TO: A. Bensoussan
C. T. Clingen
F. J. Corbató
J. W. Gintell
N. I. Morris
J. H. Saltzer
M. D. Schroeder
S. H. Webber

FROM: R. J. Feiertag, V. L. Voydock

DATE: July 22, 1971

SUBJECT: Proposals for access control

Enclosed are some documents describing additional proposals for modifications to the Multics access control mechanism. A meeting to discuss these proposals will be held at 3:00 p.m., Tuesday, July 27, in Room 511.

TO: A. Bensoussan
C. T. Clingen
F. J. Corbató
J. W. Gintell
N. I. Morris
J. H. Saltzer
M. D. Schroeder
S. H. Webber

FROM: R. J. Feiertag, V. L. Voydock

DATE: July 20, 1971

SUBJECT: Additional proposals for access control

Discussion subsequent to the proposals of the June 7 document have caused reconsideration of some of the ideas in that document and consideration of some new ideas. The following paragraphs describe two ideas which we now believe should be implemented.

The first of these ideas is to associate ring brackets with segments only instead of with users and segments. Recall that this change does not cause any loss of functional capability, for anything that could be done using different ring brackets for different users on the same segment can be done by programming a separate procedure to run in a privileged ring to accomplish the same result by software. This change does not affect most users since most users do not use ring brackets at all and of those that do, probably none have different sets of ring brackets on the same segment. We had previously decided to keep ring brackets as they were for two reasons. First, although allowing multiple sets of ring brackets were used in only a few cases it was very convenient for those cases. Secondly, leaving

ring brackets as they were required no work. These are not particularly strong arguments and the decision made at that time was a close one. Two new considerations have caused us to change the recommendations. First, one set of ring brackets per segment is much easier to explain. A segment can be said to reside in a particular set of rings irrespective of who is referencing it. Also the new scheme does not allow a user to have multiple sets of ring brackets even if he wishes to. Previously we had decided that most users would only use one set of ring brackets per segment, but they could use multiple sets if they desired. However, multiple sets is confusing enough that it is probably better not to let anyone use it. The second consideration concerns the problem of what ring brackets are associated with a user not specified on the ACL. With only one set of ring brackets per segment it is obviously these ring brackets that are used.

Implementing one set of ring brackets per segment in an upward compatible manner should be straightforward since it affects very few, if any, users. Once users have been given sufficient warning, the ACL primitives will be changed so that different sets of ring brackets on a segment are not permitted. A program can then be run over the hierarchy to delete any ACL entries that do not conform. This completes the change functionally. Eventually, the next time directory structures are reformatted, ring brackets should be removed from ACL entries and placed in the branch. The interface to the ALC primitives should be changed and some new primitives dealing with ring brackets should be implemented.

The second idea to be presented is the addition of a delete attribute to each ACL entry. Currently, permission to delete a segment requires both write permission in the segment and modify permission in the directory. It is the only primitive operation that requires checking for two attributes. The checking for write permission on the segment is really a crude way of providing a protection feature against unintentional deletions. One would really like to protect data segments that have write permission as well as procedure segments that do not. For this reason a new attribute, called the delete attribute, should be added to the rewa attributes already in each ACL entry. If the delete attribute is on then a call to the delete primitive will be successful; if the delete attribute is off then a call to the delete primitive will return an error code. The modify attribute of a directory would no longer control deleting of segments, only the modification of attributes of segments in the directory. Modification of attributes and deletion of segments would still be subject to the rule that the validation level be less than the r1 of the segment being affected.

The implementation of the delete attribute can be accomplished in an upward compatible manner in several steps. What is now called the trap attribute bit will be used for the delete attribute. The ACL primitives and the append primitive will be modified to properly set the delete attribute and the ACL commands can be changed to handle it. System programs that create segments and modifiy ACL, will be changed to turn the delete attribute on as the default. Users will be

notified of the change. Then for some period of time the delete primitive will delete segments if either the delete attribute is on or both the segment has write permission and the directory has modify permission, i.e., both the old and new criterion will be used. Once users have had a chance to accustom themselves to the new attribute the old delete criterion will be invalidated.

TO: A. Bensoussan
C. T. Clingen
F. J. Corbató
J. W. Gintell
N. I. Morris
J. H. Saltzer
M. D. Schroeder
S. H. Webber

FROM: R. J. Feiertag, V. L. Voydock

DATE: June 28, 1971

SUBJECT: Plans for Initial ACLs and Elimination of CACLs

The initial ACL is a means by which a user can specify the ACL to be added to a newly created segment in a specific directory. Each directory will contain two sets of initial ACLs, one for newly created directories and one for newly created non-directory segments. Each of these two sets will contain an initial ACL for each ring.

Each initial ACL will consist of a list of star names, i.e., file system entry names which may include the star ("*") character to indicate a class of names. With each star name will be associated a list of ACL entries. When a new segment is created via a call to append, the appropriate initial ACL will be found by using the type of the segment (directory or non-directory) and the current validation level. The list of star names is then searched for the first such star name that matches the name on the new segment. The list of ACL entries associated with this star name is then used to form the ACL of the new segment. The ACL entries specified in the call to append are then added to the new ACL.

New primitives and commands will be provided to manipulate initial ACLs. Separate commands will be provided to set entries (add or change), list entries, and delete entries for both initial ACLs applying to directories and non-directory segments. The validation level at the time of the operation will determine which ring's initial ACL is involved. When an entry is added to an initial ACL it is checked to make sure the specified

This guarantees that the initial ACL can be validly added to a new segment with no possibility of error. Star names will be ordered on the initial ACL in a manner similar to the way process group id star names are ordered on ACLs. The most specific names will be listed first in a manner that favors specificity in leftmost components. Rather than state the precise algorithm the following listing will indicate the ordering.

a

x.z.z.

a.b.c.d

a.*.b

a.**

*.a.b

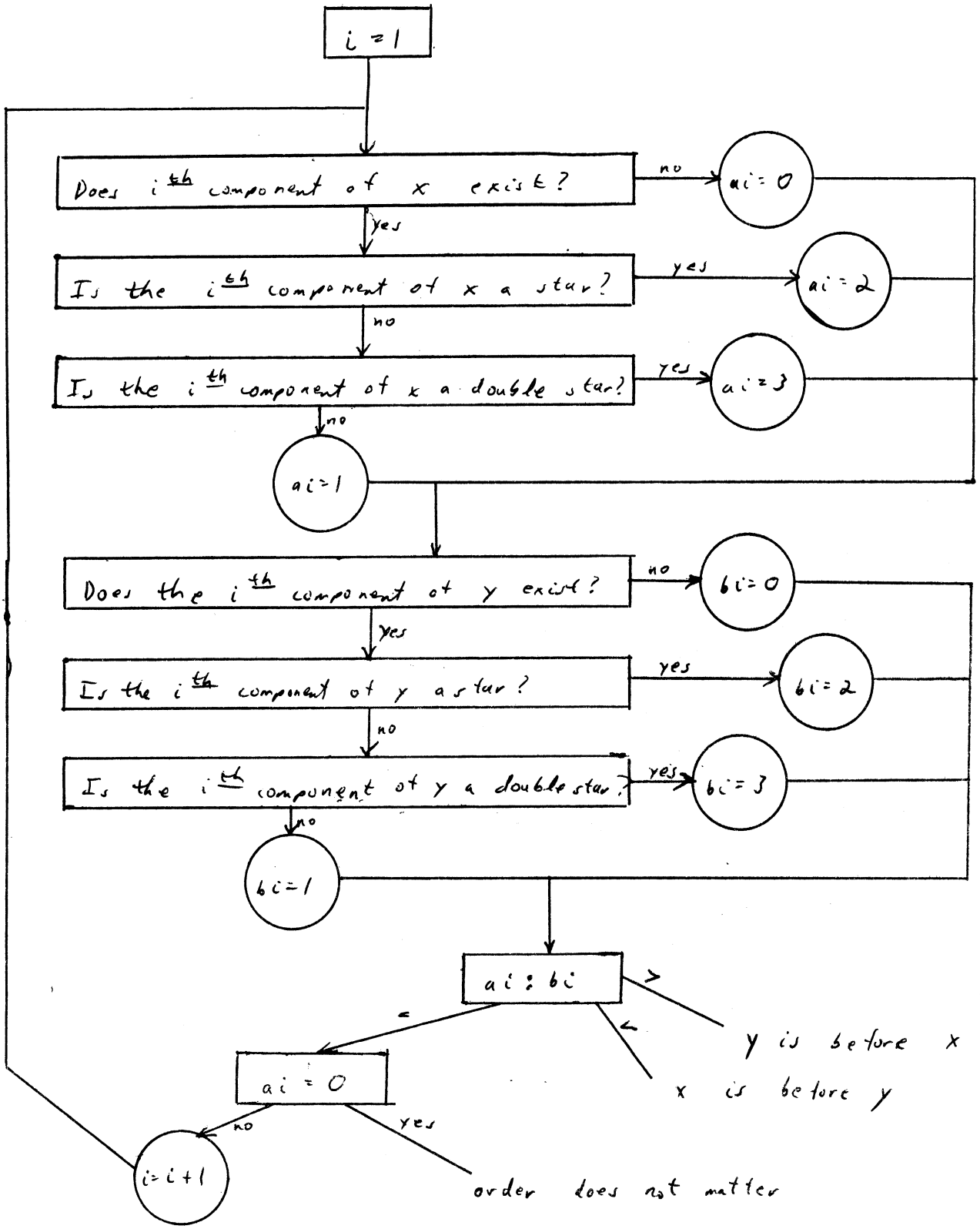
..a

**

In order to permit the easiest possible transition from CACLs to the use of ACLs and initial ACLs they should both exist simultaneously for some period of time. This means that initial ACLs will be implemented and installed. Users will then be notified to convert. Programs to do the conversion in a reasonable manner can be provided. When the transition period is ended, CACLs and their associated primitives will be disabled. CACLs can then be made unknown to the salvager which will cause their removal from the system.

Under the scheme described so far the setting of ACLs and initial ACLs are independent. It is expected that users may wish these to be coupled in some manner. However, there does not seem to be an obvious straightforward way of accomplishing this so we prefer to wait until more experience has been gained with initial ACLs and we get some suggestions.

Algorithm for determining which of two star names, x or y, should appear first on list.



TO:

FROM: V. Voydock, R. Feiertag, M. Schroeder

DATE: July 20, 1971

SUBJECT: Access control conventions for gates

As the reader is aware, the limitations of the ring mechanism make it necessary to insure that users of a protected subsystem in a given ring have no access to that ring other than through that subsystem. In practice this means that they all must be members of the same project or group of projects controlled by cooperating individuals. A user cannot be allowed to use two protected subsystems in the same ring unless the two subsystems completely trust each other. (The reasons for this have been discussed in memos by Schroeder and Clingen, and I will not repeat them here.) Thus the writer of a protected subsystem in a given ring must have complete control over which gates to that ring (or lower rings) his users can use.

In order to enforce this necessary constraint, some access control conventions for gates must be established. Two proposals have been made in this area. The first proposal is to not allow a user to put someone on the ACL of a gate unless that person is in the same project as he is. Though sufficient, this plan has two major disadvantages. First, if subsystem writer Smith wants to let subsystem writer Jones (in a different

project) use his subsystem, he must give Jones access to change the ACL of his gates (since Smith cannot do it himself). This allows Jones to freely give or deny access to Smith's subsystem to anyone. The second problem cuts the other way. We have noted that in order for Jones to use Smith's subsystem he must trust Smith completely. Let us suppose that Jones decides that he no longer trusts Smith and doesn't want his users to use Smith's subsystem. Since his users are on the ACL of Smith's gate and since Smith controls who has access to change the ACL of his gates, Jones can't take his users off the ACL unless Smith lets him. Thus, in general, Jones is powerless to prevent his users from using Smith's subsystem once he has allowed them to do so.

The second proposal does not have either of these disadvantages. It is to associate with every user, one list of gates per ring. The gate list for a given ring contains the pathnames of all gates into that ring that the user is allowed to use. If a user references a gate into ring N which is not on the gate list for ring N, an error will occur. The default contents of each gate list is controlled by the project administrator, and a user can change the gate list for a given ring only if his validation level is less than or equal to that ring. This insures that a project administrator with a protected subsystem in a given ring has complete control over which gates to that ring (and lower rings) his users can use; but it also allows his users to themselves control the gates they can use in cases where it is legal for them to do so.

A change to a gate list should take effect immediately. We may back off

from this if it proves too difficult to implement. For user convenience, sets of gates may be indicated by use of directory pathnames. If the pathname of a directory appears on the gate list for ring N, it means that any gate to ring N in that directory may be used. Gate lists only apply to gates used as gates. That is, a user can call a gate into ring N from ring N even if it is not on the gate list for ring N.

In conclusion, we feel that the gate list proposal is much more flexible than the first proposal and has none of its disadvantages. We, therefore, recommend that it be adopted.