A Linear Time Algorithm for Deciding Security

(Extended Abstract)

A K. Jones[†]

R. J. Lipton[††]

L. Snyder[††]

[†] Carnegie-Mellon University

[††] Yale University

1. Introduction

The theoretical analysis of systems for protecting the security of information should be of interest to the practitioner as well as the theoretician. The practitioner must convince users of the integrity of their programs and files, i.e. he must convince them that the operating system and its mechanisms will correctly protect these programs and files. Vague or informal arguments are unacceptable since they are often wrong. Indeed the folklore is replete with stories of "secure" systems being compromised in a matter of hours.

A key reason for the abundance of these incidents is that even a small set of apparently simple protection primitives can often lead to complex systems that can be exploited, and therefore compromised, by some user. But it is precisely this fact, simple primitives with complex behavior, that lures the theoretician. Our purpose here is to present a concrete example of a protection system and **then** completely analyze its behavior.

Our motivation for doing this analysis is twofold. The protection system that we will study is not one we made up, rather it is one that has been defined, discussed, and studied by those in operating systems (Denning and Graham [2], Cohen [1], Jones [4]). This point is most important, for the space of possible protection systems is exceedingly rich and it is trivial to think up arbitrary systems to study. We are however not interested in arbitrary systems, but in systems that are of practical interest.

The above motivation is necessary but not sufficient for us to establish that these questions should interest the theoretician. Our second

reason for studying these problems is that in a natural way they can be viewed as "generalizations of transitive closure." Roughly these protection questions can be modelled as:

Given: Some directed labelled graph G and a set of rewritting rules $\mathcal{R}$.

Determine: Whether or not there is a sequence of graphs $G_1, G_2, \ldots, G_n$ such that $G = G_1$, G_n has property X, and G_{i+1} follows from G_i by some rule in $\mathcal{R}$.

Here property X would of course encode that there was a protection violation in G_n. Our goal would then be to show that it is impossible to reach such a G_n, i.e. that a protection violation is impossible.

A common type of property X is:

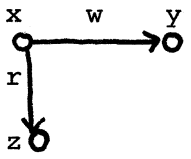
there is an edge with label α from vertex p to q.

For these properties our protection questions do indeed look very much like transitive closure questions. Indeed if the rules $\mathcal{R}$ only allowed the addition of edges, then these problems would be trivial. They do not. The rules of interest to those in protection, and the rules we will study in particular, allow new vertices to be added. This simple change of allowing graphs to "grow new vertices" makes these problems challenging. Indeed the particular one we will study is no longer even obviously decidable.

Let us now make the above concrete by sketching the particular protection system we will study. We consider directed graphs whose arcs are labelled with an r or a w or a c. While we will manipulate these graphs as formal objects it is helpful to keep in mind the following informal semantics:

1. A vertex corresponds to a "user".
2. r = "read", w = "write", c = "call".
3. If there is a directed arc from x to y with label r (respectively w, c), then x can read y (respectively write, call).

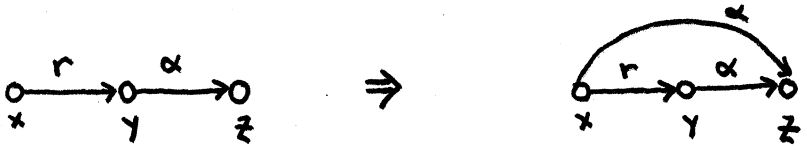
For example, in the graph



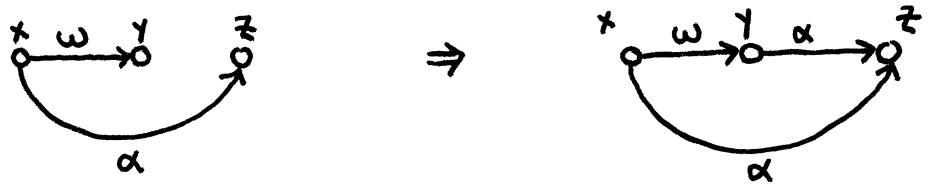
x can write y, x can read z, but y cannot write z since this edge is missing.

This protection **model, called the take and grant system, is now completed** by presenting four rewriting rules.

1. *Take:* Let x, y, and z be three distinct vertices and let **there** be an arc from x to y with label r and an arc from y to z with some label α (α = r or w or c). Then the take rule allows one to add the arc from x to z with label α. Intuitively x **takes** the ability to do α to z from y. We will represent this rule by



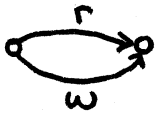
2. *Grant:* In our representation this rule is



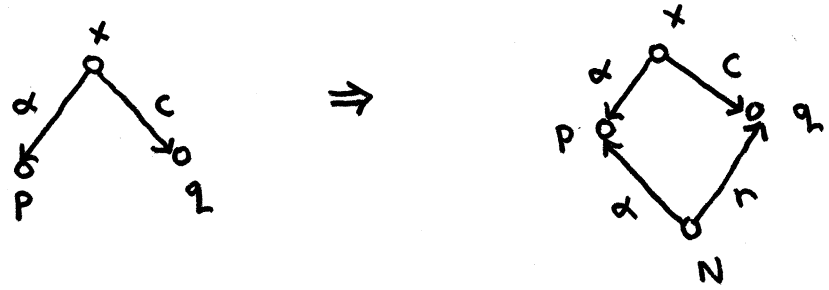
Intuitively x **grants** y the ability to do α to z.

3. *Create:* In our representation this rule is

where N is some new vertex. **Intuitively** x creates a new user that he can both read and write. (Notation: 

will be used instead of 

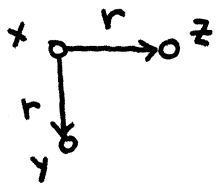
4. *Call:* In our representation this rule is



where N is again a new vertex. Intuitively x is calling a program q and passing the parameters p. N is then a "process" that is created to handle this call: N can read q the program and can do α to the parameters.

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

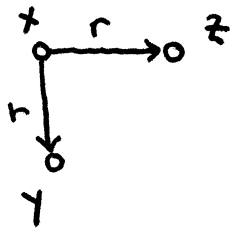
Now that we have seen the rules, let us look at their behavior. We will start with a simple question: in the graph

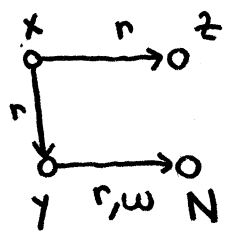


is it possible of y to r z? The answer is obviously no since there is no r arc from y to z. But what we mean is:

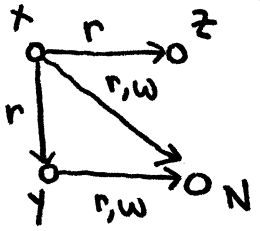
> is there a sequence of rule applications that lead to a graph with an r arc from y to z?

More generally, say p can α q if there is such a series of rules that leads to an α arc from p to q. Then our question is : is it true that y can r z? Clearly, without create, the answer is no since neither take nor grant nor call can **even** apply. The following sequence of applications of the rules shows that the answer is yes:

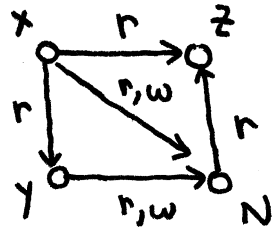




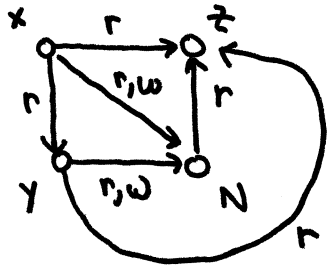
(ii) x takes (twice)



(iii) x grants



(iv) y takes



This example demonstrates the type of clean graph type problems we will be studying. Our main theorem is stated in the next section. This theorem presents a complete answer to the question: is it true that p can x q? Indeed this theorem leads easily to a linear time algorithm for **answering** the question.

A final word about how this theorem contributes to our understanding of protection. Each user of a protection system needs to know:

what information of mine can be accessed by others;

what information of others can be accessed by me?

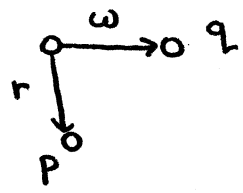
The question is vauge in general, but here it is rendered in the simple question: is it true that p can α q?

The types of protection models studied here have received considerable attention recently. Our approach is related closely to the work of Harrison, Ruzzo, and Ullman [3]. They show that what can be called the "uniform safety problem" is undecidable. Interpreted as a graph model, their result says that give an arbitrary set of rules (similar in spirit to **take**, grant, etc.) and an initial graph, it is undecidable whether or not there will ever be an arc from p to q with label α. This is a uniform problem in the sense that the rules are arbitrary. Even when the rules have to satisfy certain additional constraints the results of [3] and the results of Lipton and Snyder [5] show that protection is impractically complex.

Our view here is that since the uniform protection problem is so difficult and since operating systems require usually only one fixed set of protection rules, then the nonuniform problem should be studied. As stated before we choose the take and grant system by studying the protection literature.

2. Basic Results

In this section we state and then sketch our main result. All graphs considered are directed graphs with arc labels r, w, or c. In such a graph say p and q are connected provided there is a path from vertex p to vertex q independent of the senses of the arcs and their labels. Thus in



p and q are connected.

Theorem 1: Let G be any graph. The the predicate "p can α q" is true if and only if

(1) p and q are connected, and

(2) for some vertex x, $x \xrightarrow{\beta} q$ where β statisfies: if $\alpha = w$ then $\beta = w$; if $\alpha = r$ then $\beta \in \{r,c\}$; if $\alpha = c$ then $\beta = c$.

It should be clear that this theorem leads to a linear time algorithm for "p can α q". The key therefore is the validity of this theorem.

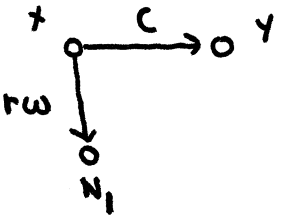
First, we observe that the conditions (1) and (2) of theorem 1 are necessary for " p can α q". An induction establishes this. Therefore the crux is the demonstration that they are sufficient. We will now sketch this.

Lemma 2: Let $p = x_1,\ldots,x_n = q$ be a rwc path, i.e. either $x_i \xrightarrow{\alpha} x_{i+1}$ or

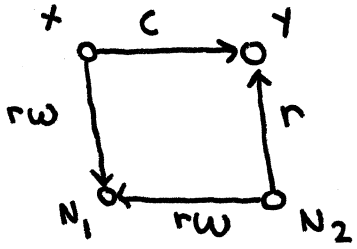
$x_i \xleftarrow{\alpha} x_{i+1}$ for each $1 \le i \le n-1$ and some $\alpha \in \{r,w,c\}$. Then in this graph p can α q if $x_{n-1} \xrightarrow{\alpha} q$.

This lemma is proved by induction on n and careful examination of the rules. A **basic** trick is we can replace $x \xrightarrow{c} y$ by $x \xrightarrow{r} y$ by proceeding as follows:

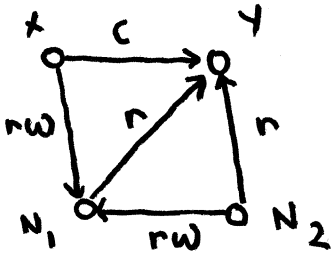
(i) create



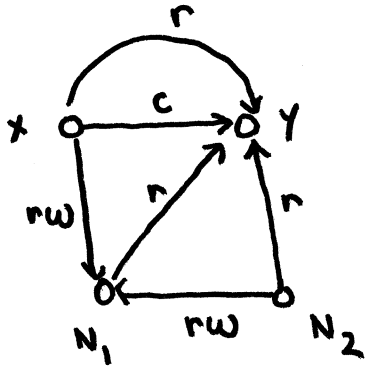
(ii) call



(iii) grant



(iv) take



Lemma 3: Let $P = x_1, \ldots, x_n = q, x_{n+1}$ be a rwc path with $q = x_n \xleftarrow{\alpha} x_{n+1}$. Then p can α q.

The proof of this lemma breaks down into three cases depending on what α is. We also use lemma 2 critically in its proof. It is interesting to note that during this proof it is necessary to have created vertices create other vertices. Once this lemma is proved we have essentially established theorem 1.

Although we have a necessary and sufficient condition for "p can α q" we have not solved the whole security problem. This is because it is necessary to dichotomize the elements of the system (i.e. the vertices of the graph) into two classes -- subjects and objects. This separation is needed to capture the distinction between programs (subjects) and the more passive objects such as files. The consequence of dichotomizing the vertices is that the rules take, grant, call and create are operations that can only be initiated by subjects, i.e. in the graphical definitions of these operations given above, the x vertex must be a **subject** in each case. This requirement adds a new dimension to the problem.

We have not yet fully analyzed this case, but we have resolved the question for a wide class of systems.

Theorem 4: There is a polynomial **time algorithm to decide if the predicate** "p can α q" is true for any graph G with subject-object **dichotomization** provided G contains no directed object cycle.

The problem is still open for general subject-object graphs.

3. Conclusions and Extensions

We have presented a clean and simple model of a special security system from the literature. We have shown a necessary and sufficient condition for the predicate "p can x q" which is linear time checkable. This predicate is basic in answering security questions.

In addition, we have only scratched the surface of what promises to be a rich area of research. First, for the present model we must solve the "p can α q" predicate problem for general graphs for the subject-object case. We also conjecture that the polynomial algorithm of theorem 4 can be made linear or nearly so. There is also the problem of representing other security primitives (from the literature) in this model as well as analyzing other predicates.

Finally, and maybe most importantly, there is the question of finding graph rules that implement more complex security policies. The policies implemented by take-grant systems are simple, although they are by no means trivial. This is the first precise statement of these policies and thus we are only now aware of how undiscriminate it is. But we would probably prefer more discriminating security policies than these and so now the synthetic phase of research could be initiated. The goal would be complex protection policies with efficient verification methods.

References

1. E. Cohen.
Ph.D. Thesis (in progress), Carnegie-Mellon University, 1976.

2. P. J. Denning and G. S. Graham.
Protection principles and practice.
AFIPS Conference Proceedings 40:417-429, 1972.

3. M. A. Harrison, W. L. Ruzzo, and J. D. Ullman.
On protection in operating systems.
Proceedings of the 5th annual SIGOPS Conference, 1975.

4. A. K. Jones.
Protection in programmed systems.
Ph.D. Thesis, Carnegie-Mellon University, 1973.

5. R. J. Lipton and L. Snyder.
Synchronization and security.
In preparation, 1976.