PROJECT MAC | February 7, 1973

Computer Systems Research Division | Request for Comments No.5

REPAIRED SECURITY BUGS IN MULTICS

by J. H. Saltzer

A short time ago I began to compile a list of all known ways in which a user may break down or circumvent the protection mechanisms of Multics. The list is quite interesting, and available for individual study, but until the problems are repaired, it does not seem wise to distribute it widely. On the other hand, it would be wise to promote discussion of the topic, so as problems are fixed, I will publish their descriptions.

Examining post-mortems of fixed bugs may initially strike one as unrewarding, but there are some potential payoffs. Since one of our objectives is to discover how to construct a simple, auditable supervisor which has a very low probability of such errors, the following questions seem worthy of discussion about each bug:

1. How did it get in to the system? What design decisions helped create an environment in which the error was made?
2. Why was it not detected immediately, at checkout time or during system installation? What better auditing tools might have resulted in earlier detection?
3. Was a design principle violated, thereby leading to the error?
4. Is this bug a member of a class of errors, of which there may be more examples in Multics? What design principle or auditing technique might be useful in eliminating all such related errors?

By way of definition, let us use the following arbitrary definition of security-related problems: those which permit

1) unauthorized disclosure of information.

2) unauthorized changing of information.

3) denial of accessibility to authorized users.

A bug which may be exploited to force a system crash is considered to be in the third category. To constrain our area of concern, only security-related problems which are part of operating system design or implementation are

This note is an informal working paper of the Project MAC Computer Systems Research Division. It should not be reproduced without the author's permission, and it should not be referenced in other publications.

of interest. For example, the practice of leaving the door to the machine room unlocked is not of interest to us. (Unless the cause is a bad design feature of the operating system which prevents convenient system operation inside locked doors.)

Recently Repaired Security Bugs

Several problems were fixed in the installation of system 18.0 which simplified the access control strategy of the system:

1. The CACL ring brackets trap. Before system 18.0, every ACL and CACL entry contained its own separate ring bracket specification, leading to great ease in slipping up, especially if one creates a segment in a strange directory without first checking its CACL. This trap was fallen into by the linker in the following way: if a user in ring 4 called a ring 1 entry for the first time, the linker tried to create a new combined linkage section for ring 1 in the process directory. If the user had previously planted a link with the name "combined_linkage_1.01" in his process directory, the combined linkage segment would actually be created wherever he wished -- in some other directory, for example. Although the linker carefully set the ACL of the new segment to permit ring-one access only, the CACL of the target directory could give access in higher rings to other users.

   Since 18.0 fixed this problem by making the ring bracket specification a property of the segment, as specified by the creator, rather than a property of the individual ACL or CACL entry.

   It should be noted that a contribution to this trap was made by the automatic system feature of allowing segments to be created through links. It would perhaps make sense to allow protected subsystems to specify that they do not want this feature, so that when they create a segment by name, it is created exactly where they expect.

   Security Principle: If the protection status of a segment depends on its position in the naming hierarchy, the creator of a segment nust be given complete control of that position; no one else may be allowed to influence its position.

This principle is currently at odds with two system deficiencies, both of which lead to desire to put links in the process directory:

a) an inflexible process directory record quota scheme, which leads to the need to place some system segments in other directories.

b) the automatic discarding of a process directory contents upon accidental process termination, which leads to a need to place some system segments elsewhere so that they may be examined to discover the reason for the process termination.

It seems quite clear that solutions to these two practical problems must be found before the basic security principle can be followed.

2. AST overflow bug. Before system 18.0 was installed, there was a requirement that whenever a segment is active, all directories superior to the segment must also be active. If a user created a directory tree deeper than the AST size, he could overflow the AST with unremovable entries. This would cause a system crash.

Although this method of systematically crashing the system has now been fixed by 18.0, which does not require that superior directories be active, it illustrates another unfollowed security principle: table overflows and other unexpected (impossible) events must be handled gracefully without crashing the system, since the assumption that the overflow (or whatever) cannot be systematically produced by an attacker is hard to verify; worse, a system change elsewhere later may render the assumption incorrect.

3. Blank names bug. If a directory contained an entry for a segment with an all-blank name, deletion of that directory would cause a system crash. System 18.0 fixed this bug, which again was based on assumption that the user could not force an impossible condition to occur, so no recovery for the impossible condition was provided.

4. fs_get bug. Entry fs_get$ref name failed to initialize its error handler, so when it got an error return from kst_man (e.g., KST has overflowed) it attempted to reset a lock it never set, crashing the system. This one seems to be a simple programming error, since setting up the error handler fixed the trouble. Some technique of auditing which detects this class of bug is needed.

One other bug has been recently fixed, in system 17.11:

5. Argument validation bug. The software validation of arguments on cross-ring calls permitted pointers with indirect modifiers to be used, but it did not follow the indirect chain to see where it led. A user could supply an indirect argument pointer in a call to a supervisor entry which writes into an argument, and thrreby redirect the writing back into a supervisor database. This bug was fixed by changing the software validation to forbid indirect modifiers in argument pointers.

   This bug has some aspects similar to those of bug number 1, above, in that unexpected indirection can easily be overlooked.

   This bug would have been automatically fixed by the 6180 argument validation hardware, which will also automatically take care of about 30 other argument address validation troubles which have been uncovered by systematically auditing the supervisor entries.