Correctness In Access Control*

Gerald J. Popek, Harvard University
Currently, University of California, Los Angeles

ABSTRACT

A number of approaches have been developed to modularize parts of multiuser computer systems so that access to each part can be controlled. The devices of rings and capabilities are two examples. However, today's systems are notably incomplete and subject to defeat by determined and clever users. A point of view is presented here which allows proving that a logical design of an access control system is correct relative to a designer-specified set of criteria. Implementation questions are also discussed.

KEY WORDS AND PHRASES: access control, correctness, time-sharing, security, data base, logical design.

THE CONTEXT

One part of the computer community is currently concerned with security design problems of the following kind. Given that components of a computer system such as terminals, communications lines, files, processes, and other resources need to be safeguarded, these parts can at least partly be protected by designing proper modularity and access control mechanisms into a system. How is such a system best modularized and how are the controls that mediate access among modules best designed and implemented? This question affects both hardware and software architecture, and is receiving considerable research attention. See for example [5], [7], and [9].

Inventive solutions have resulted from this research. However, as time-sharing systems, utilities, and networks have displayed greater and greater complexity, concern has arisen over the reliability of the implementation of the protection system design, for whatever design is finally developed. What guarantee exists that the system actually provides the controlled protection that it claims? What guarantee exists that it is not possible for some clever user to circumvent the controls, gaining access to information, operations, or other resources which the design was intended to prohibit.

Currently, no protection system implementation of any major multiuser computer system is known to have withstood serious attempts at circumvention by determined and skilled users [10].

Organizations such as the military have already begun to demand that the vendors of multiuser computers provide guarantees that their protection mechanisms actually do what they claim; verification, or certification of that part of the system is wanted.

It has been suggested by the Military Security Panel [11] that the first step in this problem is the development of a general model of access control systems that:

(a) is applicable to a sizable group of useful protection systems; and
(b) can be proven correct.

The second step would then be to guarantee a faithful implementation of an access control system specified in terms of that model. Such a model is presented below in order to suggest that its development, while useful, is straightforward, and that on the level of abstraction of the model, correctness is easily provided.

Here, access control is meant to refer simply to the problem of controlling access to specified units of information or other resources. No specific attempt is made to include problems of inference or statistical access.

The inference problem concerns the desire to prevent a user or users from gaining access to a number of pieces of information, each innocuous, which together constitute or can be used to deduce information that it is desired not be available. The statistical access problem is the reverse. It refers to the desire to allow access to aggregations or analyses of information with a guarantee that it is not possible to deduce a specific piece of information used in the aggregation. Attention is essentially restricted here instead to the desire for guaranteeing that solutions to the simpler, access control problem, are properly implemented.

The following approach, in the spirit of program schemata, as a guide to the logical design of an access control system, applies to a wide class of systems, including many of those in use today.

*This research was partially supported by the Advanced Research Projects Agency of the Department of Defense under Contract No. DAHC-15-69-C-0285.

BRIEF DESCRIPTION OF MODEL

The model is described in set theoretic language, and has six major components. First is the set O of security objects: the elements of the model, reflecting those physical or logical parts of a computer system that need to be controlled, protected, or whose status needs to be guaranteed. The objects are partitioned into disjoint classes, each containing objects of similar characteristics. An incomplete list of examples includes terminals, communication lines, processes and files.

Second, a set A of access types is presented. Each access type is a program which effects a particular variety of access, such as read, write, or execute. An attempted access operation is then completely specified by an access type and some meaningful collection of objects, i.e., a particular process being directed from a given terminal attempting to reference a specified page in memory.

Third, a collection of descriptive data D[k], from the set of all possible descriptive data collections D is required. D[k] specifies the information that forms the basis by which security decisions will be made. The subscript k indicates a time dependency.

Fourth, an evaluation program, $\mathscr{E}$ decides, for any meaningful grouping of objects, what operations are to be allowed.

Fifth, an update program $\mathscr{U}$ is characterized separately. This program is the means by which the descriptive data are changed. Operationally, this is the manner by which access decisions may be altered.

In many real implementations, the distinction between the evaluation program and update program may not be clear-cut, since the descriptive data are likely to be stored and protected like any other security object. Both programs are treated here so that their similar nature is apparent. Nevertheless, the distinction will be useful since implementations of the two programs may differ. $\mathscr{E}$, while likely to be software implemented, calls upon access programs to do its actual work, and these may be at least partly if not wholly built in hardware. $\mathscr{U}$ on the other hand in many cases will be almost exclusively software and actually changes the formatted descriptive data.

Last, external correctness criteria are required. These are a set of rules, or standards T, by which the system is to be adjudged correct. These standards must be external to the system description up to this point in order to be meaningful.

A security system S is then specified by the six-tuple: $S = (O, A, D, \mathscr{E}, \mathscr{U}, T)$.

THE COMPONENTS OF THE MODEL

Security Objects

The first component of the model, the security objects, is a finite set O:

$$O = \{o[1], o[2], \ldots, o[z]\}.$$

These are the only objects to which access will be controlled by the model, and by a resulting implementation. They include, for example, both the subjects and objects of the Lampson model [4].

Access Types

The second component of the model is a set of access types:

$$A = \{a[o], a[1], a[2], \ldots, a[w]\}.$$

Each a[i] is a program whose effect will be to provide a particular variety of access, read, write, or execute for example. The list of arguments for each a[i] must be finite and contain names of security objects. In addition, a[o] is designated as the null access program. This program will be invoked when access is to be denied. It can keep audit trails, set up warnings to administrators, etc.

Descriptive Data

The third component, the descriptive data, is merely a set of tuples:

$$D[k] = \{d[k,1], d[k,2], \ldots d[k,v]\}.$$

with some finite upper bound set on v.

We depart somewhat from our strict set theoretic notation by speaking of the structure of a tuple. Each tuple is only assumed to have a bounded number of entries, the first of which acts as a "data descriptor" to distinguish among tuples of different formats and content.

For example, one type of tuple might be an encoding of a matrix entry in Lampson's model [4]; the entry expressing an access relation between two security objects. Another might express a property: user x belongs to project y, or has clearance z. A property may also be valid only for several users jointly. Such circumstances do not fit naturally into a matrix representation of the descriptive data, so tuples are preferred here.

Explicit use of the structure of the descriptive data will not be made in the following discussion of correctness, although it is necessary in the more detailed proof. The finiteness of both the length and number of tuples will be useful here, however.

Let X^* be the set of all possible tuples, and $D = P(X^*)$ the power set of X^*. Then D[k] is some member of $P(X^*)$.

Evaluation Program

The third portion of the model is an evaluation program $\mathscr{E}$ which uses descriptive data to make decisions concerning access. For any evaluation program, the list of arguments is composed of some

fixed number of objects from each partition of the security objects O, and an access type; the name of an element in A. For convenience, those objects are denoted by Θ.

The task of the evaluation program is to decide whether or not the specified objects may be associated in the manner expressed by the access type and to indicate an appropriate action. That indication is done by selecting the appropriate access program and specifying its proper arguments.

The evaluation program $\mathscr{E}$ takes a list of object names, a particular descriptive data configuration, and the name of an access type (names of elements are underlined); and invokes the allowed access program, supplying it with the appropriate argument list.

$\mathscr{E}$ is composed from an access rule E. E is a fairly arbitrary program that is assumed only to (1) terminate, returning true or false, and (2) be read only.

The intent is that E describe conditions to be fulfilled in order to allow access. It may be an arbitrary function of its arguments, although often such programs are fairly simple. In any case E can be made an effective procedure, since all arguments are from finite sets.

Than the program $\mathscr{E}$ may be written as follows:

```
ℰ: proc (Θ, D[k], a[j]);
lock;
if E(Θ, D[k], a[j])
then begin unlock; call a[j](Θ) end
else begin unlock; call a[o](Θ) end;
end;
```

The functions lock and unlock are understood to act on a single semaphore, as Dijkstra's operators P(x), V(x). It is necessary to coordinate the operation of $\mathscr{E}$ and $\mathscr{U}$ so that $\mathscr{E}$ is not reading D[k] while $\mathscr{U}$ is updating D[k]. Otherwise, it would not be possible to prove that $\mathscr{E}$ and $\mathscr{U}$ perform in all cases as claimed.

Update Program

The update program is the means by which descriptive data are changed. Hence it is the manner by which decisions that the evaluate program makes can be affected. Let Θ' denote the set of arguments for the update program which are security objects, D[y] is the current descriptive data, and D[z] is the data to which it is desired to change. $\mathscr{U}$ yields either the original data, prohibiting the change, or the new data, having allowed the change.

The update program, too, is composed from some effective procedure U, similar in purpose to E, and so the update program $\mathscr{U}$ may be written as:

```
𝒰: proc (Θ', D[y], D[z]) returns element of D;
lock;
if U(Θ', D[y], D[z])
then begin unlock; return D[z] end
else begin unlock; return D[y] end
end;
```

The arguments for U are the same as for the procedure itself.

THE CORRECTNESS CRITERIA

The security objectives of the access control system are the qualities that are necessary to guarantee. For a certain well-defined class of criteria, there is a straightforward method of taking a logical description of a security system and altering that model to provide a derived system model in which the given correctness criteria hold.

The correctness criteria are expressed as a set T of predicates:

$$T = \{t[1], t[2], \ldots, t[q]\}.$$

These are the predicates that must be proven true for the system.

In this model, predicates may be expressed in one of two forms, and so T is partitioned into two subsets T1 and T2 corresponding to the two alternatives.

If t[i] is in T1 then it may be any predicate expressible in the following functional form:

$$t[i] : \Theta \times D \times A \rightarrow \{\text{true}, \text{false}\}.$$

The interpretation of predicates in T1 is that the object list from Θ may be associated with access type a[j] in A and a given D[k] in D only if t[i] is true.

If t[i] is in T2, then it may be any predicate expressible in the following functional form:

$$t[i] : \Theta' \times D \times D \rightarrow \{\text{true}, \text{false}\}.$$

The interpretation is that the descriptive data represented by the second argument, say D[j], may be changed by the objects expressed by Θ' to that represented by the third argument, say D[k], only if t[i] is true.

Let

$$\mathscr{T}1 = \text{And}(t[i]) \text{ for all } t[i] \text{ in } T1 \text{ and}$$

let

$\mathcal{T}2$ = And(t[j]) for all t[j] in T2.

$\mathcal{T}1$ and $\mathcal{T}2$ take the same arguments as the t[i] and t[j], respectively.

To demonstrate that a system is correct, it is necessary to guarantee the truth of $\mathcal{T}1$ and $\mathcal{T}2$. Below, a simple way is shown to take any security system S and derive from it a system S' for which the given $\mathcal{T}1$ and $\mathcal{T}2$ are true.

DERIVATION OF CORRECT SYSTEM

System Specification

As described, a security system S is a tuple:

$$S = (O, A, D[o], \mathcal{E}, \mathcal{U}, T)$$

O is the object set, A is the set of access types, D[o] is taken as the set of tuples which comprise the initial descriptive data, $\mathcal{E}$ is the evaluation program, $\mathcal{U}$ is the update program, and T is the set of predicates to be guaranteed.

For a particular system S, the entries A, $\mathcal{E}$, $\mathcal{U}$, and T are fixed. The descriptive data D[k] may be varied by use of $\mathcal{U}$. Then the state of a security system S can be completely expressed by its descriptive data D[k], for some k. The update program is the means by which a system S may change states and the compound predicate $\mathcal{T}2$ expresses the constraints on allowed state changes. The evaluation program $\mathcal{E}$ "interprets" a particular state, and $\mathcal{T}1$ expresses the constraints on $\mathcal{E}$.

Given a security system S = (O, A, D[o], $\mathcal{E}$, $\mathcal{U}$, T), system S' = (O, A, D[o], $\mathcal{E}'$, $\mathcal{U}'$, T) is produced by the following inclusion step.

$\mathcal{E}'$ is derived from $\mathcal{E}$ by the following change. Replace

"E(...)"

by

"E(...) and $\mathcal{T}1$ (Θ, D[k], a[j])".

$\mathcal{U}'$ is derived from $\mathcal{U}$ by the following change: Replace

"U(...)"

by

"U(...) and $\mathcal{T}2$ (Θ', D[y], D[z])".

Correctness Proof

First it is helpful to define a few terms.

A state D[n] of a system

$$S = (O, A, D[o], \mathcal{E}, \mathcal{U}, T)$$

is valid if and only if D[n] can be obtained from D[o] by a finite number of applications of $\mathcal{U}$ and, for each such transition from state D[k] to D[k+1],

$$T2(\Theta', D[k], D[k+1]) = \text{true}$$

for some Θ'.

Second, a state D[k] is accurately interpreted if and only if for any Θ and any j:

$\mathcal{E}(\Theta, D[k], a[j])$ invokes a[o](Θ)

whenever

$\mathcal{T}1(\Theta, D[k], a[j]) = \text{false}$

(where a[o] is the null access type).

Then to say that a system S is correct is meant the following:

(1) Every state obtainable from D[o] is valid, and
(2) every valid state is accurately interpreted.

We now state the following (system correctness) theorem;

Given a security system

S = (O, A, D[o], $\mathcal{E}$, $\mathcal{U}$, T) with T partitioned

into T1 and T2;

and S' = (O, A, D[o], $\mathcal{E}'$, $\mathcal{U}'$, T) derived from S

by the inclusion step

then S' is correct.

Proof Sketch

An easy way to prove the theorem is by contradiction. Suppose the theorem false. Then, by definition of correct, S' reaches an invalid state, or a valid state is inaccurately interpreted.

Case 1: Assume an invalid state. Label that invalid state D[k]. Then there must exist a sequence of states D[o], D[1], D[2], ..., D[k] such that $\mathcal{U}'$([i], D[i], D[i+1]) = D[i+1] for all i<k, since $\mathcal{U}'$makes the transition from state to state.

Now D[o] is valid by definition. D[k] is invalid by assumption. Then there must exist a non-negative integer j, less than k, such that D[j] is valid and D[j+1]) is invalid. Hence, by definition of valid, $\mathcal{T}2(\Theta, D[j], D[j+1])$ is false. But $\mathcal{U}'(\Theta, D[j], D[j+1]) = D[j+1]$. By inspection of $\mathcal{U}'$, these two conditions cannot hold, and hence a contradiction is reached.

Case 2: Assume an inaccurately interpreted valid state. Call that valid state D[k]. Then by definition of an accurate interpretation, for some Θ[i] and a[j], the following is true.

$$\mathscr{T}1(\Theta[i], D[k], a[j]) = \underline{\text{false}} \text{ and}$$

$$\mathscr{E}'(\Theta[i], D[k], a[j]) \text{ does not invoke } a[o](\Theta)$$

By inspection of $\mathscr{E}'$, this is a contradiction. Hence every valid state is accurately interpreted.

Both cases are impossible. Hence the theorem cannot be false.

qed

This proof is of course nearly tautologic in nature.

DISCUSSION OF APPLICABILITY

The utility of this model depends on several criteria not yet addressed. First, the access control model was purposely constructed in an extremely general way, so that many access control systems can be placed into its broad framework. As an example, a ring structure may be modeled by tuples in D which contain the ring brackets of a segment or program. An active process has its current ring changed by an agent, or gatekeeper, which uses the update program $\mathscr{U}$. A discussion of current access control systems appears, for example, in [4].

The second assumption that affects applicability primarily concerns the correctness predicates: the members of the set T. For the given characterizations, effective procedures exist for the update and evaluation programs, the predicates from which they are composed, and the predicates which make up the correctness criteria. This fact is a result of the finiteness of all the sets involved in the model. It is further argued that the descriptive data can be structured to make those procedures relatively efficient. That efficient procedures exist for all the predicates in the predicate set T makes the inclusion step meaningful. The assumption is that desired correctness criteria can be placed in the specified form. However, no meaningful correctness criteria have been suggested to the author that cannot be so handled.

Those criteria suggested have been similar to the following: Users of class so and so may not perform a certain set of operations on information marked in such and such a way. They are generally negative requirements, in the sense that certain conditions or operations are to be guaranteed forbidden. A military requirement might be that users with clearance level x cannot access information with classification y > x. In health records, it might be desirable to guarantee that members of the accounting department be unable to determine the reasons for admittance of a patient, the classic case being venereal disease. Clearly, any of these requirements can be included in a number of ways.

It may often be possible, of course, to more efficiently enforce certain theorems through the logical structure of the system rather than by what amounts here to run time checks.

Third, the model must include all mechanisms that any part might require for its proper operation. Additional apparatus needed must be local to a single module and not require any further interconnections between elements of the model if the discussion of correctness is to be meaningful and a properly structured system constructed. Each module may then, essentially in isolation, be the subject of a correctness investigation itself. To fulfill this criterion a locking mechanism is included at the top level.

One of the conclusions to be drawn from this security model is that the task of providing a correct model is simple, even for a model that can describe most contemporary systems. Hence the major problem is the implementation.

IMPLEMENTATION IMPLICATIONS

In order to construct a computer access control system in which there exists a high degree of confidence that the logical guarantees are properly implemented, the following strategy is proposed. Isolate that part of the operating system responsible for security and place it in a protected part of the system, in a manner analogous to the manner in which current supervisors are segregated from user programs through the mechanism of separate hardware states.

Call this isolated portion the kernel. It will be necessary to demonstrate that this segregation is performed in such a manner that guarantees the kernel's integrity and also guarantees that the kernel is always invoked to arbitrate attempted references. These tasks are eased by the fact that the kernel can aid in protecting itself. For example, descriptive data can be grouped as security data.

Then a great deal of attention can be paid to providing a correct implementation of the kernel. Later, as changes are made to other parts of the operating system, it is not necessary to revalidate the kernel. This point is more than a motherhood call for modularity. It argues that the security kernal should be isolated at the center of a system; it is to arbitrate and control all activity. In a hardware ring environment for example, the kernel alone should occupy the innermost ring. (See [5] for a discussion of rings.)

One of the values of this security model is that it can help specify what is necessary to include in a kernel. It is intended that the kernel of a computer system include everything that this model contains, and nothing else. Hence the model defines the boundaries of the kernel, and the ability to use the kernel to protect parts of itself allows one to provide carefully controlled access to the kernel itself.

The need to have the kernel arbitrate every reference raises the issue of efficiency. Surely one could construct a kernel which is essentially interpretive. That is, each access program a[i] performs the desired operation for the objects that request it.

As an illustrative example, a system might require as arguments to the evaluation program $\mathcal{E}$ two security objects; a terminal and a job, and the name of an access type. Then an interpretive version of write would be invoked by $\mathcal{E}$ to do that single operation, each time it is attempted.

From a practical viewpoint, the overhead of such an implementation of a security system would be very high. The current solution to this efficiency problem appears quite satisfactory, and its spirit seems in retrospect almost obvious. The hardware is designed so that all accesses must pass through a few fast registers where hardware checks on address bounds and the like are performed. An access program a[i] just sets these registers and returns control to the job, allowing a whole class of operations to be performed without further intervention; operations such as reads on any location between x and x + k. In the general access control model, such an implementation can be easily described by making the segment a security object--a member of the set O.

It was pointed out that by grouping the tuples which compose descriptive data and also making each such group a security object, this model also includes the ability to protect access to that data itself, allowing a controlled way of changing access decisions. Multics uses this strategy. A directory contains protection information about other directories and segments.

In addition to aiding the isolation and specification of that portion of a system relevant to security, this model attempts to urge upon system theoreticians, designers and implementers the following tenets:

(1) security mechanisms can and should be isolated at the heart of a system;
(2) by doing so, it is possible to take great care to produce a faithful implementation of that resulting kernel; and
(3) since the kernel is small and isolated, its operation can be verified and certified.

This general access control model is meant as a first step toward these goals.

ACKNOWLEDGMENTS

Dr. Roger Schell originally motivated the issues that led to the approach of this paper.

REFERENCES

1. Conway, R. W., et al., "On the Implementation of Security Measures in Information Systems," Comm. ACM 15, 4 (April 1972), pp. 211-220.

2. Elspas, B., K. Levitt, R. Waldinger, and A. Waksman, "An Assessment of Techniques for Proving Program Correctness," ACM Computing Surveys 4, 2 (June 1972), pp. 97-147.

3. Friedman, T., "The Authorization Problem in Shared Files," IBM Systems Journal 9, 4 (1970), pp. 258-280.

4. Graham, G. S., and P. J. Denning, "Protection--Principles and Practice," AFIPS Conf. Proc. 40 (SJCC 1972), pp. 417-429.

5. Graham, R. M., "Protection in an Information Processing Utility," Comm. ACM 11, 5 (May 1968), pp. 365-369.

6. Hoffman, L., "Computers and Privacy: A Survey," Computing Surveys 1, 2 (June 1969), pp. 85-103.

7. Hoffman, L. F., "The Formulary Model for Flexible Privacy and Access Controls," AFIPS Conf. Proc. 39 (FJCC 1971), pp. 587-601.

8. Lampson, B. W., "Dynamic Protection Structures," AFIPS Conf. Proc. 35 (FJCC 1969), pp. 27-38.

9. Lampson, B. W., "Protection," Proc. 5th Princeton Conf. on Information Sciences and Systems (March 1971), pp. 437-443.

10. Schell, R. (Head of Computer Security Branch, USAF/ESD), private communication, July 1972.

11. Computer Security Technology Planning Study, ESD-TR-73-51, USAF Hanscom Field, October 1972.