To: Distribution

From: Jerry A. Stern

Date: 07/22/74

Subject: Additional Access Controls for Message Segments

Introduction

This MTB describes proposed changes to the message segment facility needed to enforce compliance with the "security" controls described in MTB-047. The reader is assumed to be familiar with the new terminology of the Access Isolation Mechanism (AIM) defined in MTB-100.

The Access Isolation Mechanism will prohibit the message segment facility, in its present form, from supporting system-wide queues for absentee and I/O requests in a multiple access class system. The principal difficulty, of course, is that a segment of access class X can only be written by processes of access authorization X. Hence, a single message segment can only serve processes of a single access authorization so long as message segments are bound by the standard AIM restrictions. Since the approach of providing one message segment per possible process access authorization for each system queue was judged to be impractical, it was decided to incorporate the message segment facility within the security kernel. This would make it possible to define individual messages as fundamental objects and to give the message segment facility the new responsibility of interpreting AIM rules as applied to messages. Originally, this approach would have meant moving the message segment facility into ring 0. For reasons explained in MTB-089, however, it was decided instead to extend the security kernel to include ring 1. Thus, the message segment facility will remain in ring 1, but will be given security kernel privileges and responsibilities.

Access Class of Message Segments

Despite the fact that message segments will contain messages of varying access classes, the storage system still requires that a single access class be assigned to the segment as a whole. If we view the segment access class as a label indicating the degree of protection required for a segment, then it is clear that the message segment access class should define the maximum access class of any contained information. As it turns out, it is not

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

only proper, but also convenient to have the message segment access class define an upper limit on access to a message segment. Therefore, a process will only be able to use message segments having access classes greater than or equal to the process access authorization.

In keeping with the standard storage system rules, a message segment can only be created in a directory of access class X by a process of access authorization X. This implies that the access class of the parent directory of a message segment must define a lower limit on access to the message segment. Otherwise, the act of creating a message segment would be visible to processes of lower access authorization than the creator and would therefore constitute a read-up path. Hence, in order to enforce this lower limit, a process will not be permitted to search a directory unless the process access authorization is greater than or equal to the directory access class. (1)

Combining the above two results, we see that access to a message segment is constrained to processes having access authorizations in the range between the parent directory access class and the message segment access class. The only problem with this scheme is that for many applications it requires that message segments be upgraded segments, i.e., segments having a higher access class than their parent directory. In general, upgraded segments are not supported by the storage system since, as discussed in MTB-047, they can be used as write-down paths. This is accomplished by manipulating the size of an upgraded segment and thereby changing the records used and current length stored in the branch and the quota used stored in superior directories. Message segments, however, are not directly writeable in the user ring. The size of a message segment can only be changed indirectly by means of a call to the message segment facility which executes within the security kernel. In this sense, upgraded message segments are analogous to upgraded directories. Both can be used, with some difficulty, as write-down paths. Therefore, our approach to the write-down problem of upgraded message segments will be similar to our approach to that same problem as it concerns upgraded directories. Whenever a message

---

(1) The need to restrict the searching of directories stems not only from the specific problem of message segments, but also from the more general problem of the KST being an information channel. It is a well-known flaw of the KST management scheme that by attempting to initiate inaccessible or non-existent segments, one can learn of the existence of directories which are not otherwise accessible. Since directories, and directory names, discovered in this manner could be used to pass information, it is necessary to prevent a process from making known a directory unless the process access authorization is greater than or equal to the directory access class.

segment is created, it will be upgraded (by ring 1) to the maximum access authorization of the creating process. Thus, the upper access limit of the message segment is identical to the upper access limit of the user who created it. Hence, a user cannot create a message segment to be used as a write-down path by a Trojan Horse program for information which the user is not already legitimately authorized to see. Of course, a Trojan Horse program could be designed to use a message segment innocently created by a higher-authorized user. A comprehensive solution to the general problem of upgraded segments and directories is known, but is unfortunately too ambitious to implement at the present time.

Access to message segments cannot be computed in the standard fashion. The message segment facility, executing in ring 1, must have essentially unrestricted access to message segments as it does now. Therefore, a new one-bit flag will be defined in the branch structure to identify certain ring 1 segments as "multiple access class" segments. For the time being, this flag will only be used for message segments and will be set by the message segment facility at the time a message segment is created. In the future, however, it may be used for other ring 1 segments as well.

The multiple access class flag will be checked by the access_mode procedure on each access computation. If the flag is on <u>and</u> the segment ring brackets are less than or equal to 1, then special action will be taken as follows. If the segment access class is greater than or equal to the process access authorization, then the computed mode of access will be the mode specified by the ACL. Otherwise, the mode will be null. Hence, a process will be able to initiate a message segment within ring 1 so long as the process access authorization is inferior or equal to the message segment access class (and superior or equal to the parent directory access class).

<u>Control of Message Segment Operations</u>

As described in MTB-100, the AIM rules dictate that a process can only read information of an access class inferior or equal to the process access authorization and can only write information of an access class equal to or greater than the process access authorization. Furthermore, the write-up operation should only be permitted where it cannot be used to destroy information. These rules can be applied in a straightforward manner to the control of message segment operations and will take precedence over extended access controls. The basic operations performed on messages consist of adding, reading, updating (not currently used) and deleting.

Whenever a message is added to a message segment, a new item of control information, called the sender access authorization, will be stored with the message to identify the access authorization of the process which added the message. A second new item of control information, the message access class, will be set equal to the sender access authorization by default. However, the sending process may optionally specify an access class greater than the sender access authorization (but not greater than the message segment access class). The purpose and use of this option is explained later.

In order for a process to read a message, the process access authorization must be greater than or equal to the message access class. If the process access authorization is less than the message access class, the message will be "invisible" to the process. This means, for example, that if a process requests to read the first message in a message segment, it will actually read the first message of an access class equal or inferior to the process access authorization. The message segment facility will deliberately skip over those messages which the process is not authorized to see. If a process requests to read a specific message which it is not authorized to see (by somehow guessing the message ID) it will be returned an error code indicating that no such message exists. The message access class and sender access authorization will be added to the return argument structure for all read calls so as to properly identify all messages.

Since the updating and deleting of messages represents a potential means of sabotage, a process will only be permitted to update or delete those messages for which the process access authorization is equal to the message access class. As in the case of reading, if a process requests to update or delete a message which it is not authorized to see, an error code will be returned indicating that the message does not exist. If a process requests to update or delete a message of inferior access class, a "moderr" will be returned since such write-down operations are prohibited.

In addition to the basic message operations discussed above, there also exists a primitive to return the message count for a specified message segment. Since the message count can be used as a communication path between processes, we cannot allow just any process to obtain the total message count. Therefore, a process requesting the message count for a given message segment will be returned the count of only those messages which it is able to read. This implies, of course, that at sites employing AIM controls, only those processes with system high access authorizations will be able to accurately determine the state of the absentee and I/O queues. For the sake of efficiency, the case of a process of access authorization X requesting the message count for a message segment of access class X will be optimized to simply extract the total message count from the header as is done now. In all other cases, however, all messages will have to be examined in order to determine the count of readable messages.

Another message segment operation is the reading and resetting of the salvage indicator. Whenever an internal inconsistency in a message segment is detected, the message segment is salvaged and a flag is set in the message segment header. Clearly, the setting of this flag is not user-controllable and resetting does not change its state unless the flag has been previously set. Therefore, the salvage indicator cannot be used to pass information and hence no additional restrictions will be placed on its use.

One last group of message segment operations (currently unused) deals with the reading and writing of header messages. A header message is a message of fixed maximum length contained in each message segment header. Header messages exist to provide a convenient protected storage space for subsystems employing message segments. At present, the only proposed use of header messages is to store an event channel ID, process ID, and a few other items for the secure send_message facility. The primitives which read and write header messages will not be directly callable from the user ring via a gate. Instead, the use of header messages will be restricted to subsystems executing within ring 1. It will be the responsibility of these subsystems to ensure the security of header message information. Currently there exists a single primitive to write all or part of a header message. This will be replaced by two primitives, one to write a new header message, and one to update an existing header message. The write primitive will completely erase any previous message and store the sender access authorization of the new message. The update primitive can be used to change all or part of an existing header message provided that the access authorization of the calling process matches the stored sender access authorization. The existing read primitive will be modified to return the sender access authorization of the header message so the calling program can determine its origin and accordingly, how the information should be protected. The specific use of header messages by the send_message facility is discussed later.

Although the above controls will be strictly enforced for all user processes, a mechanism will be provided to allow system processes unrestricted access to message segments through the standard interfaces. This is necessary to allow processes of all access authorizations to communicate with the Initializer and the I/O Coordinator via message segments. Stored along with the process access authorization (in the pds) will be a group of one-bit flags, each indicating a special system privilege which may be enabled for a system process. One of these flags will denote privileged access to ring 1 "system segments." In the course of computing access to a message, the message segment facility will check this flag and, if it is set on, it will allow whatever operation is requested.

Side-Effects of Message Segment Operations

Unfortunately, the controls described above do not prevent the passing of information via certain side-effects produced by message segment operations. Perhaps the most obvious problem of this nature is due to the fact the message segments have a finite maximum length. Therefore, a process can communicate a number to a lower-authorized process by filling up a message segment to a certain point. The lower-authorized process can learn the number, i.e., the amount of remaining space, by adding messages to the message segment until the overflow point is reached. No practical method has been devised for closing this information path. However, its use can be detected within the message segment facility by noticing each time a request to add a message is rejected for lack of free space. Whenever this situation occurs, it will be audited so as to discourage use of the write-down path.

Another side-effect having security implications concerns the structure of message IDs. Whenever a call is made to add a message to a message segment, the caller is returned a message ID composed of a unique bit string and the offset of the message within the message segment. The message ID can be used to later identify the message for reading, updating, or deleting. Unfortunately, it can also be used as an indirect communication path since it enables a process to detect when another process has added a message. In order to eliminate this potential write-down path, it is proposed that the message offset be removed from the message ID. Instead, the unique bit string will be used as a key for entry in a hash table stored in the message segment header. The hash table can then be used to locate messages by message ID. It is expected that hashing will be no less efficient than the current use of offsets due to the amount of checking required to verify that the offset portion of a message ID is, indeed, legitimate.

Conversion of Message Segments to a New Format

The addition of a hash table to the message segment header as well as the the addition of new per message control information will, of course, require a new message segment format. The message segment facility already provides an automatic mechanism to detect and convert message segments of obsolete formats into the latest format. This is done by simply comparing a version number stored in the message segment header to the latest version number stored in an external data base. If the message segment version is not the latest, then a conversion routine is invoked. Hence, a conversion routine must be provided to convert from the current version to the new version. The routine will have to construct a hash table and set the sender access authorization and access class of each message to the access class of the message segment.

The message segment salvaging procedure will have to be modified to properly handle the new items in the message segment format.

Message Segment Meters

The message segment facility keeps certain metering information in a segment called mseg_meter_data_. If this segment is to be written by processes of all access authorizations, then it must be made a multiple access class segment and should be upgraded to the system high access class. Another possibility is to simply eliminate message segment metering since the results are rarely, if ever, examined.

Currently, mseg_meter_data_ has ring brackets of 4,4,4. This is clearly a bug since it permits any user to clobber the segment. However, simply changing the ring brackets to 1,1,1 will prevent the use of the meter printing program. A new ring 1 program and controlling gate entry would be required to extract the metering data. Furthermore, access to this gate would have to be restricted since the metering data represents a potential inter-process communication path. Given the apparent lack of usefulness of message segment meters, it seems that any significant effort to preserve them would be unwarranted. Therefore, it is recommended that these meters be eliminated unless some use for them is known.

Implications for Queue Message Segments

Because queue message segments are only used to support communication with system processes, the impact of AIM controls on the use of queue message segments will be slight. All system queues will be upgraded to system high access class thereby permitting all processes to add messages to the system queues. The access class of queue messages will always equal the sender access authorization. Hence, a process will only be able to list or count those requests having an access class equal or inferior to the process access authorization. Also, a process will only be able to delete requests having an access class equal to the process access authorization. Daemon processes, of course, by virtue of system privileges, will be able to read and delete all messages. As discussed earlier, directories containing system queues can have a terminal quota and no status permission for ordinary users in order to close the write-down paths associated with upgraded segments.

Implications for Mailbox Message Segments

This section describes the impact of AIM controls on the proposed use of message segments to implement a secure mail and send_message facility. (See MTB-070 and MTB-085). Because mailboxes are used for inter-user communication rather than user-daemon communication, the effect of AIM controls on the use of mailboxes is more severe than on the use of system queues.

The use of mailboxes for mail communication (as opposed to send_message communication) will be identical to the use of queue message segments from the standpoint of AIM controls. The access class of mail messages will always equal the sender access authorization. This implies that a process which receives mail from a process of lower access authorization will be able to read the mail, but not delete it. In fact, it will not even be possible to mark such mail as "has been read" since that would constitute a write down. Therefore, the receiver may have to see the mail repeatedly until he finally logs in at an access authorization equal to the mail classiciation at which time the mail can be deleted. Although this problem of lingering mail is an inconvenience, it is at least preferable to the alternative of no mail reception at all from processes of lower access authorization. Furthermore, the inconvenience can be minimized by use of a special mail command. For example, a speical mail command could pause after printing the first line of each message and await a user instruction to either continue printing the message or to skip to the next message.

The use of mailboxes for send_message communication will be somewhat different from mail communication. If a process is accepting messages, the header message of the mailbox for that process will contain an event channel ID, a process ID, and switches indicating whether normal and/or urgent wakeups should be allowed. The sender access authorization for the header message will identify the current access authorization of the process accepting messages. When an attempt is made to send a message, the header message of the target mailbox will be examined. In the case where the target process access authorization equals the sending process access authorization, send_message will operate in the standard fashion. The access class of the message transmitted will equal the sender access authorization. When the sending process access authorization is greater than the target process access authorization, it will not be possible to send a wakeup. In this situation, the user should be asked if he still wishes to send the message. If so, the message will be given an access class equal to the sender access authorization. The target user will not receive the message until he next logs in at a sufficiently high access authorization. The most interesting case occurs when the target process access authorization is greater than the sending process access authorization. In this case, a wakeup can be sent. In order to allow the target process to delete the message after reading it, the access class of the message will be upgraded by the sending process to the access authorization of the target process. (The interface for adding an upgraded message to a message segment will be internal to ring 1.) Unfortunately, the sending process cannot be informed as to whether or not the target process is deferring messages since this would constitute a read up operation. Similarly, the sending process cannot receive an acknowledgement from the target process. However, the sending process can be informed of whether or not the target process is accepting messages at all. Technically, this too is a read up. But since there is no way to "unaccept messages," the act of accepting messages can only be used to pass one bit of information during the lifetime of a process which is not enough to worry about.

Detailed List of Changes

1. Provide a new ring 0 primitive to set the multiple access class flag. This primitive will only be callable from ring 1 via admin_gate_.

2. Change access_mode to check the multiple access class flag as described.

3. Change find_ to not make known (and hence not search) a directory unless the process access authorization is greater than or equal to the directory access class. This check will be performed by a call to the new AIM access checking module.

4. Change mseg_add_ to store the sender access authorization and access class of each message and to maintain a hash table. Provide a new entry point to add upgraded messages. Audit attempts to add a message which are rejected for lack of space.

5. Change mseg_util_ to perform the proper AIM check for read, update and delete operations and to use the hash table. Add a new entry to get the count of readable messages. Return the sender access authorization and access class of each message read by expanding the return argument structure.

6. Change mseg_ to provide the new header message entries and a new entry to add upgraded messages.

7. Change queue_mseg_ to provide a new entry to add upgraded messages.

8. Provide a new mseg_convert_ procedure for the new message

segment format.

9. Modify ms_salvager_ and ms_salv_util_ to handle the new format.

10. Change the include file mseg_return_args_.incl.pl1 to contain a declaration for the message access class and the sender access authorization. Change the following I/O Daemon programs which use the include file (if necessary): output_request_, find_next_request_, free_oldest_request_, iodc_, and save_request_. Aslo change the following modules which contain their own declarations for the mseg_return_args structure: cancel_abs_request, lar_util_, and absentee_utility_.

11. Make the necessary changes to the forthcoming secure mail and send_message facility as described.