3. Computer System Security Models

In this chapter, we will briefly survey security models which deal with computer system security. In contrast to the previous chapter, there is very little in the way of orderly development that can be seen in the field as a whole -- models and designs are motivated mainly by sets of competing requirements and costs, and concepts tend to reappear in several disguises. It is certain that the central issues have not yet been clearly articulated. Rather than try to give a complete picture of the state-of-the-art in computer system security, we will concentrate on models which have the most relevance for the protocol applications: the access control and multilevel security models for secure operating systems and the security models for statistical databases.

3.1 Operating System Models. 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, and 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. In order to be able to approach the security problem in a mathematical way, we must make a number of simplifying assumptions. The remarkable part of our treatment will be that even with gross simplifications, the security question remains 'hard' in a technical sense.

We shall concentrate on the computer and operating system in making our first model.

In the accepted terminology, there are two (not disjoint) sets of entities which must be dealt with in the model. The first sort of entity is the <u>object</u>. It is enough to think of an object as any entity in the system which has a logically independent existence; a typical object may a terminal, a user, a program, a file, or the supervisory routine. An object is significant in that it has an identifiable name in the system and it may be operated upon by other objects. Of course, not every other object will be able to operate on a given object. Some objects, for example, are passive. By contrast, the programs which reside on files are <u>active</u> entities. They read files, write files and call other programs. To distinguish these kinds of entities, we will call the active entities <u>subjects</u>. In a completely unprotected environment, a subject can access any object without restriction. Such a situation obviously raises havoc when users must be protected from each other. Therefore, a properly designed system should associate with a user, the set of capabilities which that user enjoys with respect to every other object. The emphasis is on the access to an object by a subject. Our goal is to arrive at a model which is rich enough to model actual systems but sufficiently restricted so that one can utilize mathematical techniques.

3.1.1 A Uniform Model of Protection. 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 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.

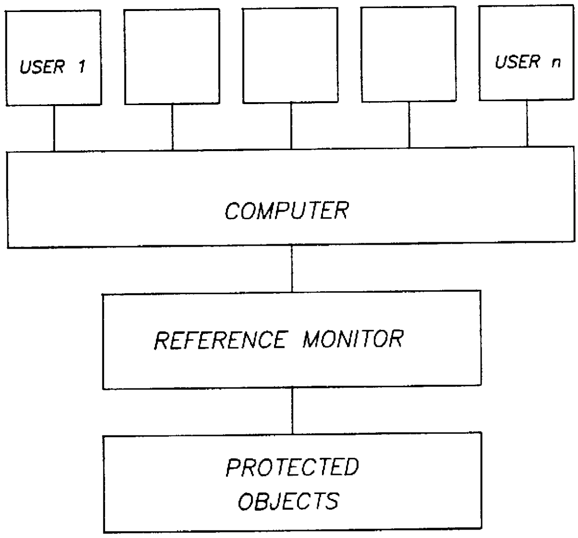


Figure 3.1
A computer system with a reference monitor.

A <u>protection system</u> 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 $r_1 \in (X_{s_1}, X_{o_1}) \land$

$r_2 \in (X_{s_2}, X_{o_2}) \land$

...

$r_m \in (X_{s_m}, X_{o_m})$:

then

op_1

op_2

...

op_n

end

Or, if m is zero, simply

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

op_1

op_2

...

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 following primitive operations.

enter r **into** (X_s,X_o)

create subject X_s

create object X_o

delete r **from** (X_s,X_o)

destroy subject X_s

destroy object X_o

By convention $r,r_1,r_2,\ldots,r_k$ denote generic rights and $s,s_1,s_2,\ldots,s_m$ and $o,o_1,o_2,\ldots,o_m$ are integers between 1 and k. 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.

A <u>configuration</u> 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, as shown in Figure 3.2. P[s,o] is a subset of R, the set of generic rights

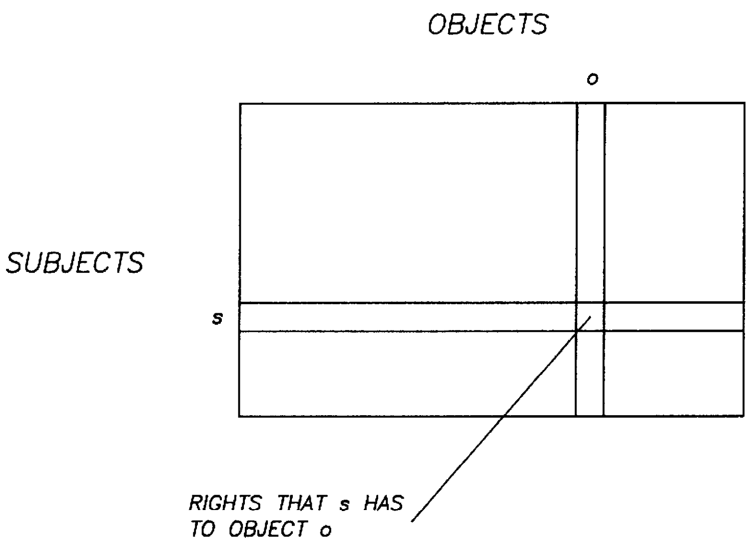


Figure 3.2
Access Matrix

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

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 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 ε (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 similar procedures.

(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 ε (owner,file) ∧
          r ε (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 3.2. 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.

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:

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

(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 $P'[s,o]=P[s,o]-\{r\}$, or

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

command $\alpha(X_1,X_2,\ldots,X_k)$
  if $r_1 \in (X_{s_1},X_{o_1}) \wedge$
    ...
  $r_m \in (X_{s_m},X_{o_m})$
then
  op_1,
  op_2,
    ...
  op_n
end

Then we can say that

$$Q \mid\!-^{*}_{\alpha(x_1,\ldots,x_k)} Q'$$

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

$$Q = q_0 \Rightarrow^{*}_{op_1} Q_1 \;\; \Rightarrow^{*}_{op_2} \cdots \Rightarrow^{*}_{op_n} Q_n$$

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.

We say that $Q \mid\!-_{\alpha} Q'$ if there exist parameters $x_1, \ldots, x_k$ such that we have $Q \mid\!-^{*}_{\alpha(x_1,\ldots x_k)} Q'$; we say $Q \mid\!- Q'$ if there exists a command α such that $Q \mid\!-_{\alpha} Q'$.

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

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.

The ability of the model to describe the policies used in real systems has been demonstrated in the literature.

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.

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 exists 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

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

where op_i^* denotes op_i after 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 it is possible to 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.

Given a particular protection system and generic right r, we say that the initial configuration Q_0 is <u>unsafe</u> 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. 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. At least we have found a technically reasonably '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.

3.1.2 Some Theorems about Protection Systems. Now that we have a model, we will mention some of the results that are known about the safety question. First, the 'good news'. It is possible to find an algorithm to test for safety in such systems.

A protection system is <u>mono-operational</u> if each command's interpretation is a single primitive operation.

Theorem 1: There is an algorithm which decides whether or not a given mono-operational protection system and initial configuration is unsafe for a given generic right r.

The proof proceeds by analyzing computation sequences. Details of the proof may be found in the literature (See the Bibliographic Notes). It is not too difficult to solve this problem in polynomial time in the size of the initial matrix. On the other hand, a uniform solution when the commands are parameters of the problem makes the decision problem NP-complete.

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: It is undecidable whether a given configuration of a given protection system is safe for a given generic right.

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 (a finite state device capable of scanning symbols a_i written on squares of an unbounded length tape, erasing and writing new symbols onto the squares, moving tape position one square to the left or right and changing states) as shown in Figure 3.3, we encode it into the matrix as shown in Figure 3.4.

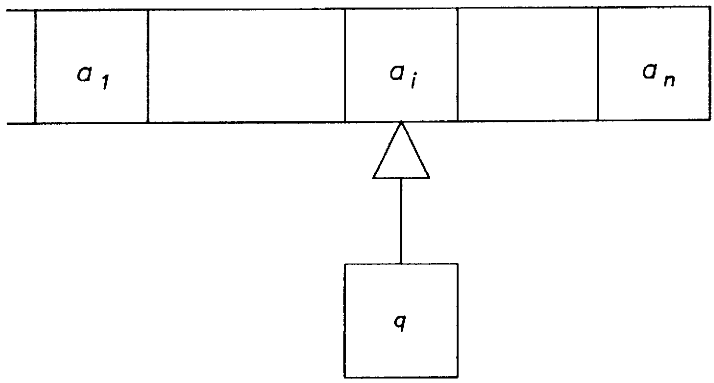


Figure 3.3
A Turing machine in state q reading a_i.

a_1	link				
	a_2	link			
		...	link		
			a_i, q	link	
				...	link
					a_n

Figure 3.4
Matrix which simulates machine in Fig. 3.3

To complete the proof, one must show how each move of the Turing machine can be accomplished by commands of the protection system. Moreover, the safety problem is solvable if and only if the Turing machine enters a designated final state. This last condition is unsolvable. The details of the construction are omitted here. Readers experienced with Turing machines or with machine-based complexity theory should now have a strong intuition about the class of theorems which hold.

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

Proof: A construction similar to Theorem 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.

Theorem 3 suggests that deciding safety for these systems probably requires exponential time.

The proof techniques which were employed above all make use of the diagonal of the access matrix in an essential way. What would happen if 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'?

Theorem 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.

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.

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 5: 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 would be to encode an instance of the Post correspondence problem 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'.

A study of previous proofs 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.

Theorem 6: The safety question for monotonic protection systems is undecidable even when each command has at most two conditions.

Theorem 6 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.

A <u>mono-conditional</u> protection system is one in which each command has at most one 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 without proof the best result known on this topic.

Theorem 7: Safety of mono-conditional protection systems with create, enter, and delete (but without destroy) commands is decidable.

3.1.3 Theories and Verifiability. In slightly different but more mathematical language, Theorem 2 can be restated as follows.

Theorem 8: The set of safe protection systems is not recursive.

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 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?

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

A <u>deductive theory</u> 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 <u>theorems</u> of T is defined inductively by:

1. if t is any axiom (i.e. if $t \in A$), then t is a theorem of T; and
2. 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.

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

We wish to interpret t_p as a statement of the safety of the protection system p. Therefore, we say that 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.

Analogs of the classic Church and Godel theorems for the undecidability and incompleteness of formal theories of arithmetic follow for formal theories of protection systems.

Theorem 10: Any theory T adequate for proving safety must be undecidable.

This theorem follows from Theorem 8 by noting that, were there an adequate decidable T, we could decide whether or not a protection system p was safe by checking whether or not $T \vdash t_p$.

Theorem 11: There is no recursively axiomatizable theory T which is adequate for proving safety.

This theorem follows from Theorems 8 and 9. 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 11 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.

A theory T together with a representation $p \rightarrow t_p$ of safety is <u>sound</u> if and only if p is safe whenever $T \vdash t_p$.

Theorem 12: 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 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

1. 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

2. $T \vdash t_{p_i}$ 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, one can effectively find an index j such that

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

Combining (1),(2), and (3), and letting $p = p_{f(j)}$, we get

4. $T \vdash t_p$ 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 (4). Hence $T \nvdash t_p$, and p is safe by (4).

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 12 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 deductive 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 11 and 12 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 13: 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.

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 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. To do this we generalize the class **NP** to the class of problems which can be solved in polynomial space. **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 time is in **PSPACE**, but it is widely believed that **PSPACE** $\neq$ **NP**.

Theorem 14: For the class of protection systems in which the number of objects (hence, also, subjects) 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 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 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 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 15: The safety of mono-operational system is polynomially verifiable.

Proof: This result follows from Theorem 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. It can be shown that there are short unsafe command sequences if any exist at all: an upper bound on the length of such sequences is $g(m+1)(n+1)$, where g is the number of generic rights, m the number of subjects and n the number of objects. Thus an unsafe sequence (if it exists) has a length bounded by a polynomial function of the system size.

By Theorems 3 and 15, 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 16: For mono-operational systems, safety is polynomial verifiable if and only if NP is closed under complementation.

Proof: If NP were closed under complement, then safety would be in NP because unsafety is in NP by Theorem 14. 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 in Theorem 14 and if any NP complete problem has its complement in NP, then NP is closed under complement.

These results 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.

3.1.4 A Decidable Model. The results of the preceding sections support skepticism toward proving systems safe, for there is no single, systematic, general approach to establishing the safety or unsafety of arbitrary protection systems. Hence we are forced to deal with approaches that are less general, or attack the problem from a different point of view. There are several possible approaches to take.

Despite the undecidability and intractability results, we can still try to prove particular protection systems safe. After all, the incompleteness, undecidability, and intractability results in number theory have not stopped mathematicians from trying to prove interesting theorems, and so our results for protection systems should not stop us from trying to prove the safety of interesting protection systems. Our results merely stand as a warning that proving systems safe is not likely to be easy.

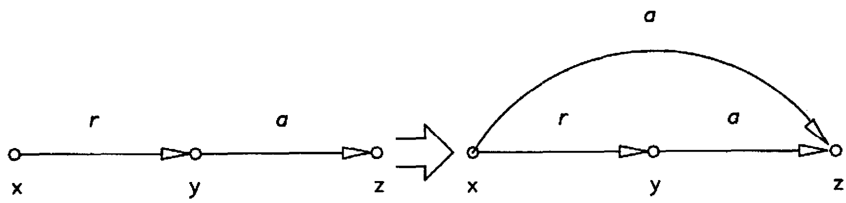
Rather than trying to guarantee the safety of a protection system, which might be expensive, we might instead seek to give shorter demonstrations that the system is 'probably safe' or 'safe beyond a reasonable doubt'. One possible approach might be to construct theories of protection which occasionally, though with very low probability, produced a 'proof' that an unsafe system was safe.

Recognizing the well-known fact that certain access paths may well be too expensive to eliminate, we might concentrate on trying to prove that any way of compromising a given protection system must likewise be too expensive. Given this approach, the question of central importance would not be whether we could prove that a system is safe, but whether we could prove that finding a breach of security is, say, NP-hard or even harder.

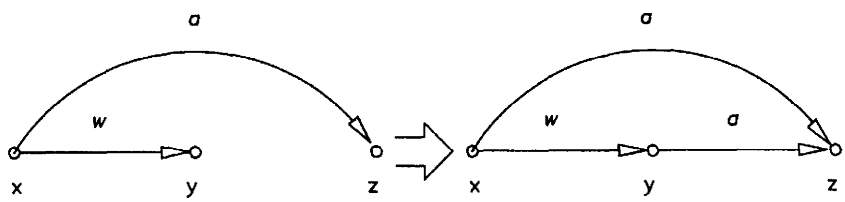
Perhaps the best approach would be to develop techniques for proving the safety of sufficiently simple protection systems. We now investigate a class of systems which possesses a linear time algorithm for deciding safety. These models are at the opposite end of the spectrum of operating systems models. Because they are less general, their expressability is of concern. We will attempt to explain their properties briefly here.

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. We write each explicit right as an arc label. Therefore, a label r on an arc from x to y means that there is a set of rights γ on an arc from x to y and $r \in \gamma$. Each graph may be transformed by any of the five rewriting rules.

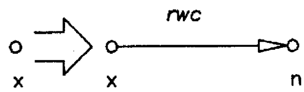
Take: Let x,y, and z be three distinct vertices in a protection graph, and let there be an arc from x to y with label γ such that $r \in \gamma$ and an arc from y to z with some label $\alpha \subseteq \{r,w,c\}$. The **take** rule allows one to add the arc from x to z with label α, yielding a new graph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



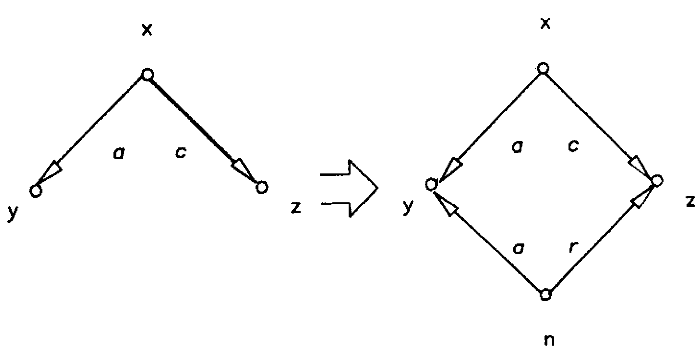
Grant: Let x,y, and z be three distinct vertices in a protection graph G, and let there be an arc from x to y with label γ such that $w \in \gamma$ and an arc from x to z with label $\gamma \subseteq \{r,w,c\}$. The **grant rule** allows one to add an arc from y to z with label α, yielding a new graph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



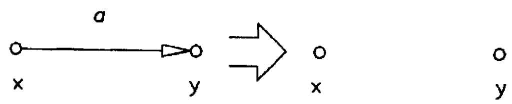
Create: Let x be any vertex in a protection graph; the create operation allows one to add a new vertex n and an arc from x to n with label {r,w,c}, yielding a new graph G'. Intuitively, x creates a new user that it can read, write, and call. In our representation



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 α can read the parameters. In our representation:



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



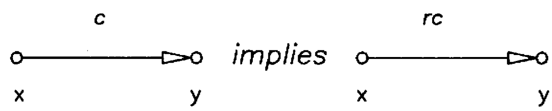
The **remove** rule is defined mainly for completeness, since real protection systems tend to have such a rule.

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, $G \vdash^* G'$ denotes the reflexive, transitive closure of $\vdash$.

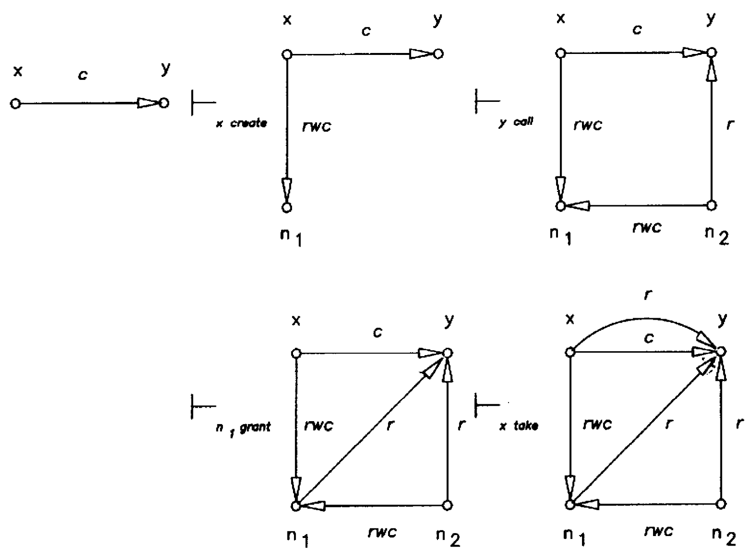
An important technical point is that this is monotone in the sense that if a rule can be applied, then adding arcs cannot change this.

The basic application of this model is to answer questions of the form: "Can p α 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.

Fact 1: In a protection graph G



Proof: Apply the following rules.



The present model is very restricted. Unlike the model in previous sections which was uniform with respect to many problems, the present model is quite specialized and was selected from the literature because of its previous occurrence in practice.

It is worth discussing what the technical results are in this case. 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 α q 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 17: 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 α q. 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 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 carried out. It should be noted that in the dichtomized model there are protection graphs that satisfy conditions 1 and 2 for which p can α q is false. There has been additional work in extending this model to handle progressively more complex cases.

3.2 Multilevel Security. There have been many attempts to use access control methods to encompass a broad range of security issues. For instance, attaching the capabilities to the subjects and requiring no subject to execute outside his specified capabilities implements the "principle of least privilege" which confines the potential damage of a subject's actions to that locality specified by his capability list. Attaching capabilities to subjects also admits an efficient implementation of access control methods. The technique of capability-based addressing associates the capabilities of a program with the addresses of segments, and accesses to data objects are always mediated by a process that does not allow access violations.

Although many famous security threats are met by access controls (e.g., access-controls meet the Trojan horse threat by requiring each program to execute only in a manner consistent with its capabiliites, so that a user program cannot execute in supervisor mode because its capabilities are not extensive enough to access its own runtime fields) they do not deal effectively with the central problem of multilevel security; that is, ensuring that undesirable information flow does not take place.

Figure 3.5 illustrates the generalized security problem in which a subject s creates an object o which operates within a specified environment or resource. Since the owner of the resource may be a subject to which s is accountable, the actions of s may have far-reaching consequences for the overall security state of the system.

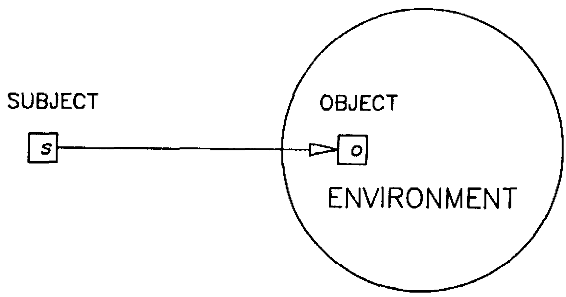


Figure 3.5
Generalized Security Threat

We have already described how various models can be applied to attempt to insure that o cannot access s's files illegally or to insure that o is protected from intrusion by s, but we have not yet dealt effectively with the situation in which o may be used as a transmitter of information. There are many channels through which o can "leak" information that s wants to keep secret. We will assume that o is owned by an enemy subject from whom the information that s passes to o is to be kept secret.

1. o may hold the sensitive data in memory, releasing it to the enemy at a later time;
2. o may write information directly into the enemy's file;
3. o may create its own file and grant the enemy access to the file;
4. o may send messages to a process 'owned' by the enemy;
5. if s is "billed" for using the environment, o may encode information in the bill;
6. o may use system interlocks as boolean variables (0 = unlocked, 1 = locked) to send bits to the enemy;
7. o may alter its performance characteristics to signal the enemy.

It is usual to classify these and other channels through which information may flow in a software system as <u>storage</u>(1,2), <u>legitimate</u> (3,4), <u>covert</u> (5,6,7) channels.

By far the greatest effort has been placed in handling the many legitimate channels for leakage that exist in a large software system. This problem has been called the <u>confinement problem</u>, and there have been many policies suggested for confining programs. One possibility is to isolate a program, disallowing the confined program from making any calls on other programs. A relaxed form of isolation is the policy of transitivity: if a program s calls an object o, then either o is "trusted" or o is itself confined.

The most widely adopted approach to the multilevel problem has been through the use of security kernels, that is, small, tightly constrained operating systems that deliver secure services to users by implementing system procedures which have been protected against system threats or by calling other software which has been previously identified as trusted. Figure 3.6 illustrates the subject/object/access configuration of a secure operating system using kernel.

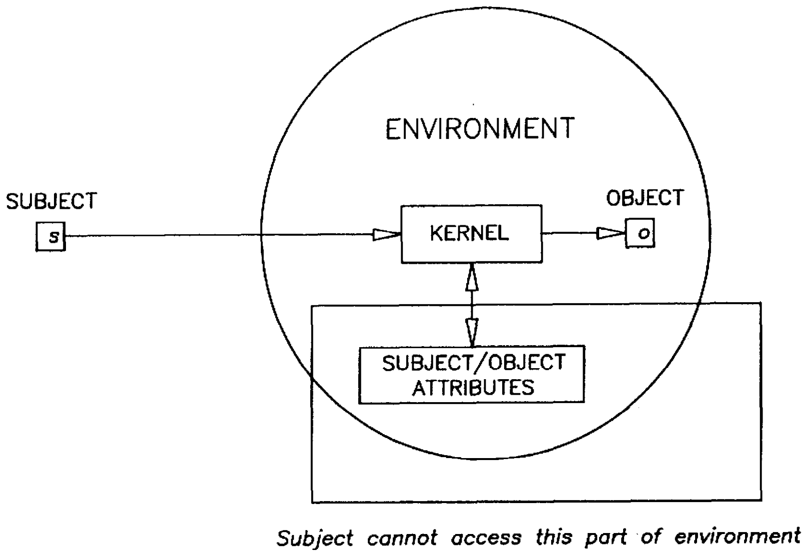


Figure 3.6
Subject/Object Access Through Kernel

The principal model on which current kernelized operating systems are constructed is the Bell and LaPadula model. In its broad outlines, the model is fairly simple. The multilevel system assigns security levels and integrity levels to each system object. A security level is specified by an ordered pair (x,y), where x is an authorization level (e.g., secret, top secret, etc.)

and y is a set of compartments (e.g., restricted, crypto). The usual situation is that the authorizations are linearly ordered and a partial ordering on the security levels is determined by the following condition:

$$(u,v) \leq (x,y) \text{ iff } u \leq x \text{ and } v \subseteq y.$$

The notion of integrity level is entirely analogous to that of security level. In general, one is given a set of levels $s_1, \ldots, s_n$ and a partial ordering on them. The operating system must allow reading and writing of information in the obvious directions between lower levels and higher levels in such a way that information can never flow from a high level to a lower level.

Early attempts at satisfying these conditions lead to so-called "highwatermark" systems in which access to classified documents by lower levels resulted in the reclassifying upwards of the documents -- leading, of course, to overclassification of documents. The Bell and LaPadula model attempts to satisfy these conditions by specifying two properties for each of the security and integrity problems.

The *-property: Writing is permitted only into an object with $\geq$ ($\leq$) the writer's security (integrity) level.

The Simple Security Condition: Reading is permitted only from an object with $\leq$ ($\geq$) the reader's security (integrity) level.

The security kernel is the authority whose responsibility it is to mediate the Bell and LaPadula model of multilevel security.

There have been a variety of attempts to produce working kernelized operating systems.

3.3 Databases and Inference. The interconnection between the gathering of data and the computer dates from the invention of the punched card which made it possible to complete the 1890 census within a decade. The 1980 census tabulations were available within six months after the completion of the census thanks to a century of advancements in our ability to collect and process data. This mode of information storage, however, gives rise to security problems of types unknown in the nineteenth century. In particular, the security measures to be discussed here were unnecessary, yet today security and integrity are major concerns. The purpose of this section is to discuss the inference mechanisms through which it is possible to extract information from statistical databases. This is the prototypical compromise problem for stored information. The underlying issues revolve around finding a method to satisfy the conflicting needs for statistical information about segments of society and the rights of the individual to privacy. The collection and analysis of data has long been a research tool in the social sciences, and few people would argue with the need for maintaining information from which general trends can be observed. On the other hand, it is quite easy to create scenarios in which the existence of large-scale databases containing information about individuals can be misused. In fact, just such abuse has been reported of a public agency's database, while a recent report of the General Accounting Office noted a number of weaknesses in the databases of the Social Security Administration.

To strike a balance, the Census Bureau of the United States does not issue its raw data, but rather makes available census tracts which are designed to give statistical information without providing information about individuals. And, because of the magnitude of the census information, individual privacy appears to be insured, simply because individuals are "lost" in the masses of data.

Such intuitions are not always correct. In the sequel, we will present a simple model of databases and the storage of information within them. This model will be used to address the issues involved in releasing information without compromising individual privacy. The questions involved will be reduced to simple combinatorial problems concerning the underlying models. While the model presented here is an abstraction of the security situation as it occurs in practice, our results appear to remain valid even when the simplifying assumptions are considerably weakened.

3.3.1 Databases. We will assume that a database is presented as an array of information. Rows of the array represent individuals and columns represent information classifications.

Consider, for example, the following database.

Person	Sex	Field	Location	Age	Salary
Alpha	M	Algebra	East	34	x_1
Beta	M	Topology	West	29	x_2
Gamma	M	Logic	South	54	x_3
Delta	F	Appl. Math	East	45	x_4
Epsilon	M	Geometry	West	48	x_5
Zeta	F	Logic	East	23	x_6
Eta	F	Algebra	South	66	x_7
Theta	M	Topology	South	58	x_8
Iota	F	Geometry	South	33	x_9
Kappa	F	Appl. Math	North	27	x_{10}
Lambda	M	Appl. Math	West	33	x_{11}

Figure 3.7
Sample Data Base

For simplicity, we will consider databases in this form rather than resort to the more complex representations which faithfully model implementation detail, but obscure structure. It is sufficient to note that the results we will quote below extend to the more realistic models in an entirely straightforward way.

Information about individuals represented in a database will be obtained by answers to queries about subpopulations. A subpopulation consists of all individuals having particular values or ranges of values for a given set of characteristics. Subpopulations may also be combined with union and intersection operations in the obvious ways. For example, the following are valid subpopulations of our sample database:

topologists

females from the south or east

males between the ages of 30 and 50.

There are two types of statistical queries about subpopulations. We can either ask for a count of the subpopulation or for statistics regarding some characteristics of the sub-population. For example, valid query responses include the following:

The median salary of topologists

The number of mathematicians from the east

The largest salary of women mathematicians

The average salary of women topologists and male logicians

Queries are assumed to be the legitimate queries necessary for statistical studies of the given population and, as such, it is desirable to provide usable answers in all cases. Conversely, the individuals represented by the information in the database have been insured of the confidentiality of their personal data. We will say that the database has been <u>compromised</u> when this confidentiality is violated. A compromise involves determining information which is regarded as hidden in the database. This hidden information may consist of actual values (e.g., salaries) or bivalent information (e.g., has been arrested). Among the mathematical questions which are natural to ask about such a concept are those which characterize the <u>complexity</u> of compromise under varying query types. The motivation for studying the complexity of compromise is much the same as for studying the cryptocomplexity of enciphering schemes. Our complexity measure for database security is a simple count of the number of distinct queries needed to compromise the database in various settings. In the sequel, we will address different aspects of these problems.

The model presented here does not give an exact representation of "real" database problems. As we mentioned above, we do not work with the relational model of databases but rather with a simpler array model. This assumption has little effect on the translation of results to the relational model, since the constraints on the systems which permit compromise turn out to be similar. We often assume that *any* group of k people can be grouped together into a query, which is often not the case. If, however, compromise is not possible when such capability is permitted, then it surely is not possible when it is denied. Such assumptions are sometimes balanced by simplifying assumptions which *restrict* the user; we may, for example, restrict the *overlap* between pairs of queries. It may in principle be possible to form two queries

$$\mathrm{AVERAGE}\{x_1, x_2, \ldots, x_n\}$$
$$\mathrm{AVERAGE}\{x_1, x_2, \ldots, x_n, x_{n+1}\}$$

from which the value of x_{n+1} is easily found. Finally, the potential penetrator of a database always brings with him a store of *a priori* knowledge which he can use in making inferences about protected information. In practice, the assumptions made here tend to be overly restrictive, therefore a statement of the form

"The database may be compromised in k queries of type..."

tends to be an accurate portrayal of a system weakness, while a statement of the form

"No fewer than k queries will compromise the database"

might provide a lower bound which is not too high in practice.

3.3.2. Compromise by Varying Query Types. We consider methods of database compromise under varying sets of queries. We begin with the situation in which AVERAGE queries are permitted with restrictions on the size of the query set and the overlap between pairs of queries. In this case there are tight bounds on the number of queries needed to compromise. These results are established by linear independence and matrix inversion techniques. Next, we turn to the case of MEDIAN queries and, by using complex forms of binary splitting techniques are able to establish nearly matching bounds. Finally, we consider instances of data distortion in order to determine the ultimate limits on security in the case where we attempt to insure security by greatly restricting the allowable query types. Tight bounds are established by finite geometric arguments.

3.3.3. Linear Queries. Perhaps the most natural setting for the query problem is the one in which the database is simply a set of values $x_1,\ldots,x_n$ so that queries are determined by an index set I. A response to an I-query might be

$$(1/|I|) \sum_{i \in I} x_i$$

If all possible query sets I are allowed, then compromise is trivial. The simplest compromise is achieved by letting $|I|=1$.

Another possibility is to let

$$J = I \cup \{x_k\},\ x_k \notin I.$$

A priori knowledge may also lead to trivial solutions. For instance, if the x_i's represent salaries, a user will know certainly his own salary and possibly the salaries of a few colleagues.

The following complexity measure takes such considerations into account. Define $S(n,k,r,m)$ to be the minimum number of queries needed to compromise a database of n elements where each query involves exactly k individuals, no pair of queries overlaps in more than r positions and at most m values are known in advance.

To set ideas, we show that $S(4,3,2,0) \leq 4$. To see this, consider the queries

$$Q_1 = x_1+x_2+x_3$$
$$Q_2 = x_1+x_2+x_4$$
$$Q_3 = x_1+x_3+x_4$$
$$Q_4 = x_2+x_3+x_4.$$

It follows that

$$x_4 = (-2Q_1+Q_2+Q_3+Q_4)/3.$$

A simple argument shows that this is optimal; i.e., $S(4,3,2,0)=4$.

One property of the S function is that it reveals that small databases can be secured by insisting on large queries. That is, $S(n,k,r,m) = \infty$ if

$$n < k^2/2r+k/2+(m+1)/2-(m+1)^2/2r.$$

To prove this, assume that a set of queries has been proposed from which the database can be compromised. Without loss of generality, let the first query be

$$x_1+x_2+x_k.$$

Now, there must be at least $(k-m-1)/r$ further queries, since some linear combinations of the queries can be reduced to a linear combination of at most $(m+1)$ x_i, k x_i were introduced in the first query and overlap between the queries is limited to r. Furthermore, the second query must introduce at least $k-r$ new elements of the database, the third at least $k-2r$, and so on. Thus, the number of elements in the database must be at least

$$k + \sum_{i=1}^{(k-m-1)/r} (k-ir)$$

from which the result follows.

The query sequence in the proof given above can be restricted further by observing that not only must all m+1 database elements of the first query appear again, but the linear combination of queries used to compromise must involve positive and negative coefficients to insure the proper cancellation. Careful application of this argument can be used to achieve the following lower bound on S:

$$S(n,k,r,m) \geq (2k-(m+1))/r.$$

The bound is obtained as follows. We assume that $x_1, \ldots, x_m$ are known in advance and that x_{m+1} is determined after t queries. We represent the query Q_i by

$$Q_i = \sum_{j=1}^{k} d_{m_{ij}} x_i, \quad i=1,\ldots,t,$$

where,

$$1 \leq m_{i1} \leq \ldots \leq m_{ik} \leq n,$$

for $1 \leq i \leq t$, and

$$|\{m_{i1},\ldots,m_{ik}\} \cap \{m_{j1},\ldots,m_{jk}\}| \leq r,$$

for $i \neq j$.

Because t queries suffice to determine x_{m+1}, we have

$$\sum_{j=1}^{m+1} b_j x_j = \sum_{i=1}^{t} a_i Q_i = \sum_{s=1}^{n} \sum_{i=1}^{t} a_i d_{is} x_s$$

where $b_{m+1} \neq 0$, $a_i \neq 0$, for all $i=1,\ldots,t$, and d_{is} is the characteristic function for x_s in Q_i. We now extend the above observation by noticing that at most m+1 of the terms in

$$\sum_{i=1}^{t} a_i \; d_{is}$$

are nonzero. We assume that each x_t is used in some query, so that the kth such term is zero iff there are i,j such that d_{it} and d_{jt} are nonzero and a_i and a_j have opposite signs.

We now use this fact to prove the lower bound. Choose two queries which appear with opposite signs and assume that they have V values in common. Let K1 and K2 represent the number of known values among the remaining k-V. The (k-(V+K1))/r queries are needed to cancel the remaining terms in the first query, and (k-(V+K2))/r are needed for the second query. Since V $\leq$ r and K1+K2 $\leq$ m+1, we have the claimed bound.

Furthermore, matching the upper and lower bounds is possible in many cases:

1. $S(n,k,1,0) = 2k-1$, if $n \geq k^2-k+1$,

2. $S(n,k,1,1) = 2k-2$, if $n \geq (k-1)^2+2$

3. $S(n,kr+a,r,2a-1) = 2k$, if $n \geq k^2r+2a$.

In each of these cases, a constructive derivation is possible. For (1), the queries used are

$$Q_i = \sum_{j=1}^{k} x_{k(i-1)+j}, \quad i=1,\ldots,k-1,$$

$$Q_{k+i-1} = x_{k^2+k+1} + \sum_{j=1}^{k-1} x_{k(j-1)+i},$$

for $i=1,\ldots,k$, with

$$(1/k)\left(\sum_{i=0}^{k-1} Q_{k+i} - \sum_{i=1}^{k-1} Q_i\right) = x_{k^2-k+1}$$

For (2), the queries used are:

$$Q_i = x_1 + \sum_{j=2}^{k} x_{(k-1)(i-1)+j}, \quad i=1,\ldots,k-1,$$

$$Q_{k+i-1} = x_{(k-1)^2+2} + \sum_{j=1}^{k-1} x_{(k-1)(j-1)+i+1}, \quad 1 \le i \le k-1$$

with

$$(1/k-1)\left(\sum_{i=1}^{k-1} Q_i - Q_{k-1+i}\right) = x_1 - x_{2+(k-1)^2}$$

so that $x_{2+(k-1)^2}$ is determined by x_1.

For (3), the queries used are:

$$Q_i = \sum_{j=1}^{kr} x_{kr(i-1)+j} + \sum_{m=1}^{a} x_{m+rk^2},$$

$$Q_{k+i} = \sum_{j=1}^{kr} \sum_{m=1}^{r} x_{kr(j-m)+(i-m)r+1} + \sum_{p=1}^{a} x_{a+p+rk^2}$$

where in both cases $i=1,\ldots,k$, and

$$\sum_{i=1}^{k} (Q_i - Q_{k+i}) = k \sum_{m=1}^{a} (x_{m+rk^2} - x_{a+m+rk^2}).$$

so that if 2a-1 of the values on the right are known, the last one can be computed.

From (3) we see that if the overlap is fixed and at least one element is known the optimal method is determined for infinitely many query sizes. Gaps between the best known upper and lower bounds occur when the allowed overlap grows at a rate proportional to the query size.

Further extensions of this model have dealt with the case in which queries are not restricted to size k, but can have any size in the ranges [k,n-1] and to the case in which answers are given as weighted sums rather than as averages. In this latter case, if the weights are unknown and if no information is available *a priori*, then the database cannot be compromised by the methods described here. However, if even one value is known, total compromise is possible. This suggests that if a conspirator is able to add information concerning himself to the database, he can compromise a previously secure system.

3.3.3. Median Queries. The median is often used as a statistic in place of the mean because of the many well-known situations in which it provides more realistic information about a sample. Therefore an important query type involves medians, or more generally, any query which returns *exact* values from the database. The model is exactly as in the pevious section, except that the AVERAGE query is replaced by MEDIAN: a typical query specifies an index set I and requests

$\{x_i \mid i \in I\}$.

Many security properties change when medians are allowed. Since the response to a median query is exact (that is, it corresponds to an actual entry), we need only assume that all entries are distinct to devise a trivial compromise algorithm: if two overlap 1 queries return the same result the database has been compromised since there is no ambiguity about the owner of the returned value. Furthermore, changing individual values in a median query need not affect the response to that query. For example, any value larger than the true median can be assumed to be $+\infty$. Thus, we know immediately that if queries are size k or larger, the top $k/2$ and bottom $k/2$ values in the database can never be determined. In addition, some values are apparently easier to determine than others: it is easier to extract the values near the median of the database than to determine extreme values. We will shortly connect these observations with a more general class of combinatorial problems.

Let us first make the assumption that all database elements are distinct. This assumption, while not justified in practice, is not unreasonable when one is working with a small random sample of a large database. After stating some results in this restricted model we will show how to remove some of the restrictions.

There are some simple methods of compromise in the best case. Two median queries with overlap 1 and which return the same value compromise the database, but if entries are not unique and unlimited overlap is not allowed, there is still a possibility for compromise by the three queries:

Q_1: $\mathrm{MEDIAN}\{x_1, x_2, \ldots, x_k\} = a$

Q_2: $\mathrm{MEDIAN}\{x_{k+1}, x_2, \ldots, x_k\} = b$

Q_3: $\mathrm{MEDIAN}\{x_{k+2}, x_2, \ldots, x_k\} = c$.

We may assume that $c>a>b$. Then Q_1 and Q_2 imply that $x_{k+1} < a \leq x_1$ and Q_1 and Q_3 imply that $x_{k+2} > a \geq x_1$. Hence, $a \geq x_1 \geq a$, so that $x_1 = a$. Notice that this is the best case since if $a=b$ or $a=c$, then x_1 cannot be determined in this way.

In the worst case, it is possible to show that $O(\log k)$ median queries of length k are required to compromise a database of n elements. The proof involves information theoretic arguments. The best known upper bound is $O(\log^2 k)$ queries. The algorithm involves applying binary search techniques to derive balanced sets from which a median element can be found to match a known median value. For simplicity we prove here the weaker $3k/2 + 7/2$ bound for median queries of size k in a database of $n \geq k+2$ elements. First, compute the medians of all k-sized subsets of $\{x_1, \ldots, x_{k+1}\}$. These queries lead to exactly two values, say h and m. These medians define two sets H and L with $|H| = |L| = (k+1)/2$, where:

$H=\{x_i | x_i \leq h\}$

$L=\{x_i | x_i \geq m\}$

The sets are determined by noting that $x_i \in H$ iff the median of $\{x_1, \ldots, x_{k+1}\} - \{x_i\}$ is m. Next, form H' from H by deleting two elements of H and taking the median of $H' \cup L \cup \{x_{k+2}\}$. If the value is m, then $x_{k+2} < m$. If the value is greater than m, then

$x_{k+2} > m$. The value cannot be less than m. Having ranked x_{k+2} with respect to m, we can now form L' by deleting an element of L and finding the medians of length k in

$$H \cup \{x_{k+2}\} \cup L' - \{x_i\}$$

for each $x_i \in H \cup \{x_{k+2}\}$. One value will occur as the answer all but one time. This value is the value of the element missing from that query.

Reducing the complexity of this algorithm to $O(\log^2 k)$ involves a more careful construction of the sets H and L, forming a balanced set and ranking of another element as above.

Finally, we consider the case where nonuniqueness is not assumed and the overlap among queries is limited. In this case, we define M(n,k,r) to be the minimum number of k-median queries necessary to compromise a database of n elements with overlap between queries limited to r. In this case, we have:

$$M(n,k,1) \geq 3/4(k+1), \text{ for } k \geq 3,$$

and

$$M(m,k,1) \leq 3k-5, \text{ for } n \geq k^2 - 2k+4.$$

The proof of the lower bound is similar to those given previously. We argue that at least (k+1)/2 of the elements involved in any query set must occur in a second query or their values are never needed. The overlap restriction says that at least (k+1)/2 new queries are needed to cover elements introduced in the first query. These queries are again limited in overlap and the ith such query may contain no more than i-1 elements from previous queries. So, the new queries must introduce a total of

$$\sum_{i=1}^{(k-1)/2} i = (k^2 -1)/8$$

new elements which need to appear in other queries. And the overlap restriction limits the number of such elements to be added to a new query to (k-1)/2, requiring (k+1)/4 new queries. Thus we have a total of

$$1+(k-1)/2 + (k+1)/4 = 3/4(k+1)$$

queries.

The upper bound involves manipulating the overlap restriction. We generalize the argument which gave the upper bound of 3 above in the unlimited general case. We assume the database elements are addressed as x_0, x_{ij}, for $1 \leq i,j \leq t$, and y_1 and y_2, where t=k-1 and q=k-2. The queries are then

$$Q_i = \text{MEDIAN}\{x_0, x_{i1} \ldots, x_{it}\} \ , \ 1 \leq i \leq t$$

$$R_i = \text{MEDIAN}\{y_1, x_{1i}, \ldots, x_{ti}\} \ , \ 1 \leq i \leq q$$

$$S_i = \text{MEDIAN}\{y_2, x_{1,i+1}, x_{2,i+2}, \ \ldots, x_{ti}\}, \ 1 \leq i \leq q,$$

with results to all Q queries being a, results to all R queries being b and results to all S queries being c, where $c<a<b$. It is easy to show now that $x_0=a$ by showing that $x_0 \leq 0$ and $x_0 \geq 0$.

3.3.4. Lying Data Bases. Median queries lead to another generalization. Suppose that a database consists of n elements and that all queries are of size k, with overlap restricted to 1. Suppose further that all queries are submitted to an "oracle" who answers by choosing any element of the query set for a response. Furthermore, this oracle can answer in any manner whatsoever to protect the database. We may even assume that the oracle receives all queries before answering a single query. Although such an oracle intuitively has a great advantage in keeping the database secure, we will see that compromise is not very difficult. In fact, if $R(n,k)$ is the complexity of compromise for this model, then:

$$R(n,k) \leq 4k^2.$$

Compromise involves essentially constructing a finite projective plane. That is, all we need to show is the existence of $M = m(k)$ sets Q_i, for $1 \leq i \leq M$, such that $|Q_i \cap Q_j| \leq 1$ and each is a query on the elements $\{x_i | i \leq M-1\}$. Clearly, then the uniqueness of the elements guarantees compromise by the pigeon-hole principle. If k is a prime power, there is a projective plane of order k which is a system of k^2+k+1 sets chosen from k+1 of the k^2+k+1 database elements, each pair of sets having exactly one element in common. Deleting one query from the system and removing all of its elements leaves k^2+k queries of size k on a database of k^2. If k is not a power of a prime, there is surely a prime p, $k \leq p \leq 2k$, so that the construction for this prime p can be reduced to a construction for the desired k, while only adding a factor of 4 to the complexity of the entire construction.

3.3.5. Combinatorial Inference. The database problem is a special instance of a more basic type of problem: Given a finite set X, to infer properties of elements of X on the basis of "queries" regarding subsets of X. Besides the database security problem, the following combinatorial problems can also be formulated in this way.

Function Identification: determine the structure of a computer program by observing selected parameters of its operation.

Group Testing: a group of blood samples is to be processed rapidly to identify diseased persons. This is accomplished by mixing samples to determine whether or not any members of a set of subjects is infected and identified for further samples. The disease is such that sets of carriers of different strains can negate each others' effects in certain situations.

Balance Problems: given a number of objects of some standard weight β and two defective objects which weigh slightly more and less than β but are otherwise undistinguished, isolate the defective objects by weighings on a k-arm balance.

Multidimensional Search: given an algebraically defined set X, does X determine a point y (for example X may be a set of linear varieties, one of which may contain y).

<u>Coin Weighing</u>: by choosing X from {1,...,k} and allowing queries of the form "what is $|S_i \cap X|$?", $S_i \subseteq \{1,...,k\}$, to determine X.

All of the aspects of database problems are interesting in this more general setting. We may vary the choice of primitives by allowing suitably restricted queries. The problem may be to insure that k queries always solve the problem or to determine minimal values of k needed to solve the problem. General audit problems (proving upper and lower bounds on the complexity of determining whether or not a given sequence of queries allows the appropriate inference) and enumeration problems (determine the number of "unsafe problems" for a given number of queries) are also important.

3.4 Bibliographic Notes. The first published work on protection models is [37]. Their research was heavily influenced by the PhD Dissertation of Jones [44]. A number of diverse and interesting models appears in the work of Jones, Lipton and Snyder [4, 8, 9, 54, 84]. Much of section 3.3 is drawn from [25].

Parker's case studies gives a good overview of the security threats modern computers face [64]. Other surveys that are helpful are [24,15,72]. The papers of [19] also lend a current perspective.

The Bell and LaPadula model is presented in [2], while the application of this model to kernelized systems is presented in [62]. The proceedings of the 1979 summer school on security [26] describe the current state of technology for the various kernel projects. The specification [33] is a complete specification of KSOS-11. A critical evaluation of verification technology is offered in [18].

The concept of confinement is defined in [50], where the examples of leakage channels are also to be found.

Other inference mechanisms which illustrate the ease with which information may be extracted from software systems have been proposed in the context of database systems [14,16,17,29,69].