PROJECT MAC
Computer Systems Research Division

April 23, 1975
Request for Comments No. 74

M.S. Thesis Proposal: A Secure and Flexible Model of Process Initiation for a Computer Utility.

by Warren A. Montgomery

Attached is a copy of my Master's Thesis proposal which deals with a reorganization of the functions now performed by the Answering Service. The proposal concentrates on a model for process initiation rather than on details related to Multics. I expect to produce a second RFC which describes the proposed implementation of the model.

Massachusetts Institute of Technology

Project MAC

Cambridge, Massachusetts

Proposal for Thesis Research in Partial Fulfillment

of the Requirements for the Degree of

Master of Science

Title: A Secure and Flexible Model of Process Initiation for a Computer Utility

Submitted by: Warren A. Montgomery
10 Westgate Dr., Apt 207
Woburn, Massachusetts
01801

Signature of Author:

Date of Submission: April 10, 1975

Expected Date of Completion: June 1975

Brief Statement of the Problem:

In order to guarantee the security of information in a computer utility, the initiation of processes must be carefully controlled. As the initiation of a process tends to be a very complicated activity, requiring the intervention of many programs, it is important to identify and isolate the programs necessary to perform this control. These are the only programs that must be certified correct in order to guarantee that process initiation does not violate the security of the information in the computer utility. This thesis will develop a model of process initiation in which the set of security-related programs is small and easily isolated. This model will be tested by implementation on the Multics computer system.

Supervision Agreement:

The program outlined in this proposal is adequate for a Master's thesis. The supplies and facilities required are available, and I am willing to supervise the research and evaluate the thesis report.

Jerome H. Saltzer,
Associate Professor of Computer Science and Electrical Engineering

Introduction

Digital computers are increasingly being used to store or manage valuable information. As the manipulations are less often checked, there is growing concern over the security of the information. Many commonly used computer systems may be "cracked", allowing anybody with a terminal to examine or alter any information under the control of that system. Several efforts are currently being undertaken to produce more secure computer systems. Most of these efforts involve defining new access control mechanisms. These new mechanisms are easier to certify correct and more powerful than their predecessors in that they allow a more precise specification of the conditions under which information can be examined or modified. An equally important problem that has received less attention is that of initiating processes for users so as to be certain that the user with whom the access control mechanism associates the process is actually controlling that process. It is in this area that many penetrations occur. A user who gains control of a process that the system believes to be acting for another can examine or modify information that he should not be allowed to see. Furthermore, if the method of penetration is general, the user may obtain control of processes executing in the most privileged environments of the system and therefore have access to any information in the system. The goals of this thesis are: (1) To demonstrate that a computer system can be designed to guarantee

that a user will always control processes created in his protection domains, and (2) to show that this can be accomplished while requiring that only very little of the code necessary to initiate and manage processes be certified correct in order to guarantee this condition.

For the purpose of this document, I will consider three classes of protection problems. Most protection goals that have been discussed fit neatly into one of these classes.

The first class I will consider is enforcement of orderly access. I will define orderly access as that situation in which the access control mechanism is correctly implemented and does not allow an object to be accessed without the cooperation of a protection domain that is explicitly given access to that object. This definition is intended to prevent a user from gaining access to an object which he is not entitled to use either by bypassing the access control mechanism in some way, or by forcibly controlling a process which is executing in a protection domain which is allowed to access that object. On a system that has the property of orderly access, a user could protect his objects from unauthorized manipulation.

A second class of protection problems is the prevention of denial of service. Denial of service occurs when one user can arbitrarily prevent another from making use of the computer system. The prevention of denial of service is clearly dependent on orderly access, as lack of orderly access allows a user to penetrate the operating system and modify it to prevent another

from making use of the system.

A third class of problems is that commonly referred to as the confinement problem. A general discussion of this problem may be found in Rotenberg's thesis <R74>. This problem involves specifying for each piece of information in a computer system the set of users who will be allowed to see that information, and guaranteeing that no action which can take place inside the computer system will result in a user not in that set seeing the information. A process that has read a piece of information must be confined to prevent it from communicating that information to a process belonging to a user who is not in the allowed set through some shared resource. A denial of service can be used to transmit one bit of information, and therefore any program which can deny service must either be certified not to violate confinement or must be prevented from gaining information not available to all users of the system. A system that solves the confinement problem must therefore prevent unauthorized denial of service. The confinement problem is by far the most difficult of the three problems discussed, and it may not be possible to solve. For this reason, it will be largely ignored in this proposal, although the thesis is expected to provide some insight into what must be certified about process initiation to solve the confinement problem.

From the above discussion, we can see that the code which must be certified to prevent denial of service must include all of the code which must be certified to insure orderly access.

Furthermore, as certification techniques for computer software are currently very primitive, it is desirable to minimize the code in each category. The following section presents a model for process initiation that accomplishes this goal.

A Model for Process Initiation

As a framework for the discussion of process initiation, I will take a system in which processes are allowed to wander among protection domains under control of the owners of the domains. Each protection domain is defined by a set of objects and of accesses applicable to those objects that may be performed by a process executing in that domain. Associated with each process is a principal identifier that specifies the protection domain in which that process is executing. Each user has a home domain that is under his control and each user may specify which domains have access to objects that he creates. This model is an attempt to capture the essential features common to both access control list based protection schemes, such as Multics <O72>, and capability based schemes, such as Jones <J73>.

The act of initiating a process can now be divided into two actions. First the process itself must be created, and then the newly created process must be inserted into some protection domain. This insertion consists of associating the new process with some principal identifier and specifying the entry point where it will begin execution. If the insertion is carefully controlled, the creation of a new process cannot violate the

condition of orderly access. This can be accomplished by limiting entrance to each domain to a set of entry points supplied by the owner of the domain. Mechanisms similar to those used to control calling a protected subsystem <Sc72> or calling a domain <J73> or calling "subroutines" in Lampson's message model <L71> can be used for this purpose.

There is one point not covered by the above mechanisms that is important in the area of process initiation. This is the structure of the process that is allowed to enter a domain. We can see that if the process creation mechanism were not certified, the mechanism that controlled the entrance of a process into a domain could make no assumptions about the structure of the process attempting to enter. For example, before a PL/1 program could be executed we would have to verify that all of the attributes of a process that make up the PL/1 execution environment (stack segments, temporary storage, inter - procedure linking, etc.) were functional. Furthermore, these checks would have to be made for every domain crossing. This can be very expensive and difficult, since the domain entering mechanism may have to execute using the process that it is checking. It seems desirable to certify a small kernel of the process creator that creates a process with a set of standard attributes. This set must include all attributes necessary for the proper functioning of the system's process implementation and all procedures that execute in a user created process and that must be certified to guarantee orderly access. It would be

wasteful to include any attribute that can be changed during the execution of a process, as its proper creation would not guarantee its continued existence. In short, we must certify that the virtual processor created to evolve the user's process meets certain minimal requirements. Thus to guarantee orderly access, we need only certify a domain entry mechanism, which is also needed for the implementation of cross-domain calls, and a small kernel of the process creation mechanism.

Authentication

We have shown how a computer system can be constructed such that the owner of a protection domain can always gain control of processes entering that domain by specifying the only allowed entry points to the domain. The purpose of having a process execute in a user's domain is generally to perform some function for the user. The function to be performed is usually not known until the time of execution and therefore must be supplied to the process at the time of execution through some communication channel, which I will refer to as a stream. In writing the code for the entry points to his domains, the user will want to insure that each stream that in some way will direct the course of the process has a source who he trusts and that the information in the stream is not tampered with before reaching the process. Otherwise, it would be possible for some user to control a process inside a domain that he does not own.

The identity of the source of a stream can be established by

authentication. This can be done by prefacing the information content of the stream by a password previously assigned by the owner of the domain. Another authentication scheme that avoids the possibility of password theft is to encrypt the information sent through the stream. The source of the stream must provide an encryption key that matches the one used by the process to that it is connected.

In a conventional computer system, the identity of the source of any stream originating outside the computer system is verified before the stream is connected to a user process. When a process is created to serve a stream, it generally begins execution in the home domain of the source of the stream and executes code that is prepared to receive commands from the stream. Thus all users depend on the authentication mechanism used by the process initiator for their security.

In our model, the owner of a domain always receives control of processes which enter his domain so that he is not dependent on an authentication mechanism for the security of his domain. Each domain owner can choose his own authentication mechanism to identify the sources of the streams connected to processes executing in his domain. This allows greater freedom in choosing authentication mechanisms and eliminates any dependence on a system-wide mechanism. As most authentication mechanisms are only probabilistically secure, a free choice of mechanisms also allows a user to choose a mechanism with a lower chance of failure than any system defined one.

There are still cases, however, where it is undesirable for each domain to authenticate the streams it uses. Whenever a stream will be connected to many processes and domains during its lifetime it would be awkward and wasteful for each to have to authenticate the source of that stream. If some of the domains involved trust others, a great deal of effort can be saved by recording the result of each authentication and allowing each domain that encounters the stream to examine the results of previous authentications rather than authenticating the stream for itself. I will refer to this activity as the forwarding of an authentication. This is a very common and important phenomenon in the physical world. For example, a store is using a forwarded authentication when it requires a customer wishing to pay by check to produce a driver's license. The store is assuming that a careful authentication was made by the issuer of the license.

There is a second type of information source that may direct a process. This is information recorded within the system. As the file system can determine exactly the domain writing such information, it is easy to record the source of stored information along with the information itself. This allows any process that draws commands from a stored file to know the source of those commands before acting on them.

Resource Accounting and Control

An executing process consumes processor time and uses various forms of the system's memory. As these resources are shared, their use must be carefully controlled. I will refer to the set of programs that account for and control these resources as the resource controller. The traditional model for the interaction of resource control and process initiation is a hierarchical structure in which each process is responsible for the resources consumed by its descendants. Thus whenever a process is created, the creator designates some of the resources granted to it to be granted to the new process. In an extreme form of this organization demonstrated by the CAP system <W73>, the creator is also responsible for establishing the protection domain in which a new process will execute as a subset of its current protection domain. This model has the advantage that a process should know the needs of the processes it creates and therefore is in a good position to determine the resources to grant them.

A major drawback to this scheme is that resource control must be performed in many processes, thus duplicating much of the mechanisms needed. Another drawback that makes this scheme unsuitable for this thesis is that a process has a great deal of control over the processes it creates, thus making it impossible for a process to be suspicious of its creator. As our goal is to allow any process to initiate a process in any domain, it is clear that we must allow a process and its creator to be

mutually suspicious. Thus in our model we will consider resource control to be performed by a central authority.

We will assume that a process is devoted to resource and that this process executes in a domain that gives it access to the functions necessary to perform resource control. We can see that the resource controller will need to be able to read the resource usage statistics of all of the processes in the system. It will also need the ability to halt any process that attempts to use resources to which it is not entittled. A third necessary ability is to be able to prevent the initiation of any process which is not entittled to some resource required for its initiation. With only these abilities, we can implement many resource control policies. It is not possible, however, to violate the condition of orderly access using these abilities.

Details of the Multics Implementation

In the Multics system, each process is assigned to a group of eight protection domains that are nested with respect to the accesses they allow. The largest domain of each group includes all objects in the system. Each process may wander among its eight domains but cannot leave its particular group. There is one privileged process that has the ability to create a process in any protection domain; it is known as the Answering Service. This process performs the authentication and resource control functions as well as attempting to insure that the owner of a group of protection domains always gains control of processes

entering that group. In this organization, all of the code that executes in a domain with access to the process creator must be certified to insure orderly access. This includes all code required to perform the functions mentioned above and some additional code not related to security at all.

As shown by the model, we can by reorganization greatly reduce the amount of code that must be certified. This reorganization is the proposed thesis.

The first aspect of process initiation that I will consider is the certified kernel that creates a standard process. As shown previously, this kernel must create a process with all of the attributes required by the certified procedures that will execute in each process, but need not include any attribute which may change during the execution of the process. In Multics, the process creation function that executes in the most privileged domain provides a set of attributes between these limits and therefore seems a good choice for the certified kernel. This mechanism provides each new process with a new addressing environment, a process directory for temporary storage, a new I/O environment, a mechanism for the orderly handling of faults and interrupts, and a unique identifier that distinguishes it from all other processes. Any attribute that is not supplied by the kernel and that a user wishes any process entering his domains to have must be checked at the entry points to his domains.

The other part of the process initiation scheme that must be certified in order to guarantee orderly access is the mechanism

that controls entry to a domain. A simple mechanism that serves this purpose is to allow each domain owner to create domain entrance objects for his domain. The access control list associated with such objects would be used to determine whether or not to allow a process to create a new process using that entry point. These objects could be maintained in that section of the file system devoted to storage for the user's home domain. This is similar to the scheme user by Schroeder <Sc72> to control access to protected subsystems. A process creator that checks the access control list before calling on the kernel to create the appropriate process could be written and made available to all domains.

The resource control functions that now execute in the Answering Service could easily be moved to a domain that had only the ability to terminate any process. Currently, the resource controller learns of all process initiations because the Answering Service is the only process that initiates processes. A small amount of code would need to be added to the process creator to inform the resource controller of new processes being created. None of the modifications needed appear to complicate significantly the code that must be certified.

The authentication forwarding mechanism described in the previous section could easily be implemented on Multics. It would record the domain and procedure which performed the authentication as well as the result of that authentication. The author of stored segments can in most cases be deduced from

information that Multics currently maintains. Extension of the recorded information to include the last domain to write in each segment would not be difficult.

Appendix B presents a summary of the reorganized process initiator, indicating which modules must be certified to guarantee orderly access and which need certification to prevent denial of service. It should be compared with the current list presented in Appendix A.

Resources Required for This Thesis

The completion of the thesis will take approximately 3 months. The project will require computer resources from the M.I.T. Multics system for coding and debugging the proposed changes for the new organization of process initiation. A smaller amount of time is required with a Honeywell 6180 computer system for final checkout and demonstration of the new mechanisms. No programming support should be necessary, as the amount of programming is relatively small.

Summary

The problem of computer security is becoming increasingly important. This thesis attacks an area of that problem that has been somewhat neglected: Guaranteeing that a user gains control of processes created to act in his protection domains. A model is developed to show that this can be accomplished with much less certified code than is commonly used. The proposed thesis would

apply the model to the Multics system to demonstrate its feasability. This would involve the generation of a process initiator for Multics that allows any one process to request the creation of another process for any user, and requires a minimum amount of certified code.

Appendix A
A summary of the Multics Answering Service

The following is a list of the modules which make up the Multics Automatic Answering Service. A summary of each module's security requirements and its dependence on the current organization is given. All modules listed now execute in the Answering Service and therefore must now be certified.

absentee_user_manager_

Under the proposed reorganization, the absentee subsystem needs only the ability to communicate with the resource controller in order to give absentee processes a different status when checking loading restrictions and charging for CPU time. It is possible to operate the absentee subsystem in an unprivileged environment as follows: When submitting an absentee request, the user's process records the name of the segment containing the commands to be executed in a segment only accessible in the user's home domain. The user provides an entry point to his domain that checks to see that the proposed command file is on the list of those submitted for absentee requests before executing any commands. When finished, each absentee process removes its command segment from the list of unfinished absentee requests. With these precautions, it is impossible for the absentee subsystem to violate the condition of orderly access.

act_ctl_

This module handles resource accounting. It needs certification to prevent denial of service, as it requires the ability to halt processes that have exhausted their funds. It now makes use of data in the Answering Service tables and would require extensive reworking.

admin_

This module handles the "system control" functions assigned to the Answering Service. It requires the ability to halt processes. Currently, this module acts as control for most of the Answering Service. If the current interface is to be maintained, a communication mechanism to allow admin_ to pass commands to the absentee subsystem, load controller, and resource controller will be needed.

cpg_

This is the process creator. Very little of it needs certification to insure orderly access, as the hardcore function it calls provides an adequate standard process. The primary function of this module is to gather data from various tables and feed it to the new process. Some

alternate mechanism should be provided to make this information available to processes not created by the Answering Service.

create_homedir_
This module creates home directories for processes that need them. Unfortunately, this act requires access to all of the places where home directories can be put, including the project directories. This function should be performed by the newly created process whenever possible. The process requesting the new process could create the home directory if it were sufficiently privileged, but the new process should no longer depend on having a home directory created and should be prepared to create its own.

daemon_user_manager_
This module serves a similar function to dialup_ for daemon processes.

dial_ctl_
This module manages dialed consoles. It does not deal with process creation or management and does not need certification.

dialup_
This module is an interface between the user's terminal, his process, and the rest of the Answering Service. It receives requests for Answering Service services from the terminals attached to the system and from existing processes and passes these requests to the appropriate part of the Answering Service. It does not need certification.

dpg_
This module calls on a hardcore function to destroy processes after erasing them from Answering Service tables. As it calls the process destruction function, it must be certified to prevent denial of service.

lg_ctl_
This module authenticates incoming streams and sets up some of the information for initiating processes. This module need not be certified, as a process can authenticate its streams for itself. It will be necessary, however, to provide some certified authentication mechanism.

load_ctl_
This module controls the number of processes allowed on the system and thus must be certified to prevent denial of service. Once again its data bases are too closley mingled with the rest of the Answering Service and this will require reworking.

up_pdt_, up_pnt_, up_sat_, up_sysctl_

These modules all update tables that control process initiation and resource control. These tables control the action of the resource controller and therefore all mechanisms dealing with them must be certified to prevent denial of service. If the information that controls entry to a domain is seperately maintained, these modules cannot interfere with orderly access.

Appendix B
Modules of the Reorganized Answering Service.

1) Modules that require certification for orderly access.

   a) Kernel of the process creator. This requires few changes.
   b) Domain entrance mechanism. This is a small new module (100 lines of PL/1 code).
   c) Resource controller's interface with the process creator. This has not been precisely worked out, but will require approximately 100 lines of new code.

2) Modules that require certification to prevent denial of service.

   a) Resource controller. The current resource controller can be adapted to this purpose with relatively few changes.
   b) Operator interface to the resource controller. This requires almost no changes.

3) Modules that do not require certification.

   a) Terminal management software.
   b) User interface to the process creator (Software that prints log on messages, identifies the user, and starts a process for him).

4) Modules that do not require certification, but whose certification would facilitate system use.

   a) Authentication recorder.
   b) Standard entry point procedures.
   c) Certified authentication procedures.

References

<J73> Jones, A. K. "Protection in Programmed Systems" Ph.D. Dissertation, Department of Computer Science, Carnegie - Mellon University, 1973.

<L71> Lampson, Butler W., "Protection", Proceedings Fifth Princeton Conference on Information Sciences and Systems, Princeton University, March 1971, pp. 437-443

<O72> Organick, E. I. The Multics System: An Examination of Its Structure M.I.T. press, Cambridge, Mass., 1972

<R74> Rotenberg, Leo J., "Making Computers Keep Secrets", M.I.T. Project MAC, MAC-TR-115, 1974

<Sc72> Schroeder, M. D. "Cooperation of Mutually Suspicious Subsystems in a Computer Utility", M.I.T. Project MAC, MAC-TR-104, 1972

<W73> Walker, R. D. H. "The Structure of a Well-Protected Computer" Ph.D. Dissertation, University of Cambridge, 1973