To: Distribution

From: D. H. Hunt

Subject: Implementing Security Enhancements to the Interprocess Communication Facility

Date: 05/02/74

Overview

This memo is one in a series of design proposals describing enhancements to Multics access control, as outlined in MTB-47. The effect of the enhancements is to control information paths between processes, in order to prevent compromise of information. In addition to shared segments, which are the primary information paths that Multics provides, there are secondary information paths between processes. One of these is the Interprocess Communication (IPC) facility. The IPC facility represents an information path of sufficient bandwidth that it must be controlled to prevent the compromise of information. The requirement is that each message must be validated according to the clearance of the sender and the receiver. A mechanism to support this requirement has been designed, and is described here.

It is worth pointing out that even if an IPC wakeup is sent without the optional "event message," information is still transmitted between processes. A wakeup is a message with one bit of information. In this MTB, "IPC wakeup" is used to describe any interprocess communication transmitted by the IPC facility, whether or not there happens to be an associated event message.

In order to provide the proper background for this proposal, the current ring 0 IPC mechanism is reviewed here. The initial description is of the basic ring 0 IPC mechanism and does not cover "special" event channels. Following the initial description, those aspects of special event channels relevant to this design will be explained.

Review of Ring 0 IPC

Since the sending side of IPC is simpler, it is described first. On the sending side, there is no "front end" to the IPC facility which is in the user ring; accordingly, the sending process calls the ring 0 entry hcs_$wakeup. There are three input arguments to hcs_$wakeup: a target process id, an event channel, and an event message; there is also one output argument, which is a status code. The gate transfers control to the ring 0 procedure "hc_ipc" which performs a few checks on the format of the event channel argument and then calls a traffic controller

entry, pxss$wakeup, with the same arguments. The traffic controller is invoked because it manages the Interprocess Transmission Table (ITT), which is the system-wide data base used to store and forward IPC wakeups. For each process with pending wakeups, there is an associated threaded list, containing these wakeups, in the ITT. Each APT entry contains a relative pointer to the head of its associated ITT list. If a process has no pending wakeups, then there is no list of entries allocated for it in the ITT and the relative pointer to the head of its ITT list is null.

The sending process, while executing in the traffic controller, locates the APT entry of the target process, allocates an entry in the associated ITT list, and stores the event channel, event message, and information about the sender into the new ITT entry. After associating the wakeup with the target process, the traffic controller makes sure that the target process is scheduled for execution, in order that it may act upon the wakeup. Control is then returned to the user ring.

The receiving side of IPC, which is a bit more complicated, includes an interface in the user ring. The user ring IPC procedures manage the event channels, channel priorities, and the event wait and event call facility, together with the Event Channel Table (ECT). The user ring interface is ipc_$block, which has a "wait list" as an input argument and a "wait message" and status code as output arguments. A call to ipc_$block may result in a call to one of two ring 0 entries: hcs_$read_events or hcs_$fblock. These entries have identical interfaces; the only difference is that calling the latter one may result in giving away the processor. A call to one of these entries may in turn result in a call to one of two traffic controller entry points, pxss$get_event or pxss$block. Again, these two differ only in that the latter may give away the processor; the interface is the same. They both return a pointer to the head of the threaded list of wakeups stored in the ITT, and reset the head-of-ITT-list pointer to null. If the returned ITT thread is non-empty, then, based upon the value of a 3-bit ring field in the event channel name, the ring 0 procedure hc_ipc dispatches wakeups to the corresponding rings. This is accomplished by copying the contents of an ITT entry into a per-ring buffer associated with the ECT. After all wakeups have been dispatched to the proper ECT buffers, ring 0 IPC calls pxss once again to free the ITT threaded list and then returns to the user ring.

Special event channels were added to the IPC mechanism in order to handle certain IPC wakeups more efficiently. They are intended to be used to transmit "device wakeups" -- i.e. wakeups originating in ring 0 as a consequence of an interrupt from some device. Special event channels have no associated event message nor information about the sending process. They are distinguished by a particular pattern in the 72-bit event channel name. There is no restriction on creating special event

channels; however the ring 0 procedure hc_ipc prevents any process from sending a wakeup to another over a special channel. A process may send a wakeup to itself over a special channel. Since wakeups originating in ring 0 are sent by calling pxss directly, only ring 0 wakeups and wakeups sent from a process to itself can make use of special channels. The implementation of special channels bypasses the ITT (as well as the EOT). In each APT entry, there is one bit per special event channel. Calling pxss$wakeup with a special channel as an argument will set the bit corresponding to that special channel; calling pxss$get_event or pxss$block will copy all special channel bits into an output argument and reset the bits in the APT entry.

The Impact of Security Controls on IPC

The security requirements, as described in MTB 47, impose controls on information paths between processes. Clearances are associated with subjects (e.g., processes), and classifications are associated with objects which need to be protected (e.g., segments and directories). The access rules to be enforced by the system restrict information paths by preventing "read-up" and "write-down" operations. Since IPC is an information path, it is affected by the security requirements. The IPC "send" operation corresponds to "write" and the "receive" operation corresponds to "read." The security rules as applied to IPC should prohibit "send-down" operations, and log any "send-down" attempts in the system audit file. (1) A process would be allowed to receive IPC wakeups from any process of equal or lower clearance, and send to any process of equal or higher clearance. Note that although the write-up operation is not allowed for segments due to the possibility of sabotage, send-up is permissible for IPC because the exchange of information is totally structured by ring 0 and is therefore immune to sabotage.

In order that the system continue to operate properly, however, the restrictions described above must not apply to every IPC wakeup. There are two classes of IPC wakeups which must not be subject to the security constraints.

1. The first class consists of the device wakeups which originate in ring 0 of some arbitrary process. (2) A

---

(1) Another MTB in this series describes the auditing mechanism in detail.

(2) Currently, the only other wakeups which originate in ring 0 are those resulting from (a) receiving a "quit" from the user's terminal, or (b) the expiration of a per-process timer. Wakeups of the first sort can be grouped with the device wakeups, and wakeups of the second sort can be excluded from the security restrictions since the sender process is also the receiver process. The proposed design special-cases all ring 0 wakeups.

device interrupts an arbitrary process, which in turn sends a wakeup to the process waiting for current status from that device. The process which maps the interrupt into the wakeup should be considered an anonymous intermediary; its clearance should be of no consequence. If the rules were to be applied in this case, a process could not send a device wakeup to another process with lower clearance.

2. The second class of wakeups which must not be restricted consists of wakeups sent and received by those system processes, such as the initializer, which must maintain two-way communication with all other processes. If the rules were enforced in this case, the initializer would be able to carry on two-way IPC communication only with other processes of the same clearance.

The security access controls, accordingly, should be applied to all wakeups not originating in ring 0, except those sent or received by designated system processes.

Comparison of Alternative Implementations

In order to accommodate the new security controls, only the ring 0 IPC mechanism must be modified. The user-ring IPC, as well as all external and internal interfaces (with the possible addition of a new value for a returned status code) will remain unchanged. The ring 0 modification will consist of code which compares the clearance of the sending process with the clearance of the receiving process, and based upon that comparison, either permits or prevents the sending of an IPC wakeup. There are two reasonable strategies for implementing the clearance checks: checking while in the sender's process, or checking while in the receiver's process.

A sending-side check requires that the sender process be able to obtain the clearance of the target process. Accordingly, the choice for storing the clearance of the target process is the APT entry. The sending process will compare clearances while in the wakeup entry of the traffic controller.

A receiving-side check requires that the receiver be able to obtain the sender's clearance. For this approach, it is sufficient to include the clearance of the sending process in each associated ITT entry. Ring 0 IPC, while copying entries out of the ITT, would compare the clearance stored in each ITT entry with that of the executing (target) process.

The recommended approach to implementing the IPC security controls is to perform the checking on the sending side. There are a number of arguments in favor of this approach.

1. Checking on the sending side eliminates "false alarm" wakeups: if a process attempts to send a wakeup to another process of lower clearance, the access violation will be detected in the sending process. The primary advantage of this is that the receiving process will not be awakened unnecessarily, merely to discover that the wakeup cannot be received. A secondary advantage is that it is possible to return an error status to the caller in the sending process.

2. The send-side check allows all non-ring-0 wakeups to be checked. Only wakeups which use the ITT can be checked on the receiving side. Currently, there is a restriction that wakeups (to other processes) over special channels must originate in ring 0. If this restriction is removed, then the send-side check will still be effective, whereas the receive-side check will not.

3. Auditing any attempt to send down is more easily accomplished if clearances are checked on the sending side. The "process group id," which is used by the audit trail procedures to identify a process, is stored in the pds. Since the process group id of the sending process is not readily available to the receiving process, it is preferable to generate audit messages while executing in the process which caused the infraction.

4. The send-side check is more economical in terms of wired-down storage. The clearance information is expected to occupy two words: one word for the categories, and the other for the level and additional control information. Therefore, the alternatives are to allocate the two words of clearance information either in the APT entry or in the ITT entry. In typical Multics system configurations, there are about twice as many ITT entries allocated as APT entries. Thus more wired storage would be required for the receive-side method. In fact, although the ITT would need to be expanded to hold the clearance information, it is possible to reclaim space in the APT entry for this purpose. The 2-word space for "x_paging_measure" is obsolete and will be made available for storing the clearance information.

There are, of course, some disadvantages to the proposed approach. Send-side checking requires that the APT remain locked for a longer period of time. Roughly a dozen ALM instructions are required to perform the clearance check, which must be done while the APT is locked. The receive-side approach requires only two additional ALM instructions, which store the sender's clearance into the ITT. A method of reducing the number of instructions which must be executed on the sending side is described in the section on performance. Another disadvantage to the send-side approach is that the clearance of a process must be

stored in two places: not only in the PDS, but in the APT entry as well. Although it is possible to employ a receive-side check, the arguments above favor the send-side check.

Implementation Details

The send-side implementation is now described in detail, with the aid of a flowchart. The data bases affected are the PDS and the APT entry; the only procedure affected is the "wakeup" entry to pkss. The current flow of control in the wakeup entry is depicted in Figure I. The dotted lines indicate the point where the clearance-checking code should be inserted. It should be emphasized that, among various alternatives, this proposed modification has the least effect upon the current structure of the code in the wakeup entry. In this code sequence the "ring 0 wakeup" flag is set to true only for device wakeups; all other wakeups are checked for the clearance of the sender and receiver. The new code sequence will compare the clearance in the PDS (that of the sender) with the clearance in the APT entry which has been located (that of the receiver). As can be seen, if special channels are ever used for non-ring-0 wakeups, they will also be checked by this code. The new code will allow a wakeup to be sent, regardless of the clearance comparison, if either the sender or receiver has the exception bit set.

The 2-word field in the APT entry which is to hold the clearance information will have the following structure.

```
2 security,
  3 categories          bit(36),
  3 level               fixed bin(17) unaligned,
  3 exceptions          unaligned,
    4 segments          bit(1),
    4 directories       bit(1),
    4 ipc               bit(1),
    4 pad               bit(15)
```

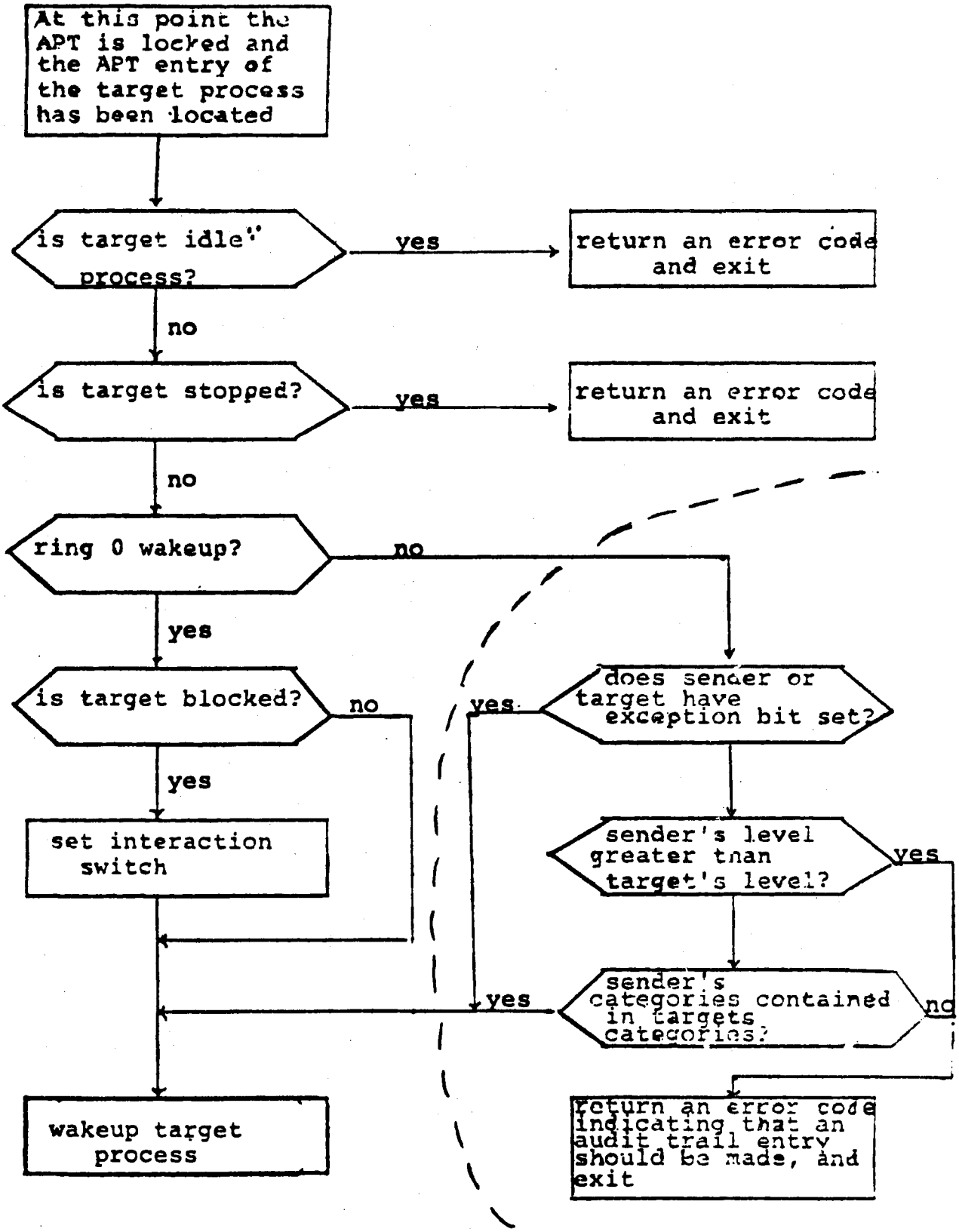
At this point the APT is locked and the APT entry of the target process has been located
is target idle process?
yes
return an error code and exit
no
is target stopped?
yes
return an error code and exit
no
ring 0 wakeup?
no
yes
is target blocked?
no
yes
does sender or target have exception bit set?
yes
set interaction switch
sender's level greater than target's level?
yes
sender's categories contained in targets categories?
yes
no
wakeup target process
return an error code indicating that an audit trail entry should be made, and exit

Figure 1

Performance Considerations

The aim of this rather qualitative discussion of performance is to show that the proposed changes will have only a very slight effect. The only IPC path which is affected is the infrequently-used one which sends user-ring wakeups. The performance cost in this case can be divided into the additional wired storage and the additional execution time.

The wired storage cost is low, since an obsolete 2-word field in the APT entry can be reused to store the security control information. Moreover, the actual instruction sequence which compares the clearances will require only about twelve additional words of wired storage.

The additional execution time should have only a slight degrading effect. Since the proposed instructions are to be added within the scope of a global lock, the effect of multiprocessor interference should also be considered. Today, there are about 150 ALM instructions executed, in pxss, for each user-ring wakeup. Essentially all of these instructions are executed while the APT lock is locked. Twelve additional instructions amounts to an 8% increase in the length of time that the APT is locked (assuming that variation in instruction times can be neglected). The relative time increase for user-ring calls to hcs_$wakeup will be less than 8% since this path includes hcs_ and hc_ipc as well as pxss.

The effect of increasing, in some cases, the time that the APT remains locked is not expected to be significant. In the two-processor MIT system, the amount of time that any one processor spends looping on the APT lock because the other has it locked is very small (on the order of a few seconds a day). As a consequence, increasing the lock interval for user-ring wakeups by 8% is not expected to have noticeable effect on the looping time.

It should be re-emphasized that these estimates characterize an upper bound on performance degradation. The path with the heaviest traffic (ring 0 wakeups) does not include the security checks.

One offsetting performance improvement which ought to be suggested is to streamline the code in the pxss$wakeup entry. Without much effort, half a dozen instructions could be eliminated merely by eliminating redundant tests. This change would cut the cost introduced by the security checks in half, in the case of user-ring wakeups. In the case of ring 0 wakeups, the performance would be improved. Hence, an overall improvement in performance would be anticipated.