TO: MSPM Distribution
FROM: M. A. Padlipsky
SUBJ: BG.9.00
DATE: 01/16/68

This revision covers the following changes to Access Control:

1. "Gates" are no longer part of the access control
    information stored in the branch. (As forthcoming
    revisions to BD.9.01 and BD.7.01 will indicate,
    gate information is now kept in the linkage segment.)

2. The trap attribute will not be implemented in
    Initial Multics.

3. Details of implementation and calling sequences
    have changed.

Identification

Access Control
C. C. Cushing, M. R. Thompson, L. Shpiz

Purpose

Access Control provides a means whereby users may protect
their segments from being accessed by certain or all other
users or even by themselves. An interaction between Segment
Control, Directory Control and Access Control, as explained
below, is necessary for complete implementation of the
access controlling mechanism.

Introduction

Access Control is called by Directory Control to evaluate
the access control information for a particular branch.
This access control information consists of a list of
user names and associated with each user name are five
items: a mode, a trap procedure, and two ring brackets
(expressed by three integers). See section BX.8.00 for
definition of terms and discussion of how the user causes
this information to be associated with a segment.

The name of the user and the access control information
for the branch in question are made available to Access
Control. Access Control then determines the mode of the
user with respect to this branch and returns this mode
and the ring brackets as found in the access information
to Directory Control.

The Access Control primitives determine the mode of the
user with respect to a branch by searching the access
control list (ACL) of the branch from the top until the
user's name or a generic name that includes this user
is found. In the context of this section the access control
list consists of the ACL of the branch followed, if necessary,
by the common access control list (CACL) of the directory
containing this branch.

The mode associated with this user in the ACL is the apparent
mode of the user with respect to the branch. If the apparent
mode indicates that the trap attribute is ON and the trap
procedure associated with this name in the ACL is processed,
the mode obtained from the trap procedure is the effective
mode of the user. If the trap attribute is OFF, the effective
mode is the apparent mode.

The effective mode is the mode which governs the use of
the segment by the current user or process. This mode
is used by Directory Control to determine if the operation
currently being requested is to be permitted. The effective
mode along with the ring brackets is also used by Segment
Control to set up the access information for the segment
in the user's descriptor segment.

The ring brackets consist of three ring numbers; the first
two numbers are the low and high bounds of the <u>access bracket</u>
and the third number is the high bound of the <u>call bracket</u>.
The low bound of the call bracket is automatically one
greater than the high bound of the access bracket (i.e.,
it is not permitted to have lower rings that are less
privileged than higher rings). The access bracket delimits
the rings in which the user's access to the segment is
determined by his effective mode. The call bracket delimits
the rings from which the user can only transfer to the
segment. Only linkage segments should have call brackets,
since the ring crossing fault occurs when control is transferred
to the called procedure's linkage section. (Note that
the call bracket is considered in conjunction with the
presence of "gates", as indicated in the linkage segment,
when a call is actually made; see also BD.7, BD.9.)

Direct access to the segment from rings lower than the
access bracket is controlled by the user's effective mode
except for the use of the execute attribute. An attempt
to execute the segment from a lower ring (given the execute
attribute on for the user) results in a ring crossing
(BD.9.00). Direct access to the segment from the rings
in the access bracket is controlled by the user's effective
mode and attempted execution of the segment (given the
execute permission) does not cause a ring crossing. Direct
access to the segment from the rings in the call bracket
is denied, but attempted execution (given the execute
permission) is allowed - subject to subsequent action
by the Gatekeeper (BD.9.01) - and causes a ring crossing.
All access from rings higher than the call bracket is
denied.

Special Users

There are a number of special system processes which are
allowed some standard access to all files. These special
processes fall into two classes, those processes which
cannot be denied their standard access to any file and
those processes that can be explicitly denied access to
any file. To deny access to a segment to the second class
of special processes, the explicit name of the process
(the star, "*", convention does not apply here) must appear
on the access control list in the branch of that segment
with the desired attributes turned OFF.

The list of all such processes and the access mode that
they are allowed is pre-set in a hard-core system data
base called the special process access control list (SPACL).
Each time Access Control is called to evaluate a user's
mode, it first checks to see if that user is a special
process in class one. If so, the effective mode of this
user is the mode listed for him on the SPACL. If the
user is a special process in class two, Access Control
must next search for this user's explicit identification
on the ACL. If it is found, Access Control proceeds as
in the normal case, taking any traps that are indicated
unless this user has inhibited them, and returning an
effective mode taken from the ACL or the trap procedure.
If the explicit user's name is not found, Access Control
sets the effective mode equal to the standard mode given
for that user on the SPACL and returns.

Primitives

1.   appmode
2.   effmode

Both primitives are privileged to hard-core procedures,
i.e., callable from procedures in the hard-core ring only
and are used only by Directory Control.

1.   appmode

The primitive <u>appmode</u> returns the apparent mode of a given
user found in the access control information for a given
branch. This primitive is called by Directory Control
when it is listing the contents of a branch.

        call   appmode(ep,dp,amode,ring_brackets);

        <u>ep</u>: pointer to the branch (given)

        <u>dp</u>: pointer to the base of the directory
              containing the branch (given)

        <u>amode</u>: apparent mode of user with respect to the
               given branch represented by a string of
               five switches indicating trap (10000),
               read (01000), execute (00100), write (00010),
               append (00001) (returned)

        ring_brackets: A three element array of bit (6)
               each as described on page 2 (returned).

Method of Implementation

The user's name, consisting of three parts: the name,
the project identification and the instance tag, is picked
up from the user's process definitions segment.

Check if user is a special process, i.e., see if the user
is in the SPACL, if so proceed as described in the introduction,
otherwise, proceed as follows.

1st part scan: scan the names in the ACL of the branch
pointed to by ep comparing the first part of each name
in the ACL with the first part of the current user's name

> A match is found if the first part of the ACL name is
> "*" or is equal to the first part of the current user's
> name.
>
> if no match found repeat the 1st part of scan using the
> CACL of the directory. If no match is found using the
> CACL, then,
>
>> amode = "00000"b
>>
>> return to caller
>>
>> if match is found

2nd part scan: compare second part of ACL match with
second part of current user's name.

> if no match found continue 1st part scan.
>
> if match is found

3rd part scan: compare third part of ACL match with the
third part of the current user's name.

> if no match is found, then continue 1st part scan.
>
> if a match is found then,
>
>> amode = mode on ACL or CACL
>>
>> ring brackets (1), (2), (3) as on the ACL
>> or CACL. Return to caller.

The pointers to the SPACL, ACL and CACL are obtained respectively
as follows:

> spaclp = base$spacl_ptr;

This creates a pointer which points at a header with two
entries and is structured as follows:

        dcl 1 spacl_header based (spacl_ptr),

             2 no_entries fixed,

             2 spaclrp bit (18)

where:

        no_entries is the number of entries in the SPACL
and spaclrp is a relative pointer to the first entry in
the spacl. If no_entries > 0 then spaclp=ptr
(spacl_ptr,spacl_ptr spacl_header.spaclrp);

        aclp=ptr(dp,ep branch.aclrp);

and for the CACL

        caclp=ptr(dp,dp dir.caclrp);

where aclrp and caclrp are relative pointers to the first
entry in their respective linked lists.

2.   effmode

The primitive effmode returns the effective mode of a
given user evaluated from the access control information
for a given branch. It also checks if the retrieval trap,
system trap, and user trap are set for this branch and
if they are, executes these traps.

This primitive is called by Directory Control when it
is establishing a segment, helping Segment Control activate
a segment, or changing branch information that will subsequently
allow changes to the file.

        call   effmode (ep,dp,slot,opname,vacant,emode,
                            ring_brackets,errcode);

        where   ep,dp,and ring_brackets are as
                             defined in appmode.

slot: slot number of the branch pointed to by ep in the
            directory pointed to by dp (given)

opname: symbolic name of the Directory Control primitive
            which is calling effmode (given)

<u>vacant</u>:   a switch which, when off, indicates the mode returned
            is valid for this branch (returned)

            when on, indicates the branch was deleted during the
            time that a trap procedure was being executed
            (returned)

<u>emode</u>:   effective mode of the user with respect to the given
            branch represented by a string of switches indicating
            read, execute, write, append (returned)

Method of Implementation

Check if the current user is a special user, if so proceed
as described in the introduction, otherwise, use the same
procedure outlined in appmode to find a match for the
current user's name in the ACL or CACL.

    if match not found

        emode = 0

        return to caller

    if match found

Get mode associated with the match in the ACL

    if trap attribute is off

        emode = mode associated with match in ACL (i.e., amode)

        return to caller

    if trap attribute is on

        and if user has inhibited traps,

            then check for traps starting from 2. below

    if the user has not inhibited traps,

        then check for traps in the following order:

            1. User trap

            2. system trap

            3. retrieval trap

Before invoking any trap handling procedure, the date
and time the branch and ACL or CACL were last modified
must be saved and the branch must be unlocked. This is
done because the ACL or CACL may be modified during the
time the trap handler was operational and allows for a
quick check to determine this. The branch must be unlocked
because control may never return to effmode since the
trap handler may be going out of the hard-core ring.

In Initial MULTICS there will be no trap handling capabilities.
In subsequent versions the trap handlers will be implemented
to supply the user's id., the apparent mode of the user,
the branch attempted to be used, and the access brackets
to the trap procedure. The trap procedure will return
the effective mode of the user.

System traps will be superior to user traps so that the
effective mode of the user with respect to branch will
be that provided by the system trap handler if a system
trap exists.

Retrieval traps will have some form of communication with
the user if the expected time to retrieve is large.

Before calling any trap handler, the branch defined by
dp and slot must be unlocked (in case the trap procedure
fails to return control). When the trap procedure returns
control to Access Control, this branch is relocked before
the return to Directory Control. During the time that
the trap procedure is in control the branch is unlocked
making it possible for some other process to modify or
delete it. Thus after the branch is relocked, Access
Control checks the date-and-time-branch modified (dtbm)
item in the branch to see if any changes have been made
to the branch. If dtbm has been changed, Access Control
first checks the vacant entry switch and unique id to
see if the branch has been deleted. If it has been deleted,
Access Control sets vacant on and returns. If the branch
was not deleted, Access Control checks the ACL and/or
CACL to see if the user's apparent mode or the trap procedure
list has changed. If either of these are changed, the
effective mode is recomputed, otherwise the computed effective
mode stands.