminal and the counter measures that (say) a homeowner can take. One can envision a classification of security according to the complexity of penetration. For example, certain locks are known to take a certain level of time to break using hand tools. When power tools are available, the classification level changes. For example, a certain kind of lock may hold for five minutes under attack with power tools. Perhaps it is possible to classify certain systems with respect to the techniques needed to penetrate them.

It now appears that the best simple technique that one can utilize is some sort of encryption method to protect sensitive data bases and ordinary computer files. It is possible to encode encrypted data into such form as it can be transmitted across a mail system. For example, in the UNIX system, there are several encryption methods available and there are companion functions, e.g. ***uuencode*** and ***uudecode***, which allow encrypted messages to be transmitted mailed through various networks. Users should be aware however, that it is known how to break ***crypt***. Moreover the program which is distributed with UNIX which appears to implement DES does ***not*** do so.

It is hard to understand why large commercial systems, particularly those involved in electronic fund transfer and commercial data bases, are not protected more. As time goes by, it will be more essential to provide some sort of security mechanisms for these commercial systems because it it expected that computer crime will increase.

5. Acknowledgements

The paper is the result of several years of investigation and involves work with a number of people. It has been a pleasure to acknowledge stimulating conversations and joint work with Richard DeMillo, Dorothy Denning, Peter Denning, Steve Garland, Anita Jones, Dick Lipton, Larry Ruzzo, Larry Snyder, and Jeff Ullman.

* *Computer*, **16**, Number 7, July, 1983.

6. Bibliography

Aho, A. V., Hopcroft, J. E., and Ullman, J. D. (1974). *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Company, Reading, Mass..

Bishop M.(1981). Hierarchical Take-Grant Systems. *Proceedings of the Eighth Symposium on Operating System Principles*, pp. 109-122.

Bishop, M. and Snyder, L. (1979). The Transfer of Information and Authority in a Protection System, *Proceedings of the Seventh Symposium on Operating System Principles*, pp. 45-54.

Budd, T. and Lipton, R. J. (1978). On Classes of Protection Systems, in DeMillo, R. A. et al (editors), *Foundations of Secure Computation*, Academic Press, Inc. New York.

Cheheyl, M. H., Gasser, M., Huff, G. A., and Millen J. K. (1981). Verifying Security. *ACM Computing Surveys.* **13**, pp. 279-340.

Cohen, Ellis S. (1976). *Problems, Mechanisms and Solutions*, Ph. D. Dissertation, Carnegie-Mellon University.

Denning, P. J. and Graham, G. S.(1972). Protection – principles and practise, *Proc. AFIPS 1972 SJCC*, AFIPS Press, Montvale, N.J. pp. 417-429.

Denning, D. E., Denning, P. J., Garland, S. J., Harrison, M. A., and Ruzzo, W. L.(1977). Proving Protection Systems Safe, Technical Report 209, Computer Science Department, Purdue University.

DeMillo, R. A. et al (editors), (1978). *Foundations of Secure Computation*, Academic Press, Inc. New York.

Harrison, M. A. (1978). *Introduction to Formal Language Theory*, Addison-Wesley Publishing Company, Reading, Mass..

Harrison, M. A. and Ruzzo, W. L. (1978). Monotonic Protection Systems, in DeMillo, R. A. et al (editors), *Foundations of Secure Computation*, Academic Press, Inc. New York.

Harrison, M. A., Ruzzo, W. L., and Ullman, J. D. (1976). Protection in Operating Systems, *Communications of the Association for Computing Machinery*, **19**, pp. 461-471.

Jones, Anita K.(1973). *Protection in Programmed Systems*, Ph. D. Dissertation, Carnegie-Mellon University.

Jones, A. K. (1978). Protection Mechanism Models : Their Usefulness. In DeMillo, R. A. *et al* (editors), (1978). *Foundations of Secure Computation*, Academic Press, Inc. New York, pp. 237-254.

Jones, A. K. and Lipton, R. J. (1978). The Enforcement of Security Policies for Computation. *Journal of Computer and System Sciences.* **17**, pp. 35-55.

Jones, A. K., Lipton, R. J., and Snyder, L. (1976). A Linear Time Algorithm for Deciding Security, in *Proceedings of the 17th Symposium on Foundations of Computer Science*, pp. 33-41.

Lampson, Butler W. (1971). Protection, *Proceedings of the Fifth Princeton Conference on Information Science and Systems*, pp. 437-443.

Landwehr, C. E. (1981). Formal Models for Computer Security. *ACM Computing Surveys.* **13**, pp. 247-278.

Landwehr, C. E. (1983). The Best Available Technologies for Computer Security. *Computer.* **13**, pp. 86-100.

Lipton, R. J. (1978). On Classes of Protection Systems. In DeMillo, R. A. *et al* (editors), (1978). *Foundations of Secure Computation*, Academic Press, Inc. New York, pp. 281-296.

Lipton, R. J. and Snyder, L. (1977). A linear time algorithm for deciding subject Security, *Journal of the Association for Computing Machinery*, **24**, pp. 455-464.

Lipton, R. J. and Snyder, L. (1978). On Synchronization and Security. In DeMillo, R. A. *et al* (editors), *Foundations of Secure Computation*, Academic Press, Inc. New York, pp. 367-386.

Millen, J. K. (1978). Constraints, Part II- Constraints and Multi-Level Security. In DeMillo, R. A. *et al* (editors), ***Foundations of Secure Computation***, Academic Press, Inc. New York, pp. 205-222.

Minsky, N. (1978). The Principle of Attenuation of Privileges and Its Ramifications. In DeMillo, R. A. *et al* (editors), ***Foundations of Secure Computation***, Academic Press, Inc. New York, pp. 255-278.

Popek, G. J. and Kline, C. (1975). A Verifiable Protection System. ***Proceedings of an International Conference on Reliable Software***, pp. 294-304.

Post, E. L. (1946). A variant of a recursively unsolvable problem, ***Bulletin of the American Mathematical Society***, **52**, pp. 264-268.

Rogers, H. Jr. (1967). ***Theory of Recursive Functions and Effective Computability***, Mc-Graw-Hill Book Co., New York.

Snyder,L. (1977). Analysis and Synthesis in the Take-Grant System. ***Proceedings of the Sixth Symposium on Operating System Principles***, pp. 141-150.

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

Snyder,L. (1981). Theft and Conspiracy in the Take-Grant Protection Model. ***Journal of Computer and System Sciences***. **23**, pp. 333-347.