PROJECT MAC February 14, 1973

Computer Systems Research Division Request for Comments No. 35

The "PRINCIPLE OF LEAST PRIVILEGE" AND MULTICS

by J. H. Saltzer

A necessary, but not sufficient, condition for progress in developing a certifiably secure version of the Multics system is identification and following of some applicable design principles. This note briefly discusses a design principle suggested by David J. Edwards, of the National Security Agency and a former member of Project MAC: the "principle of least privilege". The principle simply states that every program should be granted the least privilege necessary to get its job done.

It is clear that in a complex system, all unnecessary interconnections must be eliminated. The principle of least privilege would go a step farther: even <u>potential</u> unnecessary interconnections should be made impossible. The desirability of eliminating potential interconnection is clear: the job of certification is made easier by reducing the number of potential defects for which auditing must be systematically provided.

A survey of Multics immediately turns up many examples in which the principle of least privilege has been followed, at least to some extent. For example, the ring structure has been used to place the command processor outside the central ring_zero supervisor; it does not need supervisor privileges. As a result, a certification of, say, the traffic controller need not even inquire as to whether or not the command processor writes directly into the Active Process Table. Similarly, the message segment facility is implemented entirely within ring one; again, traffic controller certification does not need to involve the programs of the message segment facility.

Probably more interesting than listing points which follow the principle of least privilege is to identify points which don't. Ring zero is a good place to start. Although it is apparent that all procedures in a single ring must depend on one another's integrity, the potential inter-dependence currently found in ring zero could probably be greatly reduced by applying the principle of least privilege. At the language level, a single PL/I program has access only to declared variables -- unless those variables include pointers, in which case the program has immediate access to any data item anywhere in ring zero. Thus, to certify correctness of the traffic controller, one should inspect every ring-zero program which uses pointer variables, and verify that there is no way in which those other programs can possibly affect the Active Process Table. This example suggests that unnecessary use of pointer variables is a violation of the principle of least privilege.

Note that in the discussion of "accessing privileges" of a PL/I program we are not talking about hardware enforced privileges, but rather on the linguistic constraints placed on the programmer. Forbidding unnecessary use of pointer variables does not by itself guarantee the principle of least privilege; it does seem to be moving in the correct direction, though.

So far, I have discovered no examples in which the principle of least privilege is in serious conflict with either functional goals or other design principles. Most violations seem to occur in the name of short-term programming convenience, or in cases which no other goal or principle offers design guidance. It is likely that further discussion and auditing of the system with this design principle in mind may yield much more insight.