PROJECT MAC January 28, 1974

Computer Systems Research Division Request for Comments No. 47

SOME RECENTLY REPAIRED SECURITY HOLES OF MULTICS

by J. H. Saltzer and D. Hunt

This note is the third of a series* which describes design and implementation errors in Multics which affect its ability to protect information and provide service. The purpose of the series is to try to discuss what incorrectly laid groundwork permitted each trouble to creep in.

It is interesting (and comforting) to note that no security problem yet discovered has required any change in the original overall design of Multics; the problems have universally been at the level of detailed design errors or implementation slipups; the repairs have been conceptually simple readjustments to bring the design or implementation back to the originally intended one.

Reused address

Following a system crash, the salvager may discover that a single disk or drum page is being used by two or more page tables, a situation which should never occur intentionally, but may appear if a crash occurs while updating a page table value. In the original design, the page in question was awarded to the first page table encountered by the salvager, and later users of that page were assigned new pages containing zeroes. Since there is no way to tell which of the multiple users was the legitimate one, the present, safer design gives all users of a reused page distinct pages of zeroes. This improved design helps reduce the chance of one user seeing another user's data because of a system crash. Ideally, one would make the storage space which holds a page larger than the page itself, and store a copy of the segment unique identifier with each page when it is assigned to a segment. Then, since pages are identifiable, lost or multiply-used pages could be returned to their proper owners with less chance of accidental interchange.

This problem illustrates an issue which is as yet not very systematically approached in large systems: the initial design almost always assumes perfectly functioning hardware and software, and as experience is gained about which failures are most common, patches are added to protect. The design of the

* Previously issued: RFC-5 and RFC-46.

second CTSS file system included forward and backward pointers with every record of a file; the system always checked the back pointers to see that they contained the expected values. As a result, parts of user files were almost never interchanged -- a distinct improvement over the first CTSS file system which used forward pointers alone, and in which it was a common occurrence to find someone else's data in your file. Unfortunately, this particular CTSS lesson did not get transferred to Multics, probably because of the extra overhead that might have been involved in drum management.

Operator login window

When bootloading Multics, the operator dialed a telephone number to log in the "initializer" console, which controls all system operation. A hostile user, with careful timing, could dial the number and take over the system as it comes up. The design was adopted so that system initialization could be performed from any available terminal; it was originally intended that the operator supply a password, but for some reason that intent was never implemented. The design was recently changed to permit use of a terminal which is permanently wired to the system; security is higher, but when that terminal breaks, system operation may be awkward. The awkwardness can be eliminated by having several available hardwired terminals.

FSDCT update problem

The "file system device configuration table' (FSDCT) contains a bit for every storage block in every secondary storage device. A "one" means the block is unused, a "zero" means it is used. If several devices become completely used, a page of the FSDCT may become filled with zeroes. Since it is an important table, it is frequently backed up by copying it out to secondary storage. The procedure invoked for this copying is the standard page removal procedure, which has been designed to discard pages of zeroes rather than writing them out. The routines which read the FSDCT from secondary storage at system initialization time (before the standard paging program works) was a non-standard one which did not know that pages of zeroes were given special treatment; a system crash resulted whenever the system was initialized. In principle, at least, a user with a very large storage allotment could exploit this bug by creating many segments just before a system shutdown. The system would shut down with an FSDCT containing blank pages, and all future attempts to bootload the system would fail. The bug was fixed by revising the FSDCT reading procedure to correctly recognize the blank pages during initialization.

This is a category of bug which does not permit the exploiter to read information, but merely to deny use of the system to other legitimate users. The particular problem illustrates the effect of first using a special trick for efficiency, followed by later use of an old procedure for a new purpose without reviewing its operation for special tricks.

<u>Login table overflow</u>

The list of logins during a single bootload of Multics was stored in a single segment with no overflow procedure. A single user, by logging in several thousand times, could overflow the segment, making further logins by authorized users impossible.

This is another example of a "denial-of-use" bug, but one which could be rapidly recovered from by reinitializing the system. Its origin lies in the period between 1968 and 1970 when a combination of pressure to get going and also a short average "system up" time made programmed provisions for table overflow look like a non-essential luxury. It has been long since fixed by adding an overflow procedure, but its origin is instructive since there may be yet unsuspected protection bugs with the same origin.

<u>Page control magic number</u>

An old hardware bug trap places magic numbers in core where a page is to be read in, then after reading the page checks the numbers. If still there, it assumes the page didn't come in, and reports a page read error to the user. If a user places contrived names containing the magic bit patterns strategically in a directory to which he has only append access, he can effectively delete other entries in the directory.

The trap has been left in the system, but it has been placed under strict operations control by requiring a special "debug" card in the configuration deck loaded by the system operator before lootload; operation with the debug card in place is done only with special authorization, and leaves an audit trail.

Retriever acl-setting bug

The retriever, used to obtain old copies of files from backup tapes, used to work as follows:

1. Create a new empty segment in the user's directory, with an access-control-list permitting access to anyone.
2. Copy the data from the tape into the new segment.
3. Read the appropriate access-control-list from the tape.
4. Replace the initial access-control-list with the one read from the tape.

If an error of any kind occurred after completion of step 2, the retriever would exit, leaving the data reloaded but unprotected; the user received no warning of the condition. As a result, an explorer of the directory hierarchy would typically discover several files to which he had access but should not have.

The problem was repaired by making the initial access-control-list grant access to the retriever process only; any errors after that point result in a fail-safe inaccessibility of the segment. Since the user who requested the retrieval will usually try to immediately use his retrieved segment, its inaccessibility will tend to be discovered quickly, and a locksmith can be called upon to adjust the situation.

This problem is a good example of design which did not take into account all the implications of an error encountered in an otherwise acceptable sequence.

Process directory record overflow

If the user generates too much storage (more than 500 pages) in his process directory, an error is signalled to him. In the original design, the signaller used the wrong stack, crashing the system. This bug could be exploited to deny service to others at the user's whim. It was repaired by having the signaller use the correct stack. It is a good example of the effect of complexity (the need for several possible stacks) compounded with the difficulty of testing unused and limit conditions. Basically, the handlers for rare and unusual conditions tend to be poorly tested simply because normal use, which uncovers most bugs in today's systems, does not exercise them.

Locked stack base problem

In the design of the 645, a provision was made for the supervisor to lock the value of any base register. This feature was included primarily because it was planned to handle faults and interrupts using a stack, and it was not certain at the time whether or not use of a stack was possible unless the stack base register (containing the stack segment number) was locked against user tampering. For several years, Multics operated with a locked stack base register whose value was changed by a master-mode procedure as part of the ring-switching operation.

The fault and interrupt interceptors were coded assuming a locked stack base at three points, although after the ring design was complete, it became clear that the user could, in principle, be safely allowed to modify the stack base register.

With the evolution of the design of the PL/I compiler, it became apparent that the extra flexibility of allowing the stack base register to be user changeable was quite handy, so the stack base register was unlocked. Unfortunately, no one followed through with the three one-line changes to the fault and interrupt interceptors required to eliminate their dependence on a locked stack base register. As a result, one could load the stack base register with the segment numbers of one of the ring-zero stacks, and then wait for the next fault or interrupt, which would go to an interceptor which incorrectly assumed that because the stack base register had the expected value, the stack pointer register must also be loaded correctly. The result was possible overwriting of a ring zero data storage area at the direction of the user.

The problem was fixed by adding the three one-line checks mentioned. The underlying trouble here seems to be a failure to follow through all the implications of a change in a fundamental ground rule; clearly such changes are dangerous and must be approached with all possible caution. (see also RFC-46, discussion of user-ring master-mode procedures.)

New ring stack bug

The system has an internal procedure, named "append_branch", which creates a new segment, and a utility named "makeseg" which either creates a new segment (by calling "append_branch") or returns a pointer to an old one if it already exists. Since "append_branch" requires many arguments to describe the newly created segment, and "makeseg" supplies useful defaults for most of the arguments, there is a tendency among system programmers to call "makeseg" rather than "append_branch", even when use of an old segment would be incorrect. In the case of the procedure which creates stacks for newly entered rings, the user could create a segment with the stack name of a previously unused inner ring, but with ring brackets allowing him to read and write the stack contents. Then, upon calling a procedure in the inner ring, stack creation would be automatically triggered. The stack creating program called "makeseg", and thus would receive a pointer to the previously planted stack rather than an indication of an error. The inner ring procedure would then proceed, oblivious to the fact that its stack was then accessible to programs in outer rings.

The problem was fixed in moving to the 6180, since the stack creation strategy had to be modified anyway; procedure append_branch is now used. We have here an example of how a particular combination of too many conveniences in one utility program can lead to sloppy consideration of the implications of using it.