PROJECT MAC
Computer Systems Research Division

December 18, 1975
Request for Comments No. 98

A Certifiable System Initialization Mechanism

by Allen W. Luniewski

This is a copy of my recently accepted master's thesis proposal. Comments and suggestions on it will be greatly appreciated.

Brief Statement of the Problem:

In order to certify a computer system as secure, it is necessary to certify an initial state of the system as secure. It is the purpose of system initialization to produce this secure initial state of the system. Current system initialization techniques are very difficult to certify due to the ad hoc techniques that they employ. We propose an alternative system initialization mechanism that is more easily certified than current mechanisms. Its chief characteristics are the concept of a minimal system configuration and the use of dynamic reconfiguration. These serve to greatly reduce the amount of ad hoc initialization code needed and thus make the task of certification easier. An implementation on Multics, a prototype computer utility, will be undertaken as part of this thesis.

Thesis Supervisor: David D. Clark

Introduction

The need for protection mechanisms within a computer utility has become accepted within the computer system community (including users, designers and researchers). Once security is recognized as important, it is necessary to certify that the computer utility meets this requirement. To show that the computer utility is secure one normally shows that it satisfies a property of the form "Given a secure state, a sequence of stimuli applied to the security kernel (1) leaves the system in a secure state". This results in a need to certify the initial state of the system [1], created at system initialization time (also known as bootload time or initial program load time), as secure. Current initialization techniques are rather ad hoc, resulting in difficulty in performing this certification. This proposed thesis will investigate a more easily certified method of system initialization.

System Initialization - A Definition

The secure initial state of the security kernel consists, in general, of five parts:

1. Correct contents of main memory.
2. Correct contents of secondary memory (such as disks).

---

(1) The security kernel is that part of the system supervisor which is concerned with providing the facilities on which the security and protection notions depend.

3. Correct contents of registers in hardware modules.

4. Correct setting of manual switches on hardware modules.

5. Correct physical intermodule connections.

The exact nature of each of these items is dependent upon the security model one is operating under. (2) System initialization consists of all of the activities necessary to get each of these areas into its proper initial state. In doing so it also gets the security kernel into its correct initial state.

Initialization and the System Configuration

We define the system configuration to consist of the items of hardware in the system and the physical connections between them. Additionally it consists of software parameters which control the actions of the security kernel. Thus the system configuration includes the processors, memories and the I/O devices present as well as the sizes of system tables and the values of tuning parameters.

From our definition of the secure initial state of the security kernel, we see that the secure initial state of the kernel is a function not only of the security notions one is using but also of the particular system configuration present.

---

(2) For some cases it may be that subsets of each of these areas are unimportant to the secure initial state of the security kernel.

System initialization must produce a secure initial state of the kernel that reflects the system configuration. This means that the actions taken by system initialization are a function of the system configuration. System initialization is thus driven, to a large extent, by the system configuration present at bootload time.

Certification of an Initial State

The initial state of the security kernel can be certified as secure in one of two basic ways. One way is to certify the result of system initialization. That is, we certify that the components of the initial state resulting from system initialization represent a secure kernel. Alternatively, we could certify the system initialization routines themselves. In this case we certify that regardless of the configuration these routines will either produce a secure initial state or they will report failure to the operator (i.e. the attempt to initialize the system fails). We reject the first method since the certification process must take place each time the system is initialized. This is so since although it may be possible to certify a given initial state on a given configuration as secure, such a technique would allow us to certify the system as secure only for the duration of a particular bootload. Subsequent bootloads would require recertification of the initial state. This is because we do not know whether or not the system initialization routines will always produce the same state for a

given configuration and we have no idea what will happen if we change the configuration. For instance, they may have real time dependencies (that are unlikely to be reproduced) that make the difference between a secure and an insecure system. For this reason we choose to certify the system initialization routines themselves. In this case we certify that, regardless of the configuration, the system initialization routines either leave the system in a secure state or cause the bootload to fail. In this way the certification process need take place only once.

Thesis Limits

The general topic of this thesis will be the design of a certifiable system initialization mechanism. It would be nice to develop a metadesign (one applying to many computer systems), however this will not be done since system initialization is tied very closely to all aspects of a particular computer system. This means that the development of such a metadesign would entail the development of a model of a large class of computer systems, a task which looks very difficult. As our interest is not to model many computer systems this thesis will restrict itself to a particular system with the hope that the ideas developed will be applicable to other systems.

This proposed thesis will examine system initialization on Multics [2], a prototype computer utility. Multics is chosen for

three reasons. First it is written in a high level language, PL/I. Second it is reasonably modular, more so that many present operating systems. These two reasons mean that understanding of, and modification to, the existing software will be easier. These also contribute to Multics' designed ability to evolve easily. Since the initialization scheme to be developed will be an extension of the current mechanism this ability to evolve is very desirable. These two reasons also mean that the produced initialization mechanism could be more easily certified. A third reason is the availability of Multics to the author and the author's familiarity with it.

In this thesis we shall be concerned with that part of the Multics security kernel which runs in ring 0. We will not be concerned with the initialization of those parts of the security kernel which reside outside of ring 0. This is a reasonable restriction since the initialization of the non-ring 0 parts of the security kernel is relatively easy. This is because these initialization routines can use the standard environment provided by a fully initialized ring 0 to run in.

Initialization on Multics

All of the actions of system initialization on Multics (and on all systems in fact) take place either before running on the bootload processor(s) (3) or while executing on it. Actions

which take place before running on the bootload processor consist entirely of generating the bootload medium. (4) This medium contains kernel programs and kernel data as well as initialization programs (i.e. programs to run on the bootload processor prior to the attainment of the secure initial state) and their associated data. It is the purpose of the initialization routines, while running on the bootload processor, to take the kernel programs, kernel data, configuration information, the contents of main and secondary memory and the contents of the bootload medium and produce a secure initial state of the security kernel.

Currently in Multics very little happens prior to running on the bootload processor. The programs comprising the kernel and the initialization programs are compiled and placed on the bootload tape. Additionally access information is provided for each segment for incorporation into descriptor segments. Some segments also have their ultimate location in the file system specified as their well as their access control information. All other actions necessary to initialize the system occur while running on the one bootload processor.

Multics presently uses an incremental, or "bootstrapping", mechanism to initialize the system while running on the bootload

---

(3) The bootload processor is the processor that the system runs on during system initialization.

(4) Usually either disks, magnetic tape or punched cards.

processor [3]. That is, starting from a very primitive environment present when the first part of the system is read into core, the initialization programs add pieces to the environment, creating a new environment. Then, while running in this new environment, they add more pieces to the environment creating a new environment. This continues until at the very end we have a completely initialized Multics. This scheme suffers from two flaws from the point of view of certification.

First is the fact that it is an incremental mechanism. During initialization parts of the standard ring 0 environment are serially made active and then used by initialization. Thus parts of the security kernel are in use before it has all been initialized. By its very nature this incremental mechanism defines a hierarchial layering of the system [4]. For a hierarchially designed system this is a reasonable way to proceed since the layers are well defined by the system design. While running on the virtual machine defined by layers 0 to i the initialization routines can construct a secure initial state of level i+1 and then run on the virtual machine defined by layers 0 to i+1. However, for a non-hierarchial system bootstrapping is inappropriate. The global interactions present in a non-hierarchial system make the bootstrapping process difficult to understand and certify. (5) The initialization routines must

---

(5) It is not claimed that this results in a non-certifiable system. However, we do claim that the system is more difficult to certify due to these interactions.

not invoke or allow to be invoked (via a fault or interrupt) any routine which can not yet function properly. A routine might be unusable because it requires a data base that has not yet been constructed, a routine it uses may be unusable or the routine may not even be in the virtual memory yet. Invocation of an unusable routine might lead to unexpected, and insecure, results. The non-hierarchial nature of the ring 0 supervisor of Multics today thus leads to the flaw of global interactions in a bootstrapping initialization scheme.

The bootstrapping process and the non-hierarchial nature of Multics lead to a second flaw - multiple non-standard environments. These multiple non-standard environments result from the functional hierarchy imposed upon ring 0 during initialization. They make the task of certification more difficult by their presence. The certifier must understand the many environments present, instead of just the standard environment, thus increasing the complexity of the certifier's task. In Multics these flaws manifest themselves in the rather ad hoc nature of the bootstrapping process employed.

An Alternative Initialization Mechanism

It was pointed out earlier that the actions performed by the initialization routines must be guided by the current system configuration. The current structure of Multics initialization is largely motivated by this necessity. The various incremental

steps, as they execute, create a version of the initial state which is tailored to the particular configuration present. In this way, Multics can be initialized on a wide variety of configurations without changing the bootload medium.

This thesis will concern itself with the design of an alternative initialization scheme for Multics that will avoid the flaws mentioned above while preserving this highly desirable property of having one bootload medium work for many configurations. The basic idea is to reduce, as much as possible, the amount of code executed during the incremental phase. Operations now performed in the incremental bootstrapping phase will be identified which are more properly regarded as reconfiguration operations on a secure security kernel rather than as initialization operations. Other activities will be identified as being common to all bootloads and thus can be performed at the time the bootload medium is generated. The net effect is that much less code need be executed on the bootload processor prior to the establishment of the secure initial state. This reduces the number of non-standard environments present during initialization and thus makes the task of certification easier.

Initialization will proceed basically as follows. A core image of the system will be constructed when the bootload medium is generated. This will contain a copy of the security kernel as it should appear in core. At system initialization time the core

image is loaded into core and control transferred to it. The thesis will show that only a small number of operations need be performed to create a standard and secure environment at this point. This standard environment will be the normal Multics ring 0 environment, less the file system. (6) At this point another set of operations will cause the file system to work. This results in a standard Multics ring 0 environment. At this point reconfiguration operations are used to adapt the now running system to the desired hardware and software configuration.

The thesis will show that an expanded set of reconfiguration operations makes it possible to generate the core image while making few assumptions about the actual configuration the system will be coming up on. This is important since the assumptions that one does make about the configuration one will be coming up on serve to define a minimal configuration. This minimal configuration will be a subset of all possible configurations that one can bring the system up on. By keeping the minimal configuration small we increase the number of possible configurations we can boot on. (7)

---

(6) We exclude the file system as it appears to be a special case. It's initialization, while not ad hoc, is not a simple matter. Thus we prefer to define an environment with no file system rather than trying to initialize the file system as part of core image initialization.

(7) This is desirable in order to maintain compatibility with the current system which can boot on an extremely wide range of configurations with no special effort being required to do so.

Reconfiguration

As part of this thesis we will consider reconfigurations upon the security kernel. It is beyond the scope of this thesis to design and implement all possible desirable reconfigurations. Instead we shall pick reconfiguration operations which are:

1. Relevant to the proposed initialization scheme.
2. Illustrative of the types of problems one might encounter in developing other reconfigurations.

By using these critera we will choose reconfigurations which indicate the correctness of the initialization design and which provide guides to developing other reconfiguration operations which will complete a full implementation.

Reconfigurations can be classified as either hardware or software reconfigurations. Hardware reconfigurations include the addition and deletion of central processors, memories, input-output multiplexors, front-end processors and I/O devices. Software reconfigurations basically involve adjustment of security kernel parameters (such as scheduler parameters) and changing the size of system tables.

In this thesis we will consider only additive type reconfigurations. In terms of Schell's thesis [5] we can characterize additive reconfigurations as being those which do not require a rebinding of logical to physical resources nor do

they require the removal of a resource from a resource manager. Thus deleting a central processor is not additive since processes running on it must be moved to another processor. We restrict ourselves to additive reconfigurations for two reasons. First they will be the only kind needed for the proposed initialization scheme. Since we always start with a minimal configuration we need only add to the minimal configuration in order to get to the actual configuration, we never need to delete. Second, as noted by Schell, non-additive reconfigurations tend to be much harder than additive ones. As reconfiguration is not the central topic of this thesis we will avoid this harder problem.

Hardware Reconfigurations

We will discuss and implement two examples of hardware reconfigurations. We will consider the problem of adding an input-output multiplexor and of adding a front-end processor. We do not discuss the reconfiguration of central processors and memories since Schell has already covered the topic in detail. These two examples will illustrate solutions to two major problems of hardware reconfigurations. First we will see how the new hardware module can be made known to the relevant software. Second we will investigate the problems of dynamically changing channel masks, interrupt masks and simulate patterns (8) in

---

(8) A simulate pattern is usec by Multics to allow a processor to send a particular interrupt to a particular processor in the system.

response to such an addition. These changes are needed in order to open the relevant physical communication paths. These two problem areas are illustrative of the problems one might encounter in adding other hardware items such as tapes, line printers and card readers.

Software Reconfigurations

We will investigate the dynamic reconfiguration of the active segment table as an example of a software reconfiguration. It is chosen since it is illustrative of the major problems one can expect to encounter when attemoting a software reconfiguration. We shall now outline three such problems.

On Multics it is common practice to place different tables in the same segment. Thus if one table needs to be expanded it may be necessary to move one or more tables out of the way with all the potential problems associated with such a move. The active segment table illustrates this since it resides in the same segment as the core map and various paging device tables.

In Multics the tables we are interested in reconfiguring are shared by all processes. They also have the property that they are paged and a segment fault can not be taken while accessing them. Additionally Multics uses fixed sized page tables, residing in a segments active segment table entry (ASTE), for access to paged segments. All this means that if such a segment must get larger in response to a reconfiguration request and if a

larger page table (and hence ASTE) is needed to do this, the segment descriptor word (SDW), which points to the page table, must be changed in all processes to point to the new ASTE. This must be done in a way so that all processes continue to function properly. Ways to do this will be considered for the active segment table.

The third problem associated with software reconfigurations are absolute (or physical) addresses of items in main memory. For instance the descriptor base register, page table words, segment descriptor words and DCW lists (9) all use absolute addresses to reference memory. Because the software runs in a virtual memory, it has need to convert from real addresses to virtual addresses and back again. This conversion is currently facilitated by keeping each set (i.e. all page table words) in one segment which is contiguous in real memory. Thus the virtual to real conversion involves only an addition of the absolute address of the base of the segment to the offset within the segment to get the real address from the virtual address. Similiarly to get the virtual offset within a segment from an absolute address one subtracts the absolute address of the base of the segment from the given real address. For generalized reconfigurations it may not be possible to preserve this contiguity of segments. Thus alternative methods of performing these conversions are needed. We will discuss this problem using

---

(9) A DCW list is a list of words in memory used to control an input-output multiplexor.

the active segment table as our example.

Thus in this thesis we will examine the problems of hardware reconfigurations by using two examples. We will explore how to add an input-output multiplexor and how to add a front-end processor to the system as examples of these problems. The dynamic reconfiguration of the active segment table will be used to investigate the problems associated with software reconfigurations.

Implementation

As part of this thesis an implementation of the design will be undertaken. The main parts of the implementation will consist of the routines to generate the core image, the routines to load the core image and the routines to initialize the core image. Additionally some reconfiguration operations will be explored. The purpose of the implementation is three fold. First it will show that the system can in fact be initialized by this method. Second the implementation will demonstrate the ease in performing and understanding these three parts of the design. Third it will serve as a working model. It will provide a testbed on which design problems can be seen and corrected.

Current Status and Remaining Tasks

At this time the basics of the design have been formulated. Also the design, implementation and testing of the dynamic

reconfiguration of the active segment table has been completed. The details of generating, loading and initializing the core image are not precisely known at this time. The development of the detailed design of each of these can proceed in parallel as they should be essentially independent. The development of new reconfiguration operations can proceed independently of these three tasks since the reconfigurations can be designed and tested by using essentially the current system.

Bibliography

[1] Project MAC Progress Report XI, Massuchesetts Institute of Technology, July 1974.

[2] Introduction to Multics, MIT Project MAC Technical Report 123, Masssuchesetts Institute of Technology, July 1974.

[3] System Initialization, Honeywell Information Systems, Inc., OrderNo. AN70, 1975.

[4] Dijkstra,Edsger W., The Structure of the "THE" Multiprogramming System, Communications of the ACM, Vol. 11, No.5 (May 1968), pp 341-346.

[5] Schell, Roger R., Dynamic Reconfiguration in a Modular Computer System, MIT Project MAC Technical Report 86, Massuchesetts Institute of Technology, June 1971.