PROJECT MAC January 28, 1974

Computer Systems Research Division Request for Comments No. 46

SOME MULTICS SECURITY HOLES WHICH WERE CLOSED BY 6180 HARDWARE

by J. H. Saltzer, Phillippe Janson, and Douglas Hunt

This note is the second 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.

A fairly large number of security problems were fixed automatically by conversion from the Honeywell 645 to the Honeywell 6180, which has built-in argument validation hardware. As will be seen, replacement of a complex software package with a relatively simple hardware mechanism was remarkably effective, suggesting that it was a move in the right direction.

Unvalidated Gates

In the 645, the following gates to ring zero had no validation of arguments at all:

absentee_test_	(all entries)
hphcs_	(all entries)
phcs_	(all entries)
phnxhcs_	(all entries)
admin_gate_$guaranteed_eligibility_off	
admin_gate_$guaranteed_eligibility_on	

Argument validation consists of checking each argument to a gate entry to be sure it refers to an address to which the caller is permitted access. For example, if the ring zero program intends to write into the argument (e.g., an output value) then the caller of the entry should specify an address in which he is permitted to write. Failure to perform argument validation would mean that the caller could specify an address somewhere inside ring zero; if he did, the ring zero program could be used for unauthorized patching of the supervisor. It is slightly harder but still possible to exploit a gate which only reads its arguments.

---

* Previously issued memo in the series: RFC-5.

The unvalidated gates had one thing in common: they were all controlled by access control lists which limit their use to supposedly responsible individuals. This control was probably the chief rationalization for not putting in the extra effort required to specify the argument validation.

On the 6180, all arguments are automatically validated by hardware checks on the ring of origin of every argument. This approach eliminates both the extra (and sometimes neglected) effort needed to specify validation, and also any possibility of errors in that specification.

Incorrectly validated arguments

In the following entries, some argument was validated with more leniency than appropriate, permitting the user, typically, to cause the supervisor to write into an area in which the user has no access.

hcs_$get_seg_count	last argument unvalidated.
hcs_$get_entry_name	argument validated for wrong type.
hcs_$get_dbrs	argument validated for wrong usage.
hcs_$assign_channel	1st argument validated for wrong usage.
hcs_$check_device	2nd argument validated for wrong usage.
hcs_$get_search rule	argument validated for wrong usage.
hcs_$get_count_linkage	2nd argument validated for wrong usage.
hcs_$ipc_init	argument valudated for wrong usage.
hcs_$list_dir	2nd argument validated for wrong usage.
hcs_$make_ptr	1st argument validated for wrong usage.
hcs_$list_dir_acl	3rd argument validated for wrong usage.
hcs_$set_dtd	3rd argument validated for wrong usage.
hcs_$status	entire argument spec is wrong.
imp_dim_gate_$imp_read_order	3rd argument validated for wrong usage.
imp_dim_gate_$imp_write_order	3rd argument validated for wrong usage.
netp_$ncp_priv_status	3rd argument validated for wrong usage.
netp_$ncp_priv_order	3rd argument validated for wrong usage.
net_$ncp_status	3rd argument validated for wrong usage.
net_$ncp_order	3rd argument validated for wrong usage.
hcs_$acl_list	5th argument validated for wrong usage.

This list represents the accumulation of errors over several years of specifying argument validation for about 150 user-callable gates. When an argument is validated for "wrong usage" it typically means that the gate specification says that the gate only reads the argument, when the gate actually writes into it. Thus, the validator checks only to make sure that the user can read data at the specified address. If the user provides a pointer, say, to some location in the "sys_info" segment, in which he has read-only permission, the gate, which can write into "sys_info" by virtue of its ring-zero location, would then overwrite some item there.

Again, the value of the automatic hardware argument validation feature of the 6180 is clear: the opportunity for an incorrect software-declared specification is completely eliminated.

Unvalidatable arguments

In the following entries, some entry could not be checked by the automatic validator, since the correct method of validation depends on the value of some other argument.

hcs_$acl_list	3rd argument used as both input and output.
hcs_$ex_acl_list	3rd argument used as both input and output.
hcs_$ex_acl_delete	3rd argument meaning depends on 4th argument.
hcs_$initiate_seg_count	6th argument meaning depends on another argument.
hcs_$list_dir_acl	4,5th arguments meaning depend on the value of 3rd argument.
hcs_$replace_sall	3rd argument unvalidatable.
hcs_$replace_dall	3rd argument unvalidatable.

The problem in each case here was deeper than in the previous one: the particular choice of arguments lead to impossiblity of validation, and therefore to no validation at all. For example, suppose that the third argument is an input argument for some values of the first argument, but is an output value for others. Then a protection specification which says that the third argument must be writable would cause some correct programs, which intentionally provided a read-only third argument, to be declared illegal. If, when these entries were first introduced, their documentation had specified that the argument in question must be writable whether or not it is actually written into by the supervisor, then the trouble could have been avoided (at the cost of an additional obscurity in the user interface). Unfortunately, an after-the-fact change to require writeability might cause some correct user programs to stop working, so compatibility prevents correction.

Again, the automatic argument validation hardware of the 6180 provides a solution. Since every reference to an argument is separately checked, only if the argument is actually used as an output argument will it be checked for writeability.

EPL argument validation trap

The argument validator did not completely check out some of the more complex specifiers of arguments provided by EPL (the first Multics PL/I compiler) programs. Thus, a user could construct an argument descriptor which indicated that an EPL specifier was in use, and thereby induce the argument validator to allow the call to go unchecked. This problem was basically one of historical compatibility: the EPL specifier format and organization was designed before the implications of argument validation had been considered. When it became clear that certain argument types were hopelessly complex to validate, an attempt was made to prohibit (by edict) the use of those types of arguments in supervisor entries. After the later PL/I compiler eliminated the need for a restriction, some gates were installed which utilized the forbidden argument types. The argument validator, unfortunately, provided a default of "acceptable" for EPL arguments of unvalidatable type, so it turned out that one could call the new entries with programs written in EPL, which was still an available compiler. The alternatives of changing the default to "unacceptable" would have effectively denied access to the new gates for those users not yet ready to rely upon a new unseasoned, PL/I compiler. Thus, through a series of design slipups, errors in judgement, and bad practices, this protection bypass got into the system.

The 6180 argument validation hardware again automatically performs the appropriate access checking at argument usage time, independent of the format of the structure passed as an argument.

ECT terminate bug

The design of the Inter Process Communication (IPC) event channel table (ECT) had the following flaw: when the user-ring IPC created an ECT, it then called a ring-zero entry to inform the ring-zero part of IPC of the location of the ECT. The pointer in question was stored by the ring-zero part of IPC in a ring-zero data base, for future use in passing IPC messages back to the user. The user could now terminate the segment containing the ECT, and initiate some other segment (to which he had only read access in the user ring) with the same segment number as the former ECT. Then, the ring zero part of the IPC, using its stored pointer, would write the user's messages in a place the user had no business writing into.

With the 6180 hardware, the pointer passed by the user to the ring-zero part of the IPC facility, and stored there, contains the ring number of the user's ring. Thus all reference made by ring-zero IPC using that pointer will be validated as though they came from the user ring. If a segment for which the user did not have write access is substituted, the attempt of the ring-zero procedure to write in it will fail.

Exploitation of user-ring master-mode procedures

The 645 processor had a "master-mode" property, which bypassed all protection checks; certain procedures such as the fault interceptor and signaller had to operate in master-mode, yet in the ring of the user causing the fault or receiving the signal. To prevent exploitation, the hardware permitted calls to a master-mode procedure only to an entry point at location zero in the segment; the procedure was expected to very carefully examine the circumstances of its entry to insure that it was not being exploited.

Upon review of the standard entry sequence code actually being used, it was discovered that the design did not prevent exploitation at all. Three distinct problems were found, each of which could be exploited in several ways. First, the entry sequence was designed on the assumption that index register one had been set to indicate which of several actual entry points to the segment was desired. The entry sequence correctly assumed that the caller might place an out-of-bounds value in index register one, so it checked to make sure that the value was within reasonable limits. Unfortunately, if the value was out of bounds, it called out to the system trouble-handling procedure, which proceeded to "crash" the system. Thus, any user could cause a crash by transferring to location zero of the signaller, with an appropriate value in index register one. The second problem is that the call to the system trouble handler was done by an indirect transfer out through the linkage section of the master-mode procedure -- but this call occurred before verification that the linkage pointer had been set to the currect value. Thus, the user could plant a special value in the linkage pointer, transfer to location zero of the signaller, and cause the master-mode procedure to transfer anywhere he wished -- including into the middle of another master-mode procedure. Again, by preparing registers in advance, and choosing

carefully the code sequence to transfer into, one could develop an exploitation. Finally, the third problem is that safe-storing of the processor registers was done assuming that the register value in the stack base register did not need to be checked, since it was locked. Unfortunately, a 1971 modification to the system resulted in the stack base register being unlocked, so the user could, by loading the stack base register and transferring to a legal entry point of the signaller, cause it to safe-store the processor register almost anywhere.

Although the concept of securing a master-mode procedure still seems viable, the implementation is apparently very fussy. By checking the Multics System Programmers' Manual it can be established that the first two problems have existed at least since 1967, and probably earlier. It was precisely because of uneasiness about the securing of master-mode segments that the 6180 was designed without a master-mode, and with consistent and builtin hardware call and fault facilities.

Execute instruction user special protection checks

On the 645 processor, the checking of permission was special cased when an "execute" instruction was encountered, since the time of decoding of the instruction to be executed is delayed to a time when most instructions are in the midst of execution.

Apparently as a result of a field change, one of the special cased checks was accidentally disabled if the execute instruction was located in an odd location and it addressed an offset of zero in another segment. In this situation, write permission was not checked, so one could write into a read-only segment.

Here we have an example of the danger of special cases -- they tend to cover rare occurrences, which means that routine operation does not exercise them. It also points out the recertification problem: even if a design is originally sound, every later modification should be accompanied with a recertification.