To: Distribution

From: Richard Bratt

Subject: A New Access Authorization Mechanism

Date: January 27, 1977

Perhaps the most difficult decision to be made when designing a protection and authorization mechanism is deciding how changes in access authorization are to be authorized. In making this crucial decision two approaches are possible. First, one may design a very general, and possibly complex, authorization control mechanism capable of supporting a diverse set of authorization control policies. Second, one may chose an appropriate and sufficient authorization control policy and then design minimal efficient mechanisms to support the chosen policy. The problem with the latter approach, which was taken in the design of Multics, is that as the needs of the user community change, new authorization control policies become both desirable and appropriate. Unfortunately, it is often a formidable task to retrofit mechanisms supporting new or different authorization control policies into a system. This document discusses an authorization control policy which has been identified as being a desirable adjunct to Multics and describes a simple mechanism which allows graceful and natural integration of this new policy with the existing Multics authorization control policies and mechanisms.

Multics, if analyzed by Freud, would probably be accused of suffering from acute hierarchy fixation. Our storage system is hierarchically structured. Our secondary storage resource control system is hierarchically organized. Our global user name space is hierarchically structured. Our authorization control policies are hierarchical. As a further complication, these logically independent hierarchies have been mapped, in the Multics design, into a single physical hierarchy. This unnatural coercion has many deleterious effects upon the structure and function of the system. This document will concern itself primarily with the effects of the unification of the storage system hierarchy and the authorization control hierarchy on Multics authorization control. Before investigating the misinteractions between the existent Multics access authorization mechanism and storage mechanism I will briefly review the salient features of the Multics access authorization mechanism.

Access authorization in Multics is specified by

Multics Project internal working documentation. Not to be reproduced or distributed outside the Multics Project.

associating an access control list with each object in the storage system. An access control list specifies the access rights of any given principal to the associated object. The current Multics design authorizes authorization changes by treating the access control list associated with an object as an attribute of the object, stored in the directory cataloging the object and thus subject to modification by those principals who have modify permission to the containing directory. Multics authorization control is thus based upon a hierarchical model.

Hierarchical authorization control has many advantages. It seems to support quite naturally many desired authorization schemes. (1) It is easy to implement. It is easy to understand. As a result, the Multics access control mechanism has been quite successful. Unfortunately our implementation of the hierarchical access control model is flawed. We have mapped the access control hierarchy onto the storage system hierarchy and thus onto the secondary storage control hierarchy with a consequent strong coupling of access and storage control.

Coupling authorization and resource control is disadvantageous because a common class of desirable real world policies which may be modelled assuming disjoint hierarchical access control and storage system resource control cannot be coerced into a unified hierarchical model that assumes authority to control resources implies authority to control access and vice versa. As a classic example, consider the case of administration of a computing utility. Clearly, the system administrator must have the power to authorize a customer to consume secondary storage resources. Similarly, he must have the authority to reclaim the secondary storage resources used by a customer in default of his contract with the computing utility. On the other hand, it is entirely unreasonable to assume that the system administrator should have the authority to read and/or modify the information stored and processed in the computing utility by its customers. This policy is unrealizable in the current Multics system. To control secondary storage resources the system administrator must be given modify permission on the directories of his customers which allows him to inspect and damage his customer's information.

There exist many schemes for adding the capability of dealing with the "system administrator" problem to Multics. This paper presents an extremely simple modification to the Multics access control mechanism which I believe provides the desired capability in a natural, easy to understand way. The scheme I

----

(1) As the reader is doubtless aware, many useful, real life authorization policies are unrealizable within the framework of simple hierarchical access control. For example, no analogue of the policy, "it takes two keys to open the vault," can be specified with the mechanism described.

will present has a very minimal impact upon the system.

I propose that a facility exist for subdividing the access control hierarchy into multiple, disjoint access control hierarchies. To wit, I suggest that an attribute be added to each node of the Multics storage system hierarchy which specifies whether the given node belongs to the same access control hierarchy as its father or is the root node of a new access control hierarchy. In this way the structure of the system is preserved. The access control list of an object still is an attribute of the object and contained in its parent directory. However, the ability to modify the access control list of an object is only granted if the process requesting the modification has modify permission to the parent directory <u>and</u> the object is not the root of a new access control hierarchy. The ability to destroy and to move quota into and out of the storage system subhierarchy defined by an access control hierarchy is still controlled by modify permission on the parent of the access control hierarchy.

As described so far, this scheme lacks two important mechanisms. First, this scheme does not provide a mechanism for authorization modification on the root node of an access control hierarchy. Second, this scheme does not provide any control over the ability to define a new access control hierarchy. (1)

One solution to the problem of authorizing changes to the root node of an access control hierarchy is to appeal to the mechanism used in contemporary Multics to authorize changes to its single access control hierarchy root node. This scheme would authorize only the system itself to modify the attributes of the root node of an access control hierarchy, and hence its access control list. Unfortunately, though simple, such a mechanism directly contradicts the Multics policy of distributed control. A more appropriate solution, which fits nicely into the current access control mechanism, is to introduce the concept of self control, i.e. allow the access control list of the root node of an access control hierarchy to somehow specify who may modify the access control hierarchy root node. A mechanism to provide just this type of control has already been proposed by Van Vleck. This mechanism defines a new access control list mode "o", standing for "owner". This permission confers upon a principal the right to operate upon the <u>given</u> object as if the principal had modify permission to the parent of the object. (2)

----

(1) The operation of defining a new access control hierarchy may be thought of as "rerooting" an access control hierarchy since it removes a subhierarchy from an existing access control hierarchy and plants the root node.

(2) Note that the "owner" permission mechanism is completely orthogonal to the mechanism being proposed. This "owner"

Before introducing a mechanism to authorize the creation of new access control hierarchies, it is instructive to investigate what it means to create a new access control hierarchy and what controls might be desirable. As envisoned, creating a new access control hierarchy is done by giving an object (of either gender) the ROOT attribute. There seems to be no reason to constrain this differentiation to occur at the time the node is created nor does there seem to be any reason to disallow the removal of the ROOT attribute at some point in the future, independent of the destruction of the object. Therefore, the storage system hierarchy is covered by a family of access control hierarchies which may vary from instant to instant. (1) It should be obvious that some authority is necessary to root an access control hierarchy since the act of rooting a new access control hierarchy potentially denies access to the subtree to principals having modify permission on the parent directory.

A natural solution to the problem of authorizing the rooting (and uprooting) of an access control hierarchy is to require modify permission (2) to the object. This solution, with a single exception, appears to exhibit the desired behavior. The exception deals with the desire on the part of a contractor to audit the activities of his hired agents. If a building contractor was not allowed to oversee his empolyees activities, then he would have no way of assuring himself that he was not being robbed blind. An analogous situation arises in a computing utility. A programming project manager might require the ability to inspect the storage used by his empolyees to discourage unauthorized used of the computer resources he is paying for. If his employees could root a new access control hierarchy, then they could hide information from his view.

For this reason authorization to root a new access control hierarchy should be delegated much as authorization to consume secondary storage resources is delegated. A new object attribute, ROOTABLE, can be invented to control this delegation. The ROOTABLE attribute specifies that the object may serve as the root node of a new access control hierarchy. Delegation of the

----

permission mechanism is only being used as a solution to the problem of authorizing modifications to the root node of an access control hierarchy. Note also that the "owner" permission mechanism could be used to take the access control policy on the storage system hierarchy root node "out of the closet".

(1) The reader should convince himself that the dynamics of the situation do not introduce access revocation problems.

(2) Modify permission may be vested in a process by virtue of having modify permission on the containing the object or being an "owner" of the object.

authority to make a node ROOTABLE follows three simple rules. One, the root node of the storage system hierarchy is de facto ROOTABLE. Two, a process may mark an object as ROOTABLE if its parent is ROOTABLE and the process has modify permission to the object. Three, once delegated the ROOTABLE attribute may not be removed. (1)

In summary, I propose the addition of three primitives to the system: hcs_$delegate_ach_rootability, hcs_$root_ach, and hcs_$uproot_ach. (2) The hcs_$delegate_ach_rootability marks a designated node as rootable if the process has modify permission to the node and the immediate superior of the given node is rootable. The hcs_$root_ach primitive marks a node as the root of a new, independent access control hierarchy if the node is marked as rootable and the process has modify permission to the node. The hcs_$uproot_ach primitive causes a designated node to be marked as a normal, non access control hierarchy root node if the process has modify permission to the node.

Access to modify the attributes of a normal hierarchy node is controlled by both the access control list on the containing directory and the access control list on the given node ("owner" mode). Access to modify the attributes of an access control hierarchy root node is controlled primarily by "owner" access to the node. (3) The posession of modify permission on the parent of an access control hierarchy only permits a process to perform resource control operations, e.g. delete the whole subtree or move quota in and out of the subtree.

----

(1) Except, of course, by deleting the whole subtree. This restriction is stronger than necessary and may be weakened if experience suggests that doing so would be advantageous.

(2) I hope better names will suggest themselves if this scheme comes to fruition.

(3) To prevent "lost" items the access control list primitives should probably refuse to create an access control hierarchy with no "owner" permission on its root node. This, of course, is insufficient and the system will have to supply a highly privileged locksmithing primitive to deal with unaccessible nodes. To discourage misuse, this facility must record an indelible audit trail of its activities.