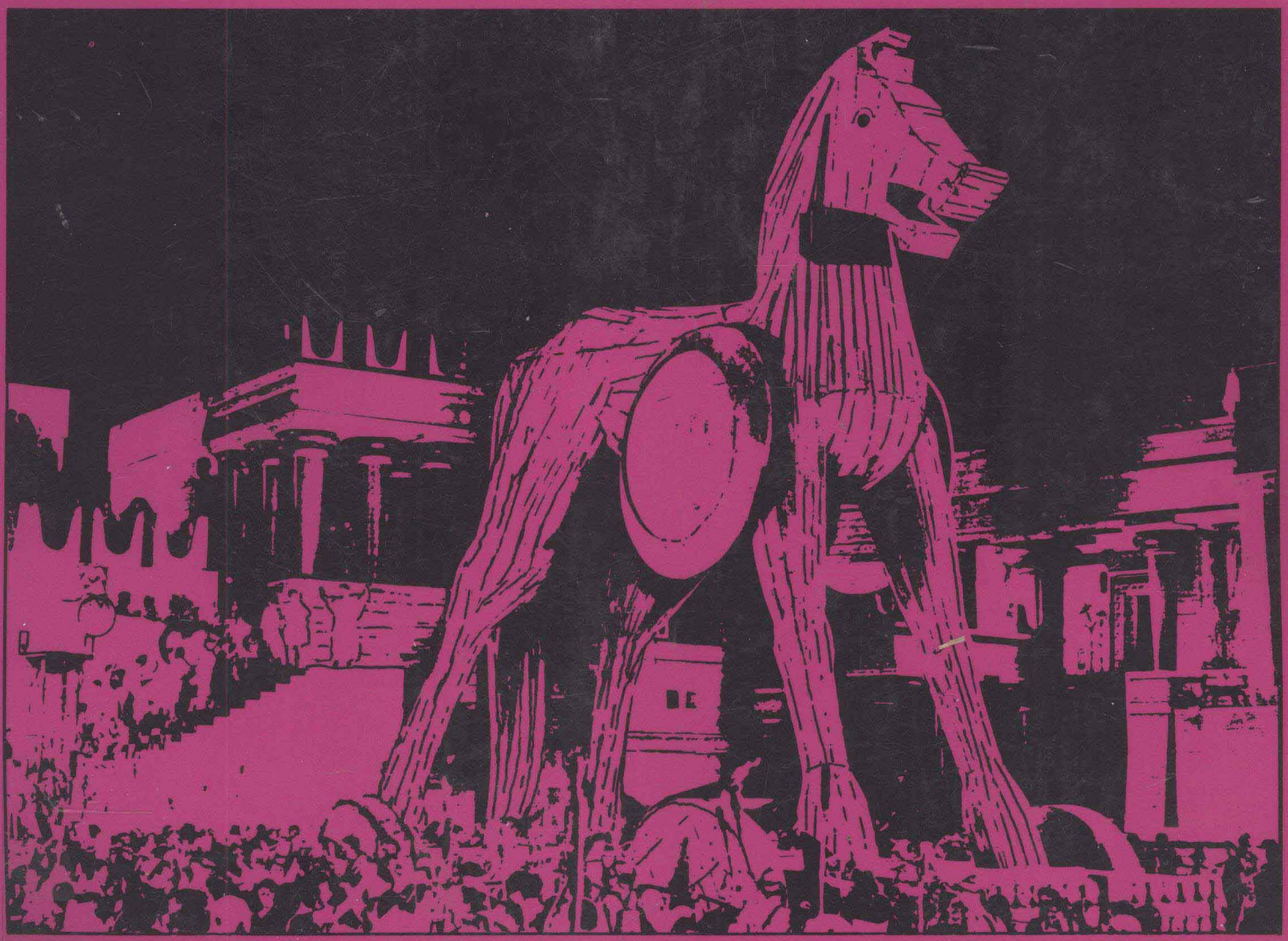
Proceedings

1990 IEEE Computer Society Symposium on

Research in Security and Privacy

May 7-9, 1990 Oakland, California

Sponsored by the
IEEE Computer Society
Technical Committee on Security and Privacy
in cooperation with
The International Association for Cryptologic Research (IACR)



IEEE Computer Society Press

Institute of Electrical and Electronics Engineers, Inc.

Proceedings

1990 IEEE Computer Society Symposium on

Research in Security and Privacy

May 7-9, 1990 **Oakland, California**

Sponsored by the
IEEE Computer Society
Technical Committee on Security and Privacy
in cooperation with
The International Association for Cryptologic Research (IACR)

IEEE Computer Society Press
Los Alamitos, California

Washington ● Brussels ● Tokyo

The papers in this book comprise the proceedings of the meeting mentioned on the cover and title page. They reflect the authors' opinions and are published as presented and without change, in the interests of timely dissemination. Their inclusion in this publication does not necessarily constitute endorsement by the editors, the IEEE Computer Society Press, or The Institute of Electrical and Electronics Engineers, Inc.

Published by

IEEE Computer Society Press
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1264

Cover illustration by Jack I. Ballestero

Printed in the United States of America

IEEE Computer Society Order Number 2060
Library of Congress Number 87-655763
IEEE Catalog Number 90CH2884-5
ISBN 0-8186-2060-9 (paper)
ISBN 0-8186-6060-0 (microfiche)
ISBN 0-8186-9060-7 (case)
SAN 264-620X

Additional copies can be ordered from:

IEEE Computer Society Press
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1264

IEEE Computer Society
13, Avenue de l'Aquilon
B-1200 Brussels
BELGIUM

IEEE Computer Society
Ooshima Building
2-19-1 Minami-Aoyama,
Minato-Ku
Tokyo 107, JAPAN

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

Proceedings

1990 IEEE Computer Society Symposium on

Research in Security and Privacy

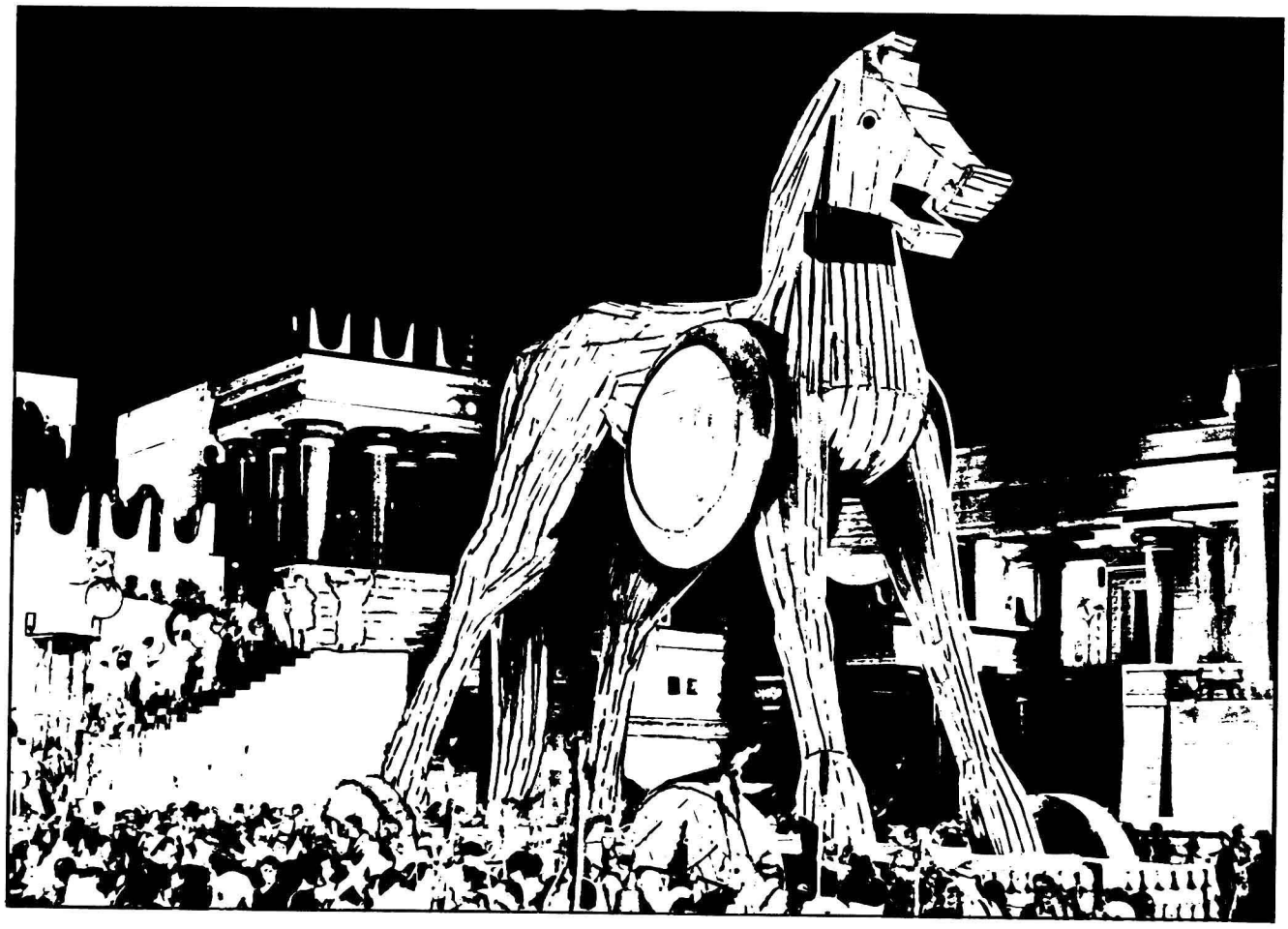
Proceedings

1990 IEEE Computer Society Symposium on

Research in Security and Privacy

May 7-9, 1990 **Oakland, California**

Sponsored by the
IEEE Computer Society
Technical Committee on Security and Privacy
in cooperation with
The International Association for Cryptologic Research (IACR)



IEEE Computer Society Press

Institute of Electrical and Electronics Engineers, Inc.

Message from the Program Chairs

It is with great pleasure that we welcome you to the 1990 IEEE Symposium in Research on Security and Privacy, sponsored by the IEEE Technical Committee on Security and Privacy, in cooperation with the International Association of Cryptologic Research (IACR). Throughout the past eleven years, these symposia have focused on both theoretical and practical research results. The topics addressed this year reflect the community's deepening interest in a wide range of areas, including database security, information flow, access control, integrity, authentication, auditing and intrusion detection, verification, and covert channels.

A significant sign of the maturing of our field is the commercial commitment to security evident from the half dozen papers from Digital Equipment Corporation describing various aspects of their integrated security architecture. This heralds a new era for computer security. In this new era, vendors will routinely address security in all their products, and users will come to expect security from their products just as they now expect user-friendliness and performance.

Fully a quarter of the papers submitted to the conference dealt with database security. The papers you see in the program manifest the recent blossoming of research in this area. These nine papers reflect the breadth of this subfield, covering such aspects as object-oriented systems, statistical inference, aggregation, polyinstantiation, security management, conflicts with integrity, concurrency controls, and data modeling.

This year there were more papers submitted than ever before, from North America, Europe, Asia and Australia, and we had the difficult job of choosing among many excellent papers. We could accept only a third of those submitted, and as a result many qualified papers were necessarily excluded from the program. For assisting us in making our selections, we are indebted to the reviewers who form our program committee. Beyond merely recommending papers for the conference, the reviewers perform the important and often unacknowledged and anonymous service of improving the quality of the papers that are accepted and of contributing to the caliber of the research of the authors whose papers are rejected. We extend our wholehearted thanks to our reviewers.

But program chairs and committees do not a program make. The quality of the conference rests primarily with the authors who take the time and effort to document their research and submit their papers here for consideration. We recognize the enormity of that task and also the wide-ranging and high-quality research represented by the authors of the submitted papers. We are honored to have a part in bringing these technical papers to the community.

In addition, we gratefully acknowledge the conference organizers, especially Deborah Downs and Daniel Schnackenberg, for their hard work in the nuts and bolts of making the conference happen. Finally we thank you, all the conference participants, not only for coming to the conference, but also for the contribution to the field that your attendance here represents.

Deborah M. Cooper **Teresa F. Lunt**
Program Co-Chairs

General Chair

Deborah Downs
Aerospace Corporation

Vice Chair

Daniel Schnackenberg
Boeing Aerospace Corporation

Program Co-Chairs

Deborah Cooper
Unisys Corporation
Teresa Lunt
SRI International

Program Committee

Jim Anderson, *JPA Co.*
Dave Bailey, *LANL*
Terry Vickers Benzel, *TIS*
Tom Berson, *Anagram Labs*
Earl Boebert, *SCTC*
Martha Branstad, *TIS*
Steve Crocker, *TIS*
Nigel Day, *TopExpress*
Dorothy Denning, *DEC*
John Dobson, *U Newcastle*
Christi Garvey, *TRW*
Morrie Gasser, *DEC*
Virgil Gligor, *U Maryland*
Grace Hammonds, *AGCS*
Tom Hinke, *TRW*
George Jelatis, *U Minnesota*
Paul Karger, *DEC*
Matt Kaufmann, *CLI*
Tanya Korelsky, *ORA*
Carl Landwehr, *NRL*
Ted Lee, *TIS*
Steve Lipner, *DEC*
Kevin McCurley, *Scandia NL*
John McHugh, *CLI*
John McLean, *NRL*
Cathy Meadows, *NRL*
Jon Millen, *MITRE*
Robert Morris, *NCSC*
Jerry Myers, *MITRE*
Peter Neumann, *SRI Int'l*
Tom Parenty, *Sybase*
Sig Porter, *SNPorter*
Rainer Rueppel, *R3*
Marv Schaefer, *TIS*
Roger Schell, *Gemini*
Emilie Siarkiewicz, *RADC*
Gary Smith, *Nat'l Defense U*
Brian Snow, *NSA*
Elizabeth Sullivan, *Amhdahl*
Mario Tinto, *NCSC*
Steve Walker, *TIS*
Bill Wilson, *Arca*
Ray Wong, *Oracle*
Yacov Yacobi, *Bellcore*

Table of Contents

Secure Systems I

Chair
S. Lipner

A VMM Security Kernel for the VAX Architecture

Paul A. Karger, Mary Ellen Zurko, Douglas W. Bonin, Andrew H. Mason, and Clifford E. Kahn*

Digital Equipment Corporation
Secure Systems Development
85 Swanson Road (BXB1-1/D03)
Boxborough, MA 01719-1326

Abstract

This paper describes the development of a virtual-machine monitor (VMM) security kernel for the VAX architecture. The paper particularly focuses on how the system's hardware, microcode, and software are aimed at meeting A1-level security requirements while maintaining the standard interfaces and applications of the VMS and ULTRIX-32 operating systems. The VAX security kernel supports multiple concurrent virtual machines on a single VAX system, providing isolation and controlled sharing of sensitive data. Rigorous engineering standards were applied during development to comply with the assurance requirements for verification and configuration management. The VAX security kernel has been developed with a heavy emphasis on performance and on system management tools. The kernel performs sufficiently well that all of its development is now carried out in virtual machines running on the kernel itself, rather than in a conventional time-sharing system.

1 Introduction

The VAX security kernel project is a research effort to determine what is required to build a production-quality security kernel, capable of receiving an A1 rating from the National Computer Security Center. A production-quality security kernel is very different from the many research-quality security kernels that have been built in the past, and this research effort has been primarily aimed at identifying the differences and their cost in development effort and in kernel complexity.

This paper describes how the VAX security kernel meets its five major goals:

- Meet all A1 security requirements.
- Run on commercial hardware without special modifications other than microcode changes for virtualization.
- Provide software compatibility for applications written for both the VMS and ULTRIX-32 operating systems.
- Provide an acceptable level of performance.
- Meet the requirements of a commercial software product.

The VAX security kernel is a research effort. Digital Equipment Corporation makes no commitment to offer it as a product.

2 Kernel Overview

The VAX security kernel is a virtual-machine monitor that runs on the VAX 8530, 8550, 8700, 8800, and 8810 processors.[1] It creates isolated virtual VAX processors, each of which can run either the VMS or ULTRIX-32 operating system. If desired, virtual machines running each of the operating systems can run simultaneously on the same computer system.[2] The VAX architecture was not virtualizable, and therefore extensions were made to the architecture and to the processor microcode to support virtualization. (See Section 3.2.)

Figure 1 shows a typical VAX security kernel configuration. While the VAX security kernel is a VMM, it is primarily a security kernel. Therefore, certain features traditionally seen in VMMs, such as self-virtualization or debugging of one VM from another, have been omitted to reduce kernel complexity.

*This paper presents the opinions of its authors, which are not necessarily those of the Digital Equipment Corporation. Opinions expressed in this paper must not be construed to imply any product commitment on the part of the Digital Equipment Corporation.

[1] The VMM does not run on VAX 8820, 8830, or 8840 processors, due to microcode and console differences.

[2] At least one virtual machine must always run the VMS operating system, to carry out certain system management functions.

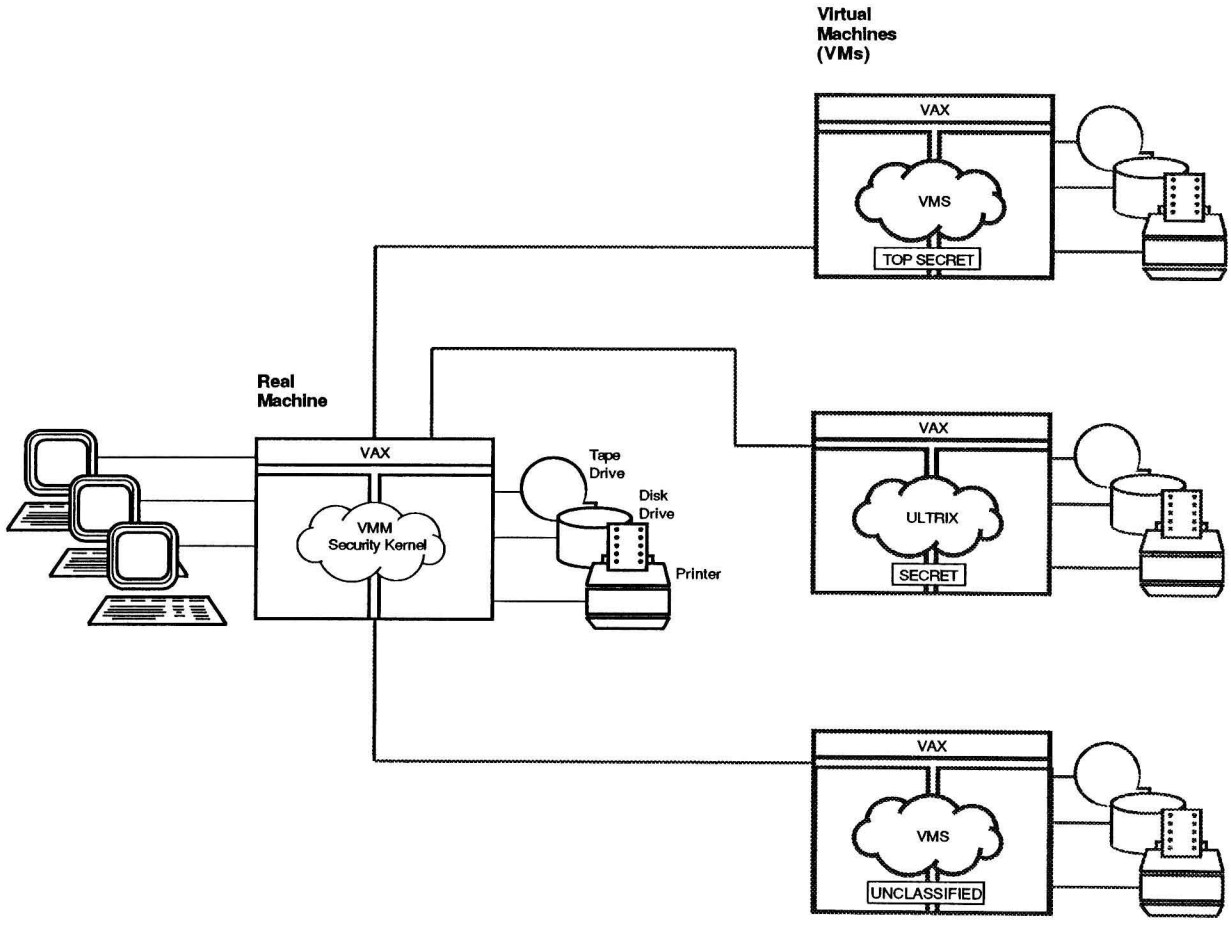


Figure 1: VAX VMM Security Kernel Configuration

The VAX security kernel applies both mandatory and discretionary access controls to virtual machines. Each virtual machine is assigned an *access class* that consists of a *secrecy class* and an *integrity class*, similar to those in the VMS Security Enhancement Service (VMS SES) [5]. The secrecy and integrity classes are based on the Bell and LaPadula security [2] and Biba integrity [4] models, respectively. The VAX security kernel also supports access control lists (ACLs) on all objects, similar to those in the VMS operating system [14].

The VMM security kernel is *not* a general purpose operating system. The principal subjects and objects are virtual machines and virtual disks, rather than conventional processes and files. That is the inherent difference between a VMM and a traditional operating system. Processes and files are implemented within the virtual machines by either the VMS or ULTRIX-32 operating systems.

The VAX security kernel can support large numbers of simultaneous users.[3] All software development of the VAX security kernel is now carried out on several virtual machines running on the VMM on a VAX 8800 system. On a typical day, about 40 software engineers and managers are logged in running a mixed load of text editing, compilation, system building, and document formatting. The system provides adequate interactive response time and is sufficiently reliable to support an engineering group that must meet strict milestones and schedules. As far as we know, the VAX security kernel is the first security kernel to support its own development team. The Multics Access Isolation Mechanism [36] was developed on Multics itself, but Multics with AIM was not a security kernel and only received a B2 rating.

The VAX security kernel is currently in the Design Analysis Phase with the National Computer Security Center (NCSC) for an A1 rating. It is being formally specified in Ina Jo and formal proofs are being done on the specifications.

[3]Exact numbers depend on the precise hardware configuration.

3 Design Approach

This section describes several of the design choices in the VAX security kernel, including details about the virtual ma-

chine approach to security kernels, virtualizing the VAX architecture, subjects and objects, access classes, our layered design, and other software engineering issues.

3.1 Virtual Machine Approach

The choice to build the VAX security kernel as a VMM was driven by two goals: to maintain compatibility with existing software written for the VAX architecture and to keep software development and maintenance costs to a minimum.

Digital Equipment Corporation began plans to enhance the security of the VAX architecture in mid-1979. Our initial effort was the design of security enhancements to the VMS operating system, first prototyped in 1980 and available today in the base VMS operating system and in the VMS Security Enhancement Service [5].

At the time of the initial prototype of the VMS security enhancements [16], Digital considered a traditional kernel/emulator security kernel to support VMS applications. However, it quickly became clear that the software development costs of a VMS emulator would be comparable to the cost of development of the VMS operating system itself. Worse still, the emulator would have to track all changes made to the VMS operating system, resulting in ongoing costs that would be unacceptably high for the limited market for A1-secure systems. The kernel/emulator system could not replace the existing VMS operating system because its performance would not be as good, and it would likely be export controlled. Furthermore, the growing demand for UNIX-based software would force development of a UNIX emulator at still more development cost.

To resolve these development cost and compatibility problems, we chose a VMM security kernel approach. A VMM security kernel presents the interface of a computer architecture that is comparatively simple and not subject to frequent change. Thus, the VAX security kernel presents an interface of the VAX architecture [21] and supports both the VMS and ULTRIX-32 operating systems with relatively few modifications.

The idea of a VMM security kernel is not a new one. Madnick and Donovan [22] first suggested the merits of VMMs for security, and Rhode [30] first proposed VMM security kernels. From 1976 to 1982, Systems Development Corporation (now a division of the UNISYS Corporation) built a kernelized version of IBM's VM/370 virtual-machine monitor, called KVM/370 [12]. While the design of the VAX security kernel is very different from KVM/370, we have applied some of the lessons learned in the KVM/370 project [11]. Section 7 compares the VAX security kernel with KVM/370. Gasser [10, Section 10.7] provides more detail on some of the trade-offs between a VMM security kernel approach and a kernel/emulator approach.

3.2 Virtualizing the VAX

The requirements for virtualizing a computer architecture were specified by Popek and Goldberg [26]. In essence, they require that all sensitive instructions and all references to sensitive data structures trap when executed by unprivileged code. A *sensitive* instruction or data structure is one that either reveals or modifies the privileged state of the processor.

3.2.1 Sensitive Instructions

Unfortunately, the VAX architecture does not meet Popek and Goldberg's requirements. Several instructions, including Move Processor Status Longword (MOVPSL), Probe (PROBEx), and Return from Exception or Interrupt (REI) are sensitive, but unprivileged. Furthermore, page table entries (PTEs) are sensitive data structures that can be read and written with unprivileged instructions.

As a result, we made a number of extensions to the VAX architecture to support virtualization. In particular, we added a VM bit to the processor status longword (PSL) that indicated whether or not the processor was executing in a virtual machine. A variety of sensitive instructions were changed to trap based on the setting of the VM bit, so that the VMM security kernel could emulate their execution. Space does not permit a full discussion of the instruction changes, but some details are discussed by Karger, Mason and Leonard [18].

3.2.2 Ring Compression

The most significant and security-relevant change to the VAX architecture was to virtualize protection rings. In the past, only processors with two protection states (such as the IBM 360/370 architecture) had been virtualized. Goldberg [13, section 4.3] described the difficulties of virtualizing machines with protection rings and therefore more than two protection states. He proposed several techniques for mapping ring numbers, some in software and one with a hardware ring relocation register, but he recognized that none of his techniques were satisfactory. His software techniques broke down because the physical ring number remained visible, and his hardware ring relocation technique broke down because virtualizing a machine with N rings always required N+1 rings.

Since the VMS operating system uses all four of the protection rings of the VAX architecture, it was essential that we develop a new technique for virtualization of protection rings. That technique is called *ring compression*.

Figure 2 shows how the protection rings of a virtual VAX processor are mapped to the rings of a real VAX processor. Virtual user and supervisor modes map to their real counterparts, but virtual executive and kernel modes both map to real executive mode. The real ring numbers are concealed from the virtual machine's operating system (VMOS) by three extensions to the VAX architecture: the addition of the VM bit to the PSL (described in Section 3.2.1), the addition of a VM processor status longword register (VMPSL), and the modification of all instructions that could reveal the real ring number. Those instructions either trap to the VMM security kernel for emulation or obtain their information from

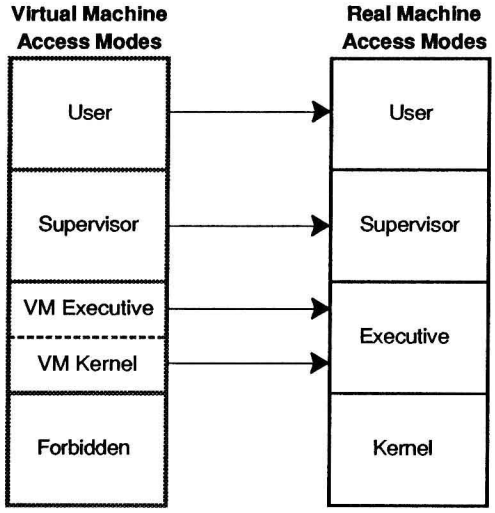


Figure 2: Ring Compression

the VMPSL, which contains the virtual ring number rather than the real ring number. Additional details can be found in Karger, Mason and Leonard [18].

Ring compression also requires that the security kernel change the memory protection of pages belonging to virtual machines so that their kernel-mode pages become accessible from executive mode. This change of memory protection could adversely affect security within a given virtual machine because the virtual machine's kernel mode is no longer fully protected from its executive mode.

For the two operating systems of interest to the VAX security kernel, there is no effective loss of security within the virtual machines themselves, although there is a loss of robustness against potentially bug-laden executive mode code. Fortunately, the VMS operating system grants all programs that run in executive mode the right to change mode to kernel at will and uses the the kernel/executive mode boundary only as a reliability mechanism. Furthermore, the ULTRIX-32 operating system does not use executive mode at all.

Of course, the compression of kernel and executive modes in the virtual machines in no way compromises the security of the VMM, as the security kernel runs only in real kernel mode, and no virtual machine ever is granted access to real kernel mode pages. If there were some other VAX operating system that actually used all four rings for security purposes, it would lose some of its own security, much as IBM operating systems lose some of their security when run in VM/370. However, no such operating systems exist for the VAX architecture.

3.2.3 I/O Emulation

Traditional virtual-machine monitors, such as IBM's VM/370, have virtualized not only the CPU, but also the I/O hardware. Virtualization of the I/O hardware allows the VMOS to run essentially unmodified. Virtualization of the VAX I/O hardware is particularly difficult because its I/O devices are programmed by reading and writing control and status registers (CSRs) that are located in a region of physical memory called I/O space. This type of I/O originated on the PDP-11 series of computers and caused performance difficulties in the UCLA PDP-11 virtual-machine monitor [27] because the VMM must somehow simulate every instruction that manipulates a CSR. Vahey [33] proposed a complex hardware performance assist, but such a device would add excessive complexity and development cost to the VAX security kernel.

Instead, the VAX security kernel implements a special I/O interconnection strategy for virtual machines. The VAX architecture [21] does not specify how I/O is to be done, and different VAX processors have implemented very different I/O interfaces. The VAX security kernel I/O interface is a specialized kernel call mechanism, optimized for performance, rather than traditional CSR-based I/O. In essence, a virtual machine stores I/O-related parameters (such as buffer addresses, etc.) in specified locations in its I/O space, but no I/O takes place until the virtual machine executes a Move to Privileged Register (MTPR) instruction to a special kernel call (KCALL) register. This MTPR traps to security kernel software that then performs the I/O. Thus, the number of traps to kernel software is dramatically less than would be required for CSR emulation.

This special kernel I/O interface means that special untrusted virtual device drivers had to be written for both the VMS and ULTRIX-32 operating systems, but this effort was no more than is typically required to support a new VAX processor, a small number of engineer-years.

Because the virtual VAX processors have an I/O interface different from that of any existing VAX processors, the VAX security kernel does not fall into any of Goldberg's traditional categories of VMMs. Goldberg [13, pp. 22-26] defines a Type I VMM as a VMM that runs on a bare machine. He defines a Type II VMM as a VMM that runs under an existing host operating system. Goldberg [13, section 3.3] also defines a Hybrid Virtual-Machine Monitor as one in which all supervisory-state instructions are simulated, rather than just the privileged instructions. The VMM security kernel is essentially a cross between a self-virtualizing Type I VMM for all non-I/O instructions and a Hybrid Virtual-Machine Monitor for I/O instructions.

3.2.4 Self-Virtualization

As we designed the extensions to the VAX architecture, we ensured that the architecture would permit *self-virtualization*. Self-virtualization is the ability of a virtual-machine monitor to run in one of its own virtual machines and recursively create second-level virtual machines. Self-virtualization is very useful for developing and debugging the virtual-machine monitor itself, but it is of little value to actual users. Since self-virtualization would have added sig-

nificant complexity to the Trusted Computing Base (TCB), no software support has been done.[4]

3.3 Subjects

There are two kinds of subjects in the VAX security kernel, users and virtual machines (VMs). A user communicates over the trusted path with a process called a *Server*. Servers are trusted processes, but unlike the trusted processes in other systems such as KSOS-11 [3], servers run only within the kernel itself. User subjects cannot run user-written code; servers execute only verified code that is part of the TCB.

The powers of a server are determined by:

- The user's minimum and maximum access class. (See Section 3.5.)
- The terminal's minimum and maximum access class.
- The user's discretionary access rights.
- The user's privileges. (See Section 3.6.)
- The privileges exercisable from the terminal.

A virtual machine is an untrusted subject that runs a VMOS. A user interacts with the VMOS in whatever fashion is normal for that operating system, for example, by logging into that VMOS and issuing commands. A user may write and run code inside a VM and even penetrate the VMOS, all without affecting the security of other virtual machines or the security kernel itself. At worst, a penetrated virtual machine could only affect other virtual machines with which it shared disk volumes.

On login to the security kernel, the VMM establishes a connection between the user's terminal line and the user's Server, called a *session*. When the user wants to use some virtual machine, the user issues the `CONNECT` command to his or her Server, specifying the name of that VM. If the connection is authorized, the system suspends the user's existing session with the Server and establishes a new session between the user's terminal line and the requested virtual machine. Thus, the Servers and the VMs have distinct identities and distinct security attributes.

Virtual machines may be run in a single-user mode to provide maximum individual accountability. Alternately, they can be run in a multi-user mode. In such a case, individual accountability might be achieved by running a VMOS with at least a C2 rating, as suggested by the proposed Trusted VMM Interpretation [1] of Trusted Information Systems, Inc.

Virtual machines can also be treated as objects because a user may request that the TCB provide a connection between the user's terminal and some VM. For this operation, the user is the subject and the VM is the object.

3.4 Objects

The VAX security kernel supports a variety of objects including real devices and volumes and security kernel files.

One group of objects comprises the real devices on the system: disk drives, tape drives, printers, terminal lines, and single access-class network lines. As these devices can contain or transmit information, access to them must be controlled by the TCB. Another object is the primary memory that is allocated to each VM when it is activated.

Disk and tape volumes are also objects. The contents of some disk volumes are completely under the control of a virtual machine. They may contain a file system structure or just an arbitrary collection of bits, depending on the method used by the VMOS to access the volume. Such volumes are called *exchangeable volumes* because they may be exchanged with other computer systems running conventional operating systems. Other disk volumes contain a VAX security kernel file structure and are called *VAX security kernel volumes*. These volumes must not be directly accessed by a VMOS or exchanged with other systems, as an untrusted subject could subvert the kernel's file system or read information to which it was not entitled. The VAX security kernel does not provide trusted tape volumes; all tape volumes are exchangeable.

VAX security kernel volumes contain VAX security kernel files that are organized as a flat file system. VAX security kernel files are used for a variety of purposes in the system and are considered objects by the TCB. One use for VAX security kernel files is to hold long-term system databases such as the audit log and the authorization file. These files are considered part of the TCB and, with the exception of the audit log, error log and crash dump files, cannot be directly referenced by virtual machines.

Another use of VAX security kernel files is to create virtual disk volumes, loosely analogous to mini-disks in IBM's VM/370 [23, pp. 549–563]. Mini-disks allow a physical disk to be partitioned, so that one need not dedicate an entire physical disk to a small virtual machine that only requires a small amount of disk space. Such virtual disks may contain the file structure of some VMOS, such as a VMS file structure or an ULTRIX-32 file structure. However, the VMM deals with virtual disks only as a whole. The contents of a virtual disk are all part of a single object as far as the VMM is concerned.

3.5 Access Classes

The VAX security kernel enforces mandatory controls, as required of all A1 systems. Both secrecy and integrity models

[4] The software changes needed for self-virtualization primarily consist of changes to the virtual device drivers described in Section 3.2.3 and some changes in the emulation of certain sensitive instructions. Under the proposed Trusted VMM Interpretation [1], it might even be possible for a self-virtualized security kernel to itself remain A1 rated. To achieve that goal, the first level VMM would map the second level VMM's kernel mode to real executive mode, while the VMs running on top of the second level VMM would have their supervisor, executive, and kernel modes all mapped to real supervisor mode. Of course, as one continues to recursively self-virtualize, one runs out of protection rings at the fourth level VMM, and that VMM would no longer be protected from its virtual machines.

are supported, based on the work of Bell and LaPadula [2] and of Biba [4], respectively. To implement mandatory controls, each kernel subject and kernel object is assigned a sensitivity label, called an *access class*.[5] An access class consists of two components, a *secrecy class* and an *integrity class*. These components are each further divided into a level and a category set. A *secrecy level* is a hierarchical classification. The *secrecy category set* is the set of non-hierarchical secrecy categories that represents the sensitivity of the access class. The integrity level and integrity category set are defined analogously. For compatibility with VMS SES [5], the kernel supports 256 secrecy levels, 256 integrity levels, 64 secrecy categories, and 64 integrity categories.

Given the complex structure of access classes, two definitions must be carefully constructed:

Definition 1 *An access class A is* equal *to an access class B if and only if:*

- *The secrecy level of A is equal to the secrecy level of B,*
- *The secrecy category set of A is equal to the secrecy category set of B,*
- *The integrity level of A is equal to the integrity level of B, and*
- *The integrity category set of A is equal to the integrity category set of B.*

Definition 2 *An access class A* dominates *an access class B if and only if:*

- *The secrecy level of A is greater than or equal to the secrecy level of B,*
- *The secrecy category set of A is an improper superset of the secrecy category set of B,*
- *The integrity level of A is less than or equal to the integrity level of B, and*
- *The integrity category set of A is an improper subset of the integrity category set of B.*

The secrecy and integrity models define that a subject may reference an object depending on the access classes of the subject and object and on the type of reference. A subject may read from an object only if the subject's access class dominates the access class of the object. A subject may write to an object only if the object's access class dominates the access class of the subject.[6] Thus, for example, a virtual machine may mount for read-write access an exchangeable volume only if the VM's access class is equal to that of the volume. However, the VM may mount for read-only access any exchangeable volume where the VM's access class dominates that of the volume.

[5]Some objects, such as terminal lines, may be assigned a range of access classes.

[6]In general, write access is even further restricted; a subject may write to an object only if the subject's and object's access classes are equal. This disallows blind writes to an object that cannot be read.

3.6 Privileges

System managers, security managers, and operators gain their powers by having *privileges*. The privileges allow great flexibility in the assignment of powers and responsibilities, including a measure of two-person control and separation of duties. Privileges restrict access beyond the protections provided by mandatory and discretionary access controls. A privileged user cannot see data that would be otherwise inaccessible. Only the downgrading privileges allow bypassing of access controls, and the use of those privileges is audited.

Most privileges can be exercised only through the trusted path and are called *user privileges*. (See Table 1.) Two privileges can be exercised by virtual machines and are called *virtual-machine privileges*. (See Table 2.)

3.7 Layered Design

The VAX security kernel has been implemented following the strict *levels of abstraction* approach originally used by Dijkstra [8] in the THE system. Janson [15] developed the use of levels of abstraction in security kernel design as a means of reducing complexity and providing precise and understandable specifications. Each layer of the design implements some abstraction in part by making calls on lower layers. In no case does a lower layer invoke or depend upon higher layer abstractions. By making lower layers unaware of higher abstractions, we reduce the total number of interactions in the system and thereby reduce the overall complexity. Furthermore, each layer can be tested in isolation from all higher layers, allowing debugging to proceed in an orderly fashion, rather than haphazardly throughout the system. This type of layering is called out in the requirements for B3 and A1 systems when the NCSC evaluation criteria [7, p. 38] state that, "The TCB shall incorporate significant use of layering, abstraction and data hiding. Significant system engineering shall be directed toward minimizing the complexity of the TCB ..."

The layered design of the VAX security kernel was based heavily on the Multics kernel design work of Janson [15] and Reed [28] and to a lesser extent on the Naval Postgraduate School kernel design [6]. Figure 3 shows a diagram of the layers of the VAX security kernel. The arrows in the diagram indicate how each layer functionally depends on the abstract machine created by lower layers.

Each layer adds specific functions within the security kernel, such that at the security perimeter, the secrecy and integrity models are enforced. The kernel itself is process structured, as described in the summary of the various layers. Unlike many other kernels, all of the trusted processes run within the security perimeter and are included in the formal specifications described in Section 5.4.

HIH The Hardware-Interrupt Handler layer is immediately above the physical VAX hardware and modified microcode. It contains the interrupt handlers for the various I/O controllers and certain CPU-specific code.