AD-771 758

# ENCAPSULATION: AN APPROACH TO OPERATING SYSTEM SECURITY

Richard L. Bisbey, II, et al

University of Southern California

Prepared for:

Advanced Research Projects Agency

October 1973

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER ISI/RR-73-17 | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle) Encapsulation: An Approach to Operating System Security | | 5. TYPE OF REPORT & PERIOD COVERED Research Report |
| | | 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s) Richard L. Bisbey II, Gerald J. Popek | | 8. CONTRACT OR GRANT NUMBER(s) DAHC 15 72 C 0308 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS USC Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, California 90291 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS ARPA Order #2223/1 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency, 1400 Wilson Blvd, Arlington, Virginia 22209 | | 12. REPORT DATE October 1973 |
| | | 13. NUMBER OF PAGES 16 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) ----- | | 15. SECURITY CLASS. (of this report) None |
| | | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

Distribution unlimited. Available from National Technical Information Service, Springfield, Virginia 22151

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

-------

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer security, encapsulation, hypervisors, minicomputer, operating systems, program verification, protection, retrofit, software security, verification, virtual machines.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

(OVER)

19

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

## 20. ABSTRACT

Currently, there does not exist a certifiably secure, multiuser operating system. No operating system has been able to withstand malicious attacks by skilled penetrators. Nevertheless, there exists a strongly felt need, both in the military and civilian sectors, for reliably secure operating system software. At the same time, any solution to the security problem must take into account the enormous investment in existing equipment and software.

Hypervisors are discussed as an approach to retrofitting security, but are rejected due to the high cost and complexity involved in their installation on existing equipment. An alternative solution, encapsulation, is proposed for batch and RJE systems. It involves the use of a small amount of additional hardware and verified software. The resulting system can be certified to be secure, and is suitable for stringent military requirements. The solution is applicable, essentially unchanged, to a wide class of hardware and software, and it is insensitive to special versions of, or changes to, operating system code. Operating efficiency and costs of construction are discussed in this paper to demonstrate the feasibility of encapsulation.

This work has been performed under Advanced Research Projects Agency Contract DAHC15 72 C 0308. It is part of a larger effort to provide securable operating systems in DOD environments.

ARPA ORDER NO. 2223/1

1.

ISI/RR-73-17
October 1973

Richard L. Bisbey II
Gerald J. Popek

# Encapsulation: An Approach to Operating System Security

INFORMATION SCIENCES INSTITUTE
UNIVERSITY OF SOUTHERN CALIFORNIA
4676 Admiralty Way/ Marina del Rey/ California 90291
(213) 822-1511

THIS RESEARCH IS SUPPORTED BY THE ADVANCED RESEARCH PROJECTS AGENCY UNDER CONTRACT NO. DAHC15 72 C 0308, ARPA ORDER NO. 2223/1, PROGRAM CODE NO. 3D30 AND 3P10.

## ABSTRACT

Currently, there does not exist a certifiably secure, multiuser operating system. No operating system has been able to withstand malicious attacks by skilled penetrators. Nevertheless, there exists a strongly felt need, both in the military and civilian sectors, for reliably secure operating system software. At the same time, any solution to the security problem must take into account the enormous investment in existing equipment and software.

Hypervisors are discussed as an approach to retrofitting security, but are rejected due to the high cost and complexity involved in their installation on existing equipment. An alternative solution, encapsulation, is proposed for batch and RJE systems. It involves the use of a small amount of additional hardware and verified software. The resulting system can be certified to be secure, and is suitable for stringent military requirements. The solution is applicable, essentially unchanged, to a wide class of hardware and software, and it is insensitive to special versions of, or changes to, operating system code. Operating efficiency and costs of construction are discussed in this paper to demonstrate the feasibility of encapsulation.

This work has been performed under Advanced Research Projects Agency Contract DAHC15 72 C 0308. It is part of a larger effort to provide securable operating systems in DOD environments.

created with security as a major design parameter are easily penetrated. In general, <u>retrofitting the multiple versions of various operating systems by revising their code is impractical</u>.

Nevertheless, the security problem of existing systems is important and will not disappear in the coming years. The only solution now available to those installations requiring protection has been to compartmentalize: to have separate operating systems for each security category or level. For the military, this means having an operating system for each of the four security levels: Unclassified, Confidential, Secret, and Top Secret. For the typical commerical installation, this might mean having separate operating systems for payroll, accounts receivable, and general computing. Security is achieved through physical isolation, but at a substantial cost. A considerable amount of useful machine time is wasted changing systems (often called charging "colors"), since it is necessary for the existing operating system to be shut down, all storage either physically removed or written over, and the next system initiated. For some installations, a single change can require an hour or more. To minimize lost time, rigid schedules must be established, meaning that the system is inflexible to the needs of its users. Also, sharing can only be achieved through off-line procedures, controlled administratively.

## INTRODUCTION

Over the past few years, the computing community has seen a steady increase in the number of applications using information that must be protected from accidental, unauthorized, or malicious use. Today's multiuser computer systems should be able to provide this protection using a combination of hardware and software controls. In fact the systems are unsatisfactory. While some progress is currently being made in the constructive design of new, secure operating systems, that work is not expected to bear practical fruit immediately. In the meanwhile, there currently exist large numbers of disparate computers and operating systems for which security is an important concern.

For someone skilled in the art, penetrating today's operating systems takes little more effort than solving a hard Sunday crossword puzzle, and it can be considerably more lucrative. Retrofitting these systems, in the general sense of repairing the respective operating systems in all their various versions, is an enormous task, which is not well understood. To date, all such attempts have met with failure. In one case, several million dollars was spent to create a multilevel secure version of an existing operating system only to have the result quickly breached by an outside penetration team. Even systems that have been

A solution to the security retrofit problem for batch and RJE systems is proposed. It is fairly simple, appears economical, and will provide certifiable security across various manufacturers, computers, and versions of operating systems. That is, the same solution may be employed, regardless of which manufacturer's equipment is involved or what particular operating system version or modification is being run. The method is related in spirit to virtual machine designs, in the sense of CP-67, VM/370 or UCLA-VM [2,4].

VIRTUAL MACHINES

A hypervisor, or virtual machine monitor, is a program whose task is to provide multiple program environments that are logically identical to the bare hardware on which the hypervisor runs. Normal operating systems, of course, provide multiple environments, but those environments have been altered from that of the original bare machine. Certain capabilities, notably resource management and I/O instructions, have been removed, while extensive user services have been added.

Because a hypervisor provides few services, it is a much simpler program than an operating system. Its major tasks are to provide separate environments, called virtual machines, and to simulate the behavior of instructions, such as those that change relocation registers, which cannot be performed directly by programs running on the virtual machine.

The programs run on virtual machines are usually operating systems, which in turn provide the needed services for conventional user programs. Hypervisors are promising candidates as a method for providing security via separation, or isolation of users, each with his own operating system [4,5].

Virtual machine separation is not obtained without significant costs. Complex simulation of I/O is a necessary part of any hypervisor running on a third generation-like architecture. This simulation of all sensitive instructions slows program performance, but more important, it requires complexities in hypervisor code that can make certification difficult, at least at present. Furthermore, some existing architectures do not have the characteristics necessary to allow virtualization, and so machine-specific hardware modifications are necessary [3]. Finally, a hypervisor must be written and certified for each model of the hardware.

While the intent of separation is promising, the virtual machine approach encounters difficulties in retrofitting because of the frequent need for special hardware, inefficiencies due to complex simulation, and the overall complexity of the hypervisor code.

## ENCAPSULATICN

A simplier approach than the above is to remove the hypervisor from the host machine, and construct externally-enforced user separation and device access controls at the peripheral devices, rather than within core memory. Such a technique both removes the high costs of simulation, and eliminates the related complexity that can make virtual machines unattractive from the point of view of certification. A description of the <u>encapsulatior unit</u> follows.

To a currently existing configuration, a small minicomputer is added that controls several large switches. These switches are connected <u>at the device</u> to tape drives, disks, and other similar units of original equipment. These switches are physically placed in the read/write circuit of the devices and allow the mini to enable or disable these peripheral cevices. As in the virtual machine case, each encapsulatec user program will run with its own operating system. Depending on which operating system is currently in control of the production CPU, the mini sets the device switches accordingly, so that the operating system can physically access only those devices that the mini has allowed. For many existing peripheral devices, these switches are already present.

How does the mini know which operating system is running, and how are operating systems switched? The minicomputer is connected to the original equipment through an I/O port, and, in addition, controls the usual hardware directed initial program load sequence (IPL). To change operating systems, the mini initiates IPL, which is arranged to load a bootload program from the mini through the I/O port. This bootload program first copies out the core image of currently operating software. Next, the mini switches the mode of the devices connected to the original computer. Finally, the bootload program swaps in the core image of another operating system. The swap process should only take a few seconds, and the newly loaded system can continue running immediately. The process is simple and straightforward, two of its virtues.

There are a number of points worth making about the encapsulation unit. First, all mechanisms and code responsible for security are isolated in the minicomputer, leading to simplicity and relative ease of certification. Full use of the original hardware is available to user operating systems and application programs. The complex simulation routines of the virtual memory approach are eliminated.

Second, since the mini has an I/O port to the original equipment, it may well be practical for the mini to control spooling of input and output. Currently existing I/O devices, such as printers, card readers, and punches, can be disconnected from the original hardware and connected to the mini. In this way, the installation obtains spooling at little extra cost, and hence may be able to cut down on certain other expensive resources. Also, if operating systems are swapped relatively frequently, this variation eliminates the necessity of switching card decks, printer paper, ribbons, and the like in synchrony. The spooling requires the mini to have its own disk for temporary storage.

Third, the system can be extended to allow read-only sharing. This can be achieved by allowing the mini to set the device in one of three modes: off, read-only, or read/write. This extension enables the sharing of common operating systems and libraries. In addition, whenever a partial ordering of security levels exists, a level n security system can be allowed read-only access to level k disk drive spindles, for all $k \leq n$. Of course if one operating system is interrupted while in the process of updating a file on a device that is available in read-only mode to another system, that second system uses the file at its own risk. This situation should not be bothersome, since in many installations the sharing of files among operating systems is used primarily with regard to common code and data, which are infrequently updated. As a further refinement to the disk switch, physically-enforced "mini-disks" can be created by introducing a simple cylinder address relocation register setable by the minicomputer.

Fourth, to minimize the context that must be saved and to simplify the swapping program, operating systems should only be swapped when in a "standard state". Peripheral devices should be in a quiescent state, and any registers or data that might be destroyed in the bootload process should be saved. This requirement in no way effects the security of the system. If a system does not enter the "standard state" before being swapped out, it may not be able to be restarted when it is subsequently swapped in.

Other embellishments might include:

- A hard-wired bypass switch to allow peripheral devices normally connected to the mini to be connected directly to the original system for maintenance;

- A cross-bar switch to allow peripherals to be physically connected to different I/O ports for systems that lack hardware or operating system flexibility in assigning devices;

- Separate operator's consoles attached to the mini for each operating system for ease of operation;

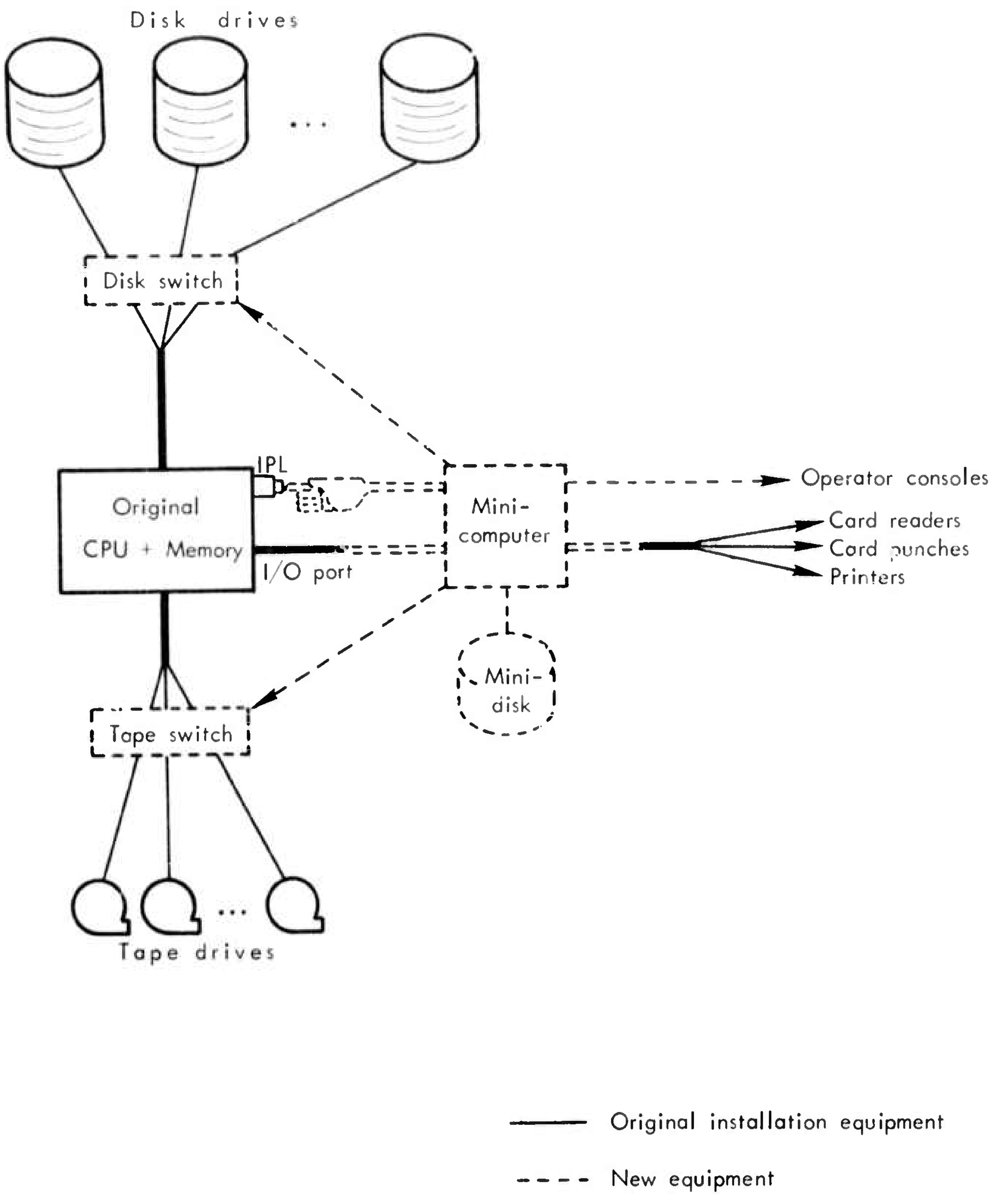

FIGURE 1. SECURITY ENCAPSULATION UNIT

- A security status panel which displays the current security level, device assignments, etc. of the system.

An encapsulation hardware configuration is shown in Figure 1.

## HARDWARE COSTS

Hardware costs for the encapsulation unit are estimated to be approximately $53,000. Figures below are based on current commercially available equipment.

| | | | |
|---|---|---|---|
| 1 | Mini Computer CPU with 16K memory | | $11,400 |
| 3 | Consoles with controllers | | $ 6,000 |
| 2 | Switches with bus mounting | | $ 900 |
| 1 | Disk with controller | | $11,000 |
| 1 | High-speed paper tape reader (for maintenance of mini) | | $ 4,000 |
| 1 | Channel Interface | (estimated) | $10,000 |
| 1 | Channel simulator | (estimated) | $10,000 |
| | | approx. | $53,000 |

Not included in the hardware cost estimates are:

a) remote IPL links;

b) mini bypass switch for I/O devices (card reader, punch, etc.);

c) security status panel.

## SOFTWARE COSTS

### Program Verification

In addition to the hardware, the code in the minicomputer that makes security decisions must be proven correct. Security assertions must be constructed and program verification techniques applied. The task requires a significant effort, but it is essentially a one-time cost. The amount of code that must be verified is of a size and order of complexity within the limits of already demonstrated ability. Roughly two-man years of work by highly competent professional personnel is required to perform the necessary proofs, given the state of currently available verification support tools [1]. Thus, the construction of the necessary verified minicomputer code is a practical task.

### Operating System Modification

Because of certain practical considerations, additional software costs may result from modification of the original operating system. These modifications fall into two categories: those necessary for system quiescence, and those for read-only sharing.

As stated previously, systems should be swapped when in a "standard state" to minimize the context that must be saved by the bootload program. Thus, the operating system must be modified to include routines to quiesce and restore I/O and to save and restore registers or data that would be destroyed in the bootload process. For most operating systems, this modification is trivial. In OS/360, for example, the only changes necessary are the inclusion of a software setable switch which is tested in the channel restart routine, and a small routine to save and restore the first few words of memory.

Software modifications may also be necessary to support read-only sharing. The system must include, or be modified to include, the ability to control the allocation of logical files on physical devices. Otherwise the system might try to allocate a file on a read-only peripheral. Also, some operating systems, such as DEC 10/50, Tenex, and GCOS, require write access to all peripherals since they store information such as the date of last read and read-only lock bits with the physical file. This software would have to be disabled for read-only peripherals in these systems.

## OPERATING COSTS

There will be some lost storage on the original equipment due to duplication of some operating system code and data. In addition, if a job of a given security classification needs more I/O units than are currently dedicated to it, manual changes may be required, slowing the swap between systems and wasting computer time.

## COST GAINS

There are a number of cost savings.

1. Spooling may be provided nearly free.

2. More freedom is gained in scheduling. An operating system running at level x can be interrupted for a high priority job of level y where $y \neq x$.

3. The security provided by encapsulation is insensitive to operating system type or version. Installation modifications or updates can be made freely and have no effect on the security of the system.

4. Once a machine has been secured for a given operating system, the incremental cost for securing a different operating system for that same machine is small, limited to the changes mentioned above in Operating System Modification. So, for example, after an installation has encapsulated IBM's OS/360, securing IBM's DOS/360 is likely to be inexpensive.

## CONCLUSION

Encapsulation provides many of the benefits of the virtual machine approach, without most of the headaches involved in retrofitting, and the cost appears economical.

The usual lost time currently devoted to scrubbing one level security system to prepare for another is significantly decreased. Most important however, _encapsulation provides a certifiable, reliable means for multilevel computer security on existing systems, possibly ending the retrofit problem_.

## REFERENCES

1. Good, D. 1973 (Oct.). Private Communication. At USC/Information Sciences Institute.

2. International Business Machines Corp. 1969 (June). CP-67/CMS, Program 360D-05.2.005. Hawthorne, New York.

3. Popek, G., and Goldberg, R. 1973. "Formal Requirements for Virtualizable Third Generation Architecture," ACM/SIGOPS Fourth Symposium on Operating Systems Principles, Oct. 15-17, 1973. Yorktown Heights, New York.

4. Popek, G., and Kline, C. 1973. "Verifiable Secure Operating System Software". (Submitted for publication.)

5. Weissman, C. 1973 (Oct.). Private Communication. At System Development Corporation, Santa Monica, California.