On Access Checking in Capability-Based Systems[1]

Richard Y. Kain *Carl E. Landwehr*
University of Minnesota Naval Research Laboratory

ABSTRACT

Public descriptions of capability-based system designs often do not clarify the necessary details concerning the propagation of access rights within the systems. A casual reader may assume that it is adequate for capabilities to be passed in accordance with the rules for data copying. A system using such a rule cannot enforce either the military security policy or the Bell and LaPadula rules. The paper shows why this problem arises and provides a taxonomy of capability-based designs. Within the space of design options defined by the taxonomy we identify a class of designs that cannot enforce the Bell-LaPadula rules and two designs that do allow their enforcement.

Index Terms--Access control, capabilities, capability-based architectures, security policy, *-property, taxonomy.

1. Introduction

Capability systems were first described in the literature in the mid-1960's. Their informal descriptions are typically based upon the notion that a capability is equivalent to a ''ticket,'' in the sense that possession of the ticket allows the possessing process access to the object described in the capability, provided that the access mode is compatible with the ''access rights'' stored within the capability. Several systems using the capability concept have been marketed (IBM System 38, CAP, i432, Plessey S250) [1].

Whether a computer system based upon capabilities can provably enforce the DoD security policy [2] has been a matter of discussion for some time. Boebert [3] has argued that an ''unmodified'' capability machine must be incapable of enforcing the *-property defined by Bell and LaPadula [4]. Since the *-property is an essential ingredient of the most widely used model of DoD security policy, this result implied that an ''unmodified'' capability machine cannot meet the DoD requirements. Boebert's discussion introduces the undefined term ''unmodified capability machine.'' In this paper we describe several classes of capability machine designs for managing access control information and show some that some classes cannot meet the DoD requirements but others can. We thereby circumvent a debate about the meaning of the term ''unmodified capability machine.''

1. Published in *IEEE Trans. on Software Engineering Vol. SE-13, No. 2* (Feb. 1987) pp. 202-207; Reprinted from *Proc. 1986 IEEE Symposium on Security and Privacy*, April, 1986, Oakland, CA; shared the Symposium Best Paper Award. This effort was supported in part by U. S. Government Contract MDA904-C-84-C-6011 and the Naval Space and Warfare Systems Command.

This paper begins with brief definitions of the basic notions concerning capabilities and capability machines. We next consider the sequence of events a capability may undergo between the creation of a segment and an access to that segment, and we discuss strategies for controlling access rights in this context. A design taxonomy is developed to describe these options. Finally, we show some classes in the design taxonomy that are not compatible with the DoD security policy.

2. Basic Notions

The basic notions center on the properties of data and processes, the descriptions of these entities, the mechanisms that control access, and the policies that define ‘‘correct’’ access limitations. We start with data, processes, and capabilities.

Definition: A *segment* is a group of data possessing identical security attributes. Additionally, the segment may contain a set of capabilities possessing identical security attributes; these security attributes need not necessarily be the same as the attributes of the data contained within the segment. A segment that can only hold capabilities is called a *capability list* or *c-list*.

Definition: A *capability* is an object describing a segment, and possibly containing access rights or other access control information, as described below. Note that if the capability contains access rights, it must be distinguished from data to prevent unauthorized changes to the access rights, which, if permitted, would defeat all attempts to limit access based upon the access rights present in accessible capabilities. Capabilities can be distinguished from data either by tagging or by limiting their locations to distinguished segments or portions of segments that may only contain capabilities.

Definition: A segment possesses certain *data security attributes*, including, but not limited to, a security level and an access control list. In addition, the segment may possess a separate set of *capability security attributes* describing any capabilities stored within the segment. All data within the segment possess the data security attributes associated with the segment. All capabilities within the segment possess the capability security attributes associated with the segment.

Definition: A *process* is the execution of a program on behalf of a user logged in at a certain security level.

Definition: The *security attributes* of a process include a security level and the identity of the user on whose behalf the program is executed. A process may have other attributes, such as its domain of execution [5], in certain designs.

Definition: A *reference monitor* is a mechanism for checking each attempted access by a process to an item within a segment for conformance with the access modes permitted for that process to that segment. A process can attempt access to a segment only via a capability that has been prepared for access (*e.g.*, by placing it in a *capability register*). Capabilities prepared for access are not shared among processes.

Definition: A *security policy* is a set of rules for determining the maximum permissible access rights for a particular process to a particular segment, given the attributes of both the process and the segment.

Definition: The *DoD mandatory security policy* limits the access rights to a segment based upon a comparison between the security level of the segment and the security level of the accessing process. Write is allowed if the level of the segment dominates the level of the process, read if the level of the process dominates the level of the segment. Domination is a reflexive relation, so that both read and write are permitted if the two levels are identical. (The DoD policy requirements have been given in detail elsewhere [2].)

Definition: The *DoD discretionary security policy* states that the maximum access rights allowed for a process to access a segment may not exceed those defined for the user (on whose behalf the process is executing) in the access control list of the segment.

In this paper we will explore system design options that seek to enforce DoD's mandatory and discretionary access control policies.

3. Event Sequences

In order to understand the system design options, we must understand the possible sequences of events between the time that an object is created and the time that the object is accessed by a process. In creating the list of possible sequences, we have taken care to cover the sequences appropriate for various types of capability-based system designs. Since all objects will be segments, we refer to objects as segments and name them S with numeric subscripts as necessary to distinguish among several segments.

We consider the following significant events here:

1. Creating the segment;
2. Creating a capability for the segment;
3. Changing the security attributes of the segment;
4. Copying a capability for the segment;
5. Preparing a capability for use in accessing the segment;
6. Accessing the segment.

Segment creation is significant because upon creation the segment S is assigned its initial security attributes. In addition, the system may create a capability $C(S)$ that describes segment S and return that capability as a result. Certain access rights may be placed in $C(S)$ at this time. Alternatively, the system may associate S with a symbolic name $N(S)$ which can later be used to retrieve or construct a capability for the segment S.

A capability for segment S can be created by presenting the segment's name to a service function. This operation may not be permitted in certain system designs, though it can be implemented under most systems as a file system utility. As with segment creation, the result of this operation may be a capability containing access rights.

A segment's security attributes may be changed. This can occur by changing the level of the information in the segment or by changing the contents of the access control list associated with the segment. In most systems the former operation is disallowed, while the latter is essential for discretionary access to be useful. The act of changing S's attributes may necessitate changing the access rights in some capabilities for S.

A capability $C(S)$ describing S may be copied. This activity may be essential if the ability to access S is to be propagated to another process. This operation may connote access rights checking or changing in certain system designs.

The effect of preparing a capability for access is to associate a short name (such as a register number or an index quantity) with the segment described by the capability. In some designs a capability is prepared for access by loading it into a capability register. In other designs placing $C(S)$ into a designated segment or performing certain operations on $C(S)$ may have the effect of preparing S for access. The proper access rights for the executing process to make access to S

may be determined as the capability is prepared for access.

The processor attempts to access S when it constructs a virtual (or physical) address which corresponds to the contents of S. At this time the reference monitor must check that the attempted access is compatible with the security policy.

4. Strategies for Access Right Control

Enforcing a security policy means controlling the rights users can acquire to data. In a capability machine, a user's rights are defined by the capabilities he (or a process acting on his behalf) can obtain and the access rights conferred by those capabilities. Propagation of access rights can be limited based on attributes of the place the capability is stored, on the context in which the capability is used, or on both. Control based on context of use is most appropriate for security purposes, since the same capability, residing in the same segment, might lead to a security violation if used for access in one context (e.g., that of a confidential process) but not in another (a secret process).

Strategies for controlling access rights can be organized according to what is checked, when it is checked, and the actions possible following a check. In a conventional system that permits free passing of capabilities, no checks are needed when a capability is created or simply copied from one segment to another. Checks are required only when a process uses a capability to apply some operation (e.g., read, write, execute) to a segment. At that time it is only necessary to check that the requested operation is consistent with the access rights in the capability.

For a system to enforce the DoD security policy, more checking is required. Specifically, the security levels of processes and segments must be checked, as must segment access control lists. Some systems [3] also associate types with segments; operations applied to segments must then be checked for type compatibility.

These additional checks can be made when a segment (hence an initial capability for it) is created, when a process moves a capability from one segment to another, when the capability is prepared for access to a segment, or when the processor actually attempts an access to an item within a segment. Changes to a segment's security level or access set, if permitted, may also require some checking. The outcome of a check may be that the requested operation is permitted, is denied outright, or that the operation occurs in a restricted way. For example, the access rights in a copied version of a capability might be restricted based on a check of the security level of the segment in which the copy is stored.

The following section provides a taxonomy of capability system designs based on these considerations and explores these issues in more detail.

5. Proposed Taxonomy

Our taxonomy is based upon the answers to the following five questions:

1. What happens when a capability is created?
2. What happens to the capabilities describing a segment if the security attributes of that segment are modified?
3. What happens when a capability is copied?

4. What happens when a capability is prepared for access?
5. What happens when the processor attempts to access an item?

All of these questions are concerned with the access rights within the capability and with the checking of same during these activities. The possible answers will be discussed as we elaborate on the questions. When we describe a system design in these terms, we will form a sextuple holding the answers to these questions in the order of their numbering above. There will be six answers, since question 2 contains two subquestions which may have different answers.

Question 1 concerns the action taken when an object is created or when a capability is reconstructed (if possible) from a file’s path name. The system may insert access rights into the new capability at that time. A typical design decision would place a comprehensive set of access rights in a capability upon object creation, since the capability is then delivered to the owner of the new object, who, it could be argued, should be allowed to perform any operation on the new object. The possible answers to this question are the following:

a) No access rights inserted;

b) Access rights inserted.

Question 2 concerns the modifications to a capability when the security attributes of the segment it describes are changed. It may seem that this situation cannot arise, since it could be argued that any reasonable system design would not change the sensitivity of the information contained in a segment. While this is probably a valid argument, it does not cover the case when the access control list associated with the segment is changed. (The action in this situation is significant since the answer to the question determines whether the changes are or are not reflected in the access rights that could be exercised immediately.) There are really two subquestions, since the action may depend upon the location of the capability in question. The two possibilities are that the capability in question has been prepared for access by some process, or that the capability in question is located in a segment. For example, if the capability is located in a capability register it may not be updated, while if it resides within a shared memory space, it might be updated. The general question is how the capabilities pointing to a segment S are updated when S’s security attributes change. The two subquestions have the same set of possible answers:

a) Access rights not changed upon attribute change.

b) Capability is flagged for future change upon attribute change.

c) Access rights are updated upon attribute change.

The first subquestion concerns capabilities that have been made ready for access, whereas the second concerns the capabilities residing in shared memory.

Question 3 concerns the action taken when a capability is copied. One possibility is that the access rights contained therein are unchanged. (This is the only possibility if no access rights were inserted when the capability was created.) It is also possible that the access rights are updated by the system in accordance with a built-in policy. One such policy would be that the rights be limited so that they do not exceed the maximum rights for access attempts from a process executing in a security context that is identical to the security properties of the destination segment of the copy operation. A variant of this policy would expand rights to this maximum. Finally, it is possible that the access rights are to be modified, but by ‘‘correct’’ action taken by trusted software. This approach was taken in the PSOS design [6]. Under this approach it is difficult to guarantee that the context in which the capability will be used is compatible with the access rights set into the capability when it was copied from its originator. The possible answers are:

a) Access rights not changed.

b) Access rights further restricted by context rules.

c) Access rights set to the maximum consistent with the access rules set by the policy.

d) Access rights guaranteed to be updated properly by software.

Question 4 concerns the action taken when a capability is presented to the system in preparation for making access to the object it describes. The specifics of this action depend upon the system design; it may be the act of loading a capability (or base) register, or it might be the act of loading the capability into a naming table. The question concerns whether the access rights in the capability are simply copied at this point or whether they are computed at this time. A third possibility is that they are limited by both the access rights in the capability and the computed maximum access rights allowed to the executing process for access to the described segment. The possible answers to this question are:

a) Access rights not changed.

b) Access rights restricted by the access rights policy.

c) Access rights set to the maximum consistent with the security policy in force.

Question 5 concerns the actions taken when the processor actually attempts to access an item in a segment. It is possible that there are no checks at this point, though that case is not compatible with the reference monitor approach to secure systems design, and probably would be ruled out on that count. The other possibilities are that the access rights are computed and checked, or simply that those present in the capability are checked. Most designs simply check the rights at this time, since recomputing them can be tedious. Note, however, that the protection ring scheme [7] implies a trivial computation of the rights (by comparing the integers defining the ring of execution and the ring brackets of the segment) upon access. The possible answers to the fifth question are:

a) No checks are made.

b) The access is checked against the available access rights.

c) The maximum possible access rights are computed and the attempted access is checked against these computed rights.

6. Using the Taxonomy

Any design can be described by the sequence of choices made. For example, the Honeywell Secure Ada Target (SAT) [5] is *aaaacb*. An ‘‘unmodified capability machine’’ is *baa*(*a* or *d*)*ab*; the Plessey System 250, as described by Levy[1], appears to be in this class. Not all combinations make sense. For example, *aaaaaa* implies no checking or access rights computations, and so corresponds to a traditional machine design without security features. On the other hand, *aaaaab* implies a check of access rights that have never been determined, which is nonsense.

*A Class of Designs that Cannot Enforce the *-Property*

Now consider the class of designs in which the access rights are checked upon access but are not modified upon capability copying. This class of designs could be described as *bxxaab*, where *x* denotes a ‘‘don’t care’’ answer. Boebert [3] considered the simple propagation rule that capabilities be propagated like data (following Bell and LaPadula rules). He showed that the *-property cannot be guaranteed in a system in which the possession of a capability with read-capability access for a segment S is tantamount to possessing the access rights conferred by all capabilities stored in S. For example, a design permitting an executing process to copy a capability C from segment S_1 to S_2 effectively confers the access rights of C on all processes that can read capabilities from S_2. Trouble results if one copies a capability C from a segment S_1 with a low security level ($L(S_1)$) to a segment S_3 with a higher security level ($L(S_3)$), and if capability C describes a segment S_3 at level $L(S_2)$ ($L(S_1) < L(S_2) < L(S_3)$), and C permits write access to S_2. If used at level $L(S_1)$ this access right is correct since the write causes information to flow upwards in security level. After the copy is completed, the same capability can be used to write into the segment at level $L(S_2)$ from the higher level $L(S_3)$. This writedown operation violates the *-property.

*Two Designs that Can Enforce the *-Property*

We now describe two designs that permit enforcement of the *-property. The first is more complex and restricts access rights when capabilities are copied. It distinguishes the security level associated with the capabilities stored in a segment, called its *capability security level* from that associated with the data in a segment (its *data security level*). The second design does not require this distinction and permits free copying of capabilities but restricts access rights when a capability is prepared for access.

In the first design, let $L(P)$ denote the security level of process P, let $L_d(S)$ denote the data security level of segment S, and $L_c(S)$ its capability security level. Let $S(C)$ denote the segment referred to by capability C. In this design, the following checks are applied:

1. Capability creation: the capability C created at the request of process P for segment S is placed in another segment S_0 for which $L_c(S_0)=L(P)$, and the access rights of C are restricted as follows:

$L(P)>L_d(S)$:	Access rights of C set to read.
$L(P)=L_d(S)$:	Access rights of C unlimited.
$L(P)<L_d(S)$:	Access rights of C set to write.
Otherwise:	Access rights set to no-access.

2. Change of S_i's attributes: if change is to discretionary security attributes only, no change. If mandatory security level ($L_d(S)$) is changed and new data security level is $L_d(S_i)$):

In memory: for each capability C in S_j such that $S(C)=S_i$, apply

$L_c(S_j)>L_d(S_i)$:	Access rights of C limited to read.
$L_c(S_j)=L_d(S_i)$:	Access rights of C not changed.
$L_c(S_j)<L_d(S_i)$:	Access rights of C limited to write.
Otherwise:	Access rights set to no-access.

Prepared for access: no change.

3. Capability copy: $L_c(S_{src})$ is the capability security level of the source segment (from which the capability is to be copied), and $L_c(S_{dest})$ is the capability security level of the destination segment (where the copied capability is to be placed).

$L_c(S_{src}) < L_c(S_{dest})$:	Access rights of C limited to read.
$L_c(S_{src}) = L_c(S_{dest})$:	Access rights of C not changed.
$L_c(S_{src}) > L_c(S_{dest})$:	Access rights of C limited to write.
Otherwise:	Access rights set to no-access.

4. Prepare capability C from segment S for access:

$L(P) = L_c(S)$:	Access rights of prepared capability copied from C unchanged.
Otherwise:	Access rights of prepared capability set to no-access.

5. Action on access of P to S via C: check type of requested access against access rights in C.

This set of rules limits capabilities' access rights at their creation, when they are passed, and when they are prepared for access; only the conventional checking of the access rights already set in the capability is required on actual access. This scheme is characterized by the taxonomy as *baabcb*. To show that this scheme is effective in enforcing the *-property, it suffices to show that that no process can prepare for access a capability C permitting write access for a segment S unless $L(P) \leq L_d(S)$.

Proof: Suppose a capability C for segment S is created at the request of process P_r and stored in segment S_0. The capability may then be copied from S_0 to some S_1, from S_1 to S_2, and so on until it reaches some segment S_n. It is then prepared for access by a (possibly different) process P_p. We want to show that for P_p to successfully prepare C for access with write access implies that $L(P) \leq L_d(S)$. Notice that rules 1-4 can never increase the access rights for C once it is created, so write access must be present in C initially, which implies (by rule 1) that $L(P_r) \leq L_d(S)$. We also know $L_c(S_0) = L(P_r)$.

First, assume that the (mandatory) security levels of segments are fixed. Then for C to contain write-access when placed in S_n, it must be the case that

$$L_c(S_n) \leq L_c(S_{n-1}) \leq \cdots \leq L_c(S_0)$$

by rule 3. So,

$$L(P_p) = L_c(S_n) \leq L_c(S_0) = L(P_r) \leq L_d(S).$$

Thus, in order for write access to be present in the capability prepared for access, it must be the case that $L(P_p) \leq L_d(S)$, as required.

Permitting the (mandatory) data security level of a segment to change implies that the final value of $L_d(S)$ might differ from its initial value. Note that whenever such a change occurs, rule 2 must be applied to all capabilities for S, and if the new data security level of S is not greater than or equal to the capability security level of the segment containing a particular C for S, write access is removed from that capability. Thus the *-property is still preserved as long as no capabilities for S are prepared for access when the change occurs[2]. Enforcement of the simple security condition can be proven similarly.

In the SAT design, characterized as *aaaacb*, a similar effect is achieved by delaying the computation of access rights until the capability is prepared for access. At that time, the security levels of the process and segment referred to by the capability are known, and the Bell-LaPadula

2. This condition can be met, for example, by forcing all capabilities currently prepared for access to be re-validated (this yields a system of type *bacbcb*) or by inhibiting changes to $L_d(S)$ until no capabilities for S are prepared for access.

rules can be used to limit access rights at this point. The precomputed access rights are then checked on the actual access. The design proposed by Karger and Herbert [8] is essentially similar: their notion of evaluating a capability and placing it in a cache corresponds to preparing the capability for access.

Discussion

In general one may ascertain some failures of a proposed design with respect to meeting the DoD security requirements by finding the number of the last answer to the taxonomic questions which is not a. For example, if the answer to question 4 is not a, then any access rights are frozen into the access checking mechanism at the time the capability is made ready for access; this may violate future access rights changes, for example, if the access control list for the segment is changed to void access by the user. If the answer to question 4 is a, one has to look at the combination of the answers to questions 2 and 3 to determine the rules enforced by the system. For example, if the security levels of segments and processes are fixed, access rights can still be restricted when capabilities are copied so that the copy restricts processes that can use it to the kinds of access authorized by the security policy. But at the same time, this approach may make capability copying more difficult.

7. Summary

We have developed a set of attributes of system designs having the property that their combination confines the security properties of the system in useful ways. By examining the pattern of the answers to the questions, one might easily determine the limitations of the design with respect to a proposed security policy that the system is supposed to enforce. This taxonomy clarifies the position of many previous designs and details the failure presented by Boebert [3]. Two designs that avoid this failure have also been presented.

8. References

[1] Levy, H.M., *Capability-Based Computer Systems*, Digital Press, Bedford, MA, 1984.

[2] Department of Defense Computer Security Center, *Trusted Computer Systems Evaluation Criteria*, CSC-STD-001-83, August 1983.

[3] Boebert, W. E., ‘‘On the Inability of an Unmodified Capability Machine to Enforce the *-Property,’’ *Proc. 7th DoD/NBS Computer Security Conference*, September 1984, pp. 291-293.

[4] Bell, D. E., and LaPadula, L. J., ‘‘Secure Computer System: Unified Exposition and Multics Interpretations,’’ Tech. Report MTR-2997, MITRE Corp. Bedford, Mass., July, 1975.

[5] Boebert, W. E., Kain, R. Y., Young, W. D., and Hansohn, S. A., ‘‘Secure Ada Target: Issues, System Design, and Verification,’’ *Proc. 1985 Symp. on Computer Security and Privacy*, pp. 176-183, April 1985.

[6] Neumann, P. G., Boyer, R. S., Feiertag, R. J., Levitt, K. N., and Robinson, L., ‘‘A Provably Secure Operating System: The System, Its Applications, and Proofs,’’ SRI Computer Science Laboratory Report CSL-116 (2nd ed.), May 1980.

[7] Schroeder, M. D., ‘‘Engineering a Security Kernel for Multics,’’ *Proc. 5th Symp. on Operating Systems Principles* (also *ACM SIGOPS Review 9*, 5), pp. 25-32, November 1975.

[8] Karger, P.A., and Herbert, A.J., ‘‘An Augmented Capability Architecture to Support Lattice Security and Traceability of Access,’’ *Proc. 1984 Symp. on Security and Privacy*, pp. 2-12, April, 1984.