Discussed is a methodology for discovering operating system design flaws as an approach to learning system design techniques that may make possible greater data security.

Input/output has been found to be involved in most of the weaknesses discovered by a study team in a particular version of the system.

Relative design simplicity was found to be the source of greatest protection against penetration efforts.

Penetrating an operating system: a study of VM/370 integrity

by C. R. Attanasio, P. W. Markstein and R. J. Phillips

There is a large body of literature relating to computing system security that includes such issues as statements of problems and requirements for secure systems,[1-5] research in the design of secure computing systems,[6-8] efforts to develop techniques for verifying the correctness (and hence impenetrability) of programs,[9] and reports of organized efforts to determine the penetrability of various operating systems[10-11]. The literature on operating system penetration is rather sparse because most of the work has been done under classified auspices[12]. Although the general references given here are a small representative portion of the existing literature, more extensive bibliographies are cited in References 13 and 14.

In this paper we summarize methods and results of a penetration study of the IBM Virtual Machine Facility/370 (VM/370) in the version that was current in February, 1973. We did not attempt to follow changes in the system through subsequent releases. Our goal was to evaluate penetrability in the light of the system architecture, rather than to track a moving target.

The virtual machine operating system,[15-16] VM/370, differs from conventional operating systems in that the interface presented to the user is that described in the Principles of Operation of System/370 and in which each user has the illusion of having a stand-alone computing system consisting of a CPU, main storage,

and I/O devices.[17] VM/370 differs further from conventional systems in that the user is presented with one or more interfaces that are more easily usable than a stand-alone machine, e.g., those defined by high-level languages, or supervisor services that provide functional interfaces to hardware facilities. (VM/370 is a successor to CP-67, which is discussed in References 18 and 19.) The virtual storage capability of System/370 hardware and the software design of VM/370 combine to provide sharp isolation of the individual virtual machines. By sharing virtual devices, data sharing with VM/370 is analogous to the way in which data are typically shared among real machines.

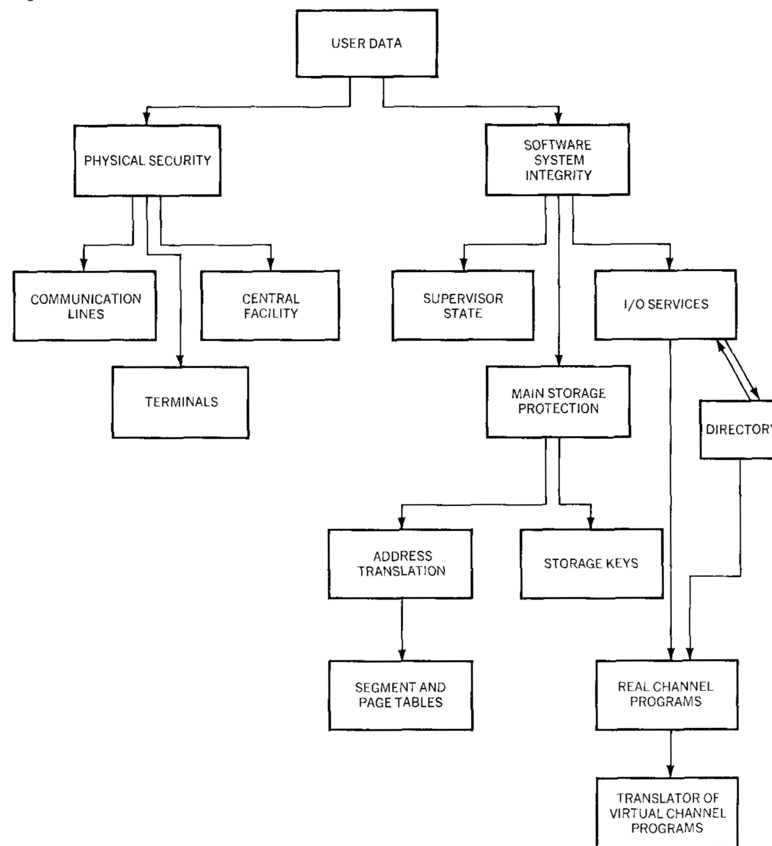
A team of penetrator-analysts was formed to attempt to penetrate VM/370, with the objective of obtaining information to which they were not entitled, such as passwords or data that belonged to other users. Lesser goals were to acquire an unreasonably large share of resources in order to deny service to other users, and to obtain resources but escape accountability. VM/370 was chosen for study to determine whether the substantially different architecture of VM/370 afforded significant security advantages over the architectures of systems previously reported in the literature. It was hoped that by performing a penetration experiment on a virtual machine system, and by combining the knowledge so gained with the available results of previous penetration studies, the researchers would be able to generalize conclusions from the study.

The team of penetrators (the present authors) worked as parts of the user communities of the VM/370 systems running at the IBM Thomas J. Watson Research Center in Yorktown Heights, New York, and at the System Development Corporation in Santa Monica, California. Team members were restricted to the lowest user privilege class, so that any penetrations discovered would be potentially available to all system users. Indeed, this restriction left open the question of penetration by system programmers and computer room personnel. One of our ground rules was that it would be sufficient to demonstrate the possibility of carrying out a penetration, without performing the penetration itself if the case was sufficiently clear. In doubtful cases, a test of the system on the computer was used for resolution. The only tool that was available to the penetration team was a listing of the VM/370 operating system code. Although the general user might not normally have that code in his possession, it is material that is, in principle, available to everyone.

Penetration study methodology

The detection of system vulnerabilities is an act of problem solving. There is no simple, automatic way of finding all significant

Figure 1 Portion of a model of interdependencies of VM/370 control objects



design oversights or implementation errors. Through systematic study of the system, an approach and an attitude developed that enabled the investigators to examine the system with a strong probability of finding vulnerabilities if they existed. No method evolved, however, that guaranteed that all vulnerabilities were found. The method used in this study comprised the following steps: modeling the system control structure; flaw hypothesis generation; flaw hypothesis confirmation; and flaw generalization. This approach was an example of developing and pruning search tree structures as proposed by Newell, Simon, and Shaw[20] for artificial intelligence application.

modeling the control structure

At the start of the study, the penetration analysts had to understand how users interact with the system, what services are provided to them by the system, and what constraints are placed on users. It was also necessary to understand the system structure well enough to recognize the control objects (that is, those parts of the system that control the system access and resource availability) and to have a basic general understanding

of the interrelationships among control objects and how they are used. Control objects may be modules, data items, hardware registers, or data files. They may be accessed through other objects, such as disk packs, channels, or terminals. A model of the system was then constructed in the form of a directed graph that showed the dependencies among control objects.

Figure 1 is an illustrative subset of the actual model in which the primary target is identified as User Data. The two major dependencies of the integrity of user data are physical security (which, in turn, is based on the physical security of the central facility, communication lines, and terminals) and software operating system integrity. Operating system integrity exists only if the supervisor state, main storage, and I/O services are managed correctly. Main storage, as an example, is protected correctly only when two conditions prevail simultaneously: (1) address tables, i.e., segment and page tables, are properly established and protected; and (2) storage keys are used properly. Similar conditions prevail for the branch of the graph that descends from I/O services, with the addition of mutual dependency between I/O services and the directory. The mutual dependency shows that either dependency can be made to fail through a penetration of the other. Mutual dependency illustrates that the dependency of the system objects cannot be represented simply as a tree.

Because of the inherent dynamic complexity of an operating system, one cannot depend entirely on a static graph like Figure 1 to pinpoint relationships and security weaknesses. We found penetrations that involved timing considerations, complex module interaction, and probabilistic, nonrepeatable events.

flaw hypothesis generation

The need for familarity with the basic internal structure of VM/370 is an obvious prerequisite. The participants in this study began with a knowledge of VM/370. As a consequence, very little time was required for this step. Identifying control objects and their relationships, however, required reorientation of this knowledge. To do this, a technique of flaw hypothesis generation was developed. A flaw hypothesis is an unproved assertion that a system weakness exists whereby a control object can be modified or circumvented in ways not intended by its designer, thus constituting a security vulnerability. The generation of flaw hypotheses is a team activity. Specific areas of the operating system are studied from different points of view, and collections of possible security weaknesses are generated for each area.

Each operating system area was examined for one or more of the following characteristics:

- Implicit or explicit resource sharing mechanisms.

- Man-machine interfaces administered by the operating system.
- Configuration management controls.
- Identity-authentication controls.
- Add-on features, design modifications, and design extensions.
- Parameter checking.
- Control of security descriptors.
- Error handling.
- Side effects.
- Parallelism.
- Access to microprogramming.
- Complex interfaces.
- Duplication of function.
- Limits and prohibitions.
- Access to residual information.
- Violation of design principles.

If such characteristics were present, the team hypothesized how the discovered characteristics might weaken the system. The directed graph model of VM/370 was used to identify control objects, the security of which was to be questioned.

Each flaw hypothesis so generated was subsequently evaluated by the team, which estimated the probability that a flaw existed. The team then assessed the potential security vulnerability, should that flaw be exploited. This filtering process eliminated hypotheses that, in the judgement of the team, had a low probability of being confirmed or a low probability of leading to a penetration if confirmed. Those of sufficient interest were documented in the form of "flaw hypothesis study sheets," which identify the flaw, the threatened control object, the potential payoff, and the probability of success.

The flaw-hypothesis method produced a perspective of the system that was different from that of the system designers. It helped

reduce a natural tendency to accept—even unconsciously—implicit assumptions made by the designers and, thereby, the tendency to overlook security flaws. Subjecting every part of the system to such examination helped make the study comprehensive.

flaw hypothesis confirmation

A two-stage detailed inquiry was made for each flaw hypothesis study sheet. The first stage may be characterized as a desk-checking procedure, wherein existing documentation, program logic manuals, and symbolic listings of the system were examined to determine the validity of a flaw hypothesis. Flaw hypotheses that could not be resolved by desk checking were then subjected to the second stage of inquiry—live system testing. Such an investigation of a hypothesis often uncovered flaws of value in an unrelated area.

Details of each live test were peculiar to each hypothesis, but testing generally involved the coding of small, one-shot progams. It was not the intention of a live test to produce an actual penetration, but to show that a flaw existed. (A penetration, as opposed to a flaw, involved a complete program and strategy to exploit a flaw or combination of flaws. A penetration effort was usually larger than a flaw demonstration effort.) A large intellectual investment, but only a small amount of code, was usually required to demonstrate the existence of a flaw. An actual penetration program involved little additional innovation after a flaw was confirmed, but it required large coding investment to exploit the flaw. A penetration effort might include straightforward programs to perform input/output, timing, initialization, set-up, and so forth.

Of the 880 flaw hypotheses generated, 76 warranted detailed study; the remainder were quickly dismissed. Thirty-five flaw hypotheses were confirmed. Nineteen of these flaws required running a one-shot program for confirmation. Two additional flaws were found as byproducts of program tests for other weaknesses.

flaw generation

When a flaw was confirmed, it was documented and further analyzed to determine whether it had been an instance of a more general class of flaws. For example, a flaw may have been that a particular parameter was not being adequately checked in a particular interface. By generalizing the nature of that parameter-checking process, it might be found that many other parameters at other similar interfaces were similarly inadequately screened. By revealing such generic classes of flaws, effective design countermeasures for generic weaknesses could be developed.

Generic flaws served two purposes beyond guiding the penetration study of VM/370 and subsequently exploiting the flaws.

Particular areas of weakness that were discovered became starting points in penetration analyses of other systems, to test whether generic flaws in one system have counterparts in other systems. Also, collections of generic weaknesses outline areas that require particularly careful design, implementation, and testing for systems of the future.

specific methods for VM/370

Online data, system capabilities, and system resource accounting became the control object targets of the VM/370 penetration team. Of these, accounting information was considered to be a subset of online data. Therefore, no effort was made directly toward inducing failures in the VM/370 accounting mechanism. The principal effort was to find design or implementation errors in VM/370 itself that would permit a general user to break out of his virtual machine address space.

The team grouped penetration into the following four categories:

- *Seize control of the entire computing system.* In essence, VM/370 was induced to relinquish control to the virtual machine in supervisor mode, or to reveal the control descriptors that controlled access to data. When a penetrator had this level of privilege, all information in the system was at his disposal.

- *Subvert a particular system mechanism.* This might cause some unauthorized data to be revealed to the penetrator, without his controlling the entire computing system. Often there was an element of chance. The penetration attempt might have to be repeated several times before all the necessary conditions were met, since only some conditions were under the penetrator's direct control.

- *Degrade system performance.* The reduction of the ability of other users to accomplish useful work was generally not perpetrated directly. Techniques for performance degradation, however, were generally found during the study of other flaw hypotheses.

- *Security risks.* Human error or inadequate operating procedures could provide the opening for a penetration. Security risk was not studied extensively because that risk depended on specific installation procedures, personnel compentence, and the degree of concern for security at each installation. Nevertheless, it was considered important that these risk areas be minimized if security was to be achieved. One lapse of security might provide a penetrator with the opportunity to make a change that would persist in the system, and permit continued penetration.

Penetration results

Almost every demonstrated flaw in the system was found to involve the input/output facility (I/O) in some manner. Other elements that contributed to system flaws were concurrent operations, resource allocation, and the human interface.

I/O interface complexities

The support of the virtual I/O interface is the most complex portion of VM/370. Simulation of a single virtual machine I/O instruction might involve the simulation of an entire channel program. For performance reasons, rather than simulate such a channel program one command at a time, VM/370 attempts to compile real channel programs from virtual ones. That is, VM/370 attempts to effect relocation on channel programs statically by deducing the addresses on the devices that are really to be used (i.e., relocation of virtual seek addresses), and by relocating data addresses to their current real main storage addresses. VM/370 then places the translated program in its own address space, where it is not vulnerable to being overwritten by a user program when the channel begins to execute the relocated program.

The VM/370 designers, realizing that not all channel programs could be statically analyzed, barred most instances of self-modifying channel programs. We discovered, however, that many complex actions were still possible with non-self-modifying channel programs. To compound the problems of handling channel programs, the same I/O logic was found to be repeated five times in VM/370, to optimally support five different requirements. Specialized versions of I/O logic support the following situations: virtual spooling support; virtual console support; virtual channel-to-channel adapter support; and a special VM/370 I/O interface. Each variation of I/O support has provided possibilities for different errors.

We found that when the system performed I/O, no address space restrictions were enforced by the hardware features to permit dynamic checking of each byte that was to be modified. The I/O channel acted as an independent parallel processor with independent and unrestricted access to all of main storage. The only hardware feature that could restrict channel access was the storage protection key. Except for shared pages, that key was controlled by the virtual machine and was not used to provide system protection.

Even after omitting self-modifying channel programs from consideration, the structure of System/370 I/O was found to be so complex as to make complete static analysis and translation very difficult. Although they are not described as such in the Principles of Operation for System/370, channel commands are

essentially variable-length commands. A modifier bit in a channel command indicates that the next nonbranch command to be fetched by the channel merely specifies a second area of main storage to be used by the hardware. Only the address and count fields are used by the channel. In this sense, the second command is an extension of the first.

The interpretation of a word by the channel depends on the context of the word, that is, whether it is the first or a subsequent word of a variable-length command. We found that the same word might—during the execution of a channel program—have played the role of a command in its own right as well as acting as the extension of another command by using transfers in the channel program or command skipping in disk channel programs. Thus the System/370 architecture allowed puns in the channel, in that a word's interpretation depended on whether it was received as the leading or trailing portion of a long command. This fact seemed to have been overlooked in the VM/370 static analysis of channel programs, since this possibility was not considered when backward branches occurred in channel programs. When a word had been examined, VM/370 assumed that its meaning remained constant throughout the execution of the channel program. This resulted in several penetrations, some of which enabled the penetrators to seize complete control of the system to access files illicitly, or to deprive other users of certain resources, e.g., I/O channels, by writing nonterminating channel programs.

concurrent operations

VM/370 allowed a virtual machine to execute one or more I/O operations at the same time as CPU instructions were being executed. After an I/O operation had been initiated, the VM/370 architecture would allow the virtual machine to be schedulable. Thus the virtual machine could start another I/O operation on a different channel, execute instructions, or issue system requests. The channels performed, in a sense, as though they were separate parallel processors. They could modify any location in virtual storage while code was being executed. In addition, a user might specify that a virtual machine be schedulable before the processing of console commands had been completed. This would make it possible for a virtual machine to execute programs while command processing for the same virtual machine was in progress by the VM/370 control program.

This capability of concurrent operations for a virtual machine puts a stringent requirement on the control program. Since it would be possible for a virtual machine's storage to be modified at any instant, the control program must never make assumptions based on values that it obtains from the virtual machine's storage.

VM/370 was found to be generally careful to observe this restriction. Selector channel programs were always found to be executed from protected system areas. The spooling and virtual console functions moved the current command word to a protected area for examination. An instruction that caused an interruption would be moved to a table early in its processing. Virtual console commands were moved to a system buffer before processing. These are all instances of strength.

However, two VM/370 features were discovered that permitted a total penetration, and others were discovered that could cause the system to fail. The first case concerned the OS/360 use of self-modifying channel programs in its ISAM access method[21]. To support this feature in a virtual machine, VM/370 had been modified to examine channel programs for the pattern associated with this use of self-modifying code by OS/360. The VM/370 method of handling such channel programs was to execute some commands out of the user's virtual storage, that is, not in VM/370 storage space. As a consequence, a penetrator, mimicking the OS/360 channel program, could modify the commands in his storage before they were executed by the channel, and, thereby, overwrite arbitrary portions of VM/370. This feature, however, was not generally available to the user. It would take a deliberate action by a system administrator to make it available to a specific user. Hence, the support of the OS/360 ISAM access method could be controlled, or, if desired, denied to all users. The second case involved a bizarre interplay of an oversight in condition-code checking, simultaneous CPU and I/O channel program execution, and a retrofit to the basic VM/370 design. With careful timing, these factors could be manipulated to gain a total system penetration.

incomplete design of resource allocation

Design oversights could be exploited to monopololize some system resources. As a result, the system might become less useful to others, even though there might be no penetration, in the sense of gaining control of the machine or accessing private information. For most system resources used by VM/370 in servicing users, the allocation is an "open-shelf" strategy. In general, an interactive system must be capable of responding to rapidly fluctuating demands for services and resources from users. In an interactive environment, therefore, maintaining control of resources—such as working main storage and I/O channels—and the ability to distinguish legitimate demands from malicious ones would be quite difficult. Such a capability was discovered to be beyond the capability of VM/370.

Virtual machines are scheduled on the basis of terminal activity, I/O activity, working-set size, and priority. The then-current implementation allowed a penetrator to misrepresent his activity

and thus be able to obtain a disproportionate share of the CPU cycles. In extreme cases, users might not be serviced at all as a result of such abuse.

human interface

As in any chain of interconnecting components, a computing system is only as secure as its weakest link. And that link may be a human being. Even if all the vulnerabilities described so far were corrected, a system as maintained at a given installation might still be vulnerable to human error or oversight. Many potential flaws involved the human interface, but we found no common element that related these weaknesses in the same manner as I/O was found to be a common factor in the penetrations just described.

The system was found to be unable to protect itself from an operator or system programmer who desired to penetrate the system. Great caution was found to be required during system maintenance to prevent unauthorized system modification or inadvertent loss of information. People doing maintenance might often be overworked and under pressure, and might become negligent in protecting critical data. They might not realize that a new version of a system must be guarded before it becomes operational as well as after it is put into use.

There seemed to be no attempt to protect the system from intentional subversion by an operator. The computer console was the link in the chain that permitted alteration of any location in storage. Operator errors might also lead to penetrations. Since any volume could be mounted on a real computer, it was consistent with VM/370 design to permit any volume to be mounted on a virtual machine. This would give great flexibility, but would make system security dependent on perfect operators. It was our belief that installations should limit operator actions so as to partially compensate for such weaknesses.

Written or telephone communication with the operator was required to schedule drives, channels, dedicated printers, communication lines, etc., that were attached to a virtual machine. Responsibility for detecting and preventing illegal requests for data rested completely with the operator.

The system identifies itself by a standard log-on sequence that a virtual machine can simulate. A terminal may be left unattended while attached to a virtual machine set up to simulate the log-on procedure. The next person to use the terminal, believing that he is interacting with VM/370, may reveal his password to the virtual machine. This can happen when several users share a terminal, which is a common mode of operation. Of course, any user who takes the precaution of not using a pre-established connection avoids this hazard.

Security strengths and weaknesses

strengths

Many types of penetrations that have been successful against conventional operating systems[10,11] are not possible against VM/370. For example, OS/360 and other systems[11,22] have been penetrated because those systems used user-addressable main storage for system control information. Penetrations have also been accomplished because the operating system and its users made use of the same file management system. Such a structure requires great care in its implementation to insure that system operations (which are subject to fewer constraints) are always distinguished from user operations.[11]

The main strength of VM/370 is its simplicity. The System/370 Principles of Operation interface presented by VM/370 does not allow VM/370 to employ user-addressable storage for its own purposes. Therefore, all control information must be in storage that is private to VM/370, and, thus, the first type of penetration just mentioned is not possible. The simplicity of VM/370 and its resultant small size allow it to be essentially resident in main storage, so that its requirements for I/O services are simple and are handled directly, rather than through the data management support that is available to users.

weaknesses

The major security vulnerability of VM/370 is due to the complexity required of the control program in simulating the System/360 interface for input/output. In this area, VM/370 is more complex than conventional operating systems, and input/output was involved in all the penetrations found. Penetrations were possible both because of errors in simulating some of the more improbable channel programs that can be written by a user, and from failure to anticipate certain effects made possible by the simultaneous operation of the CPU and input/output in the support of one user.

Concluding remarks

The virtual machine architecture embodied in VM/370 greatly simplifies an operating system in most areas and hence increases the probability of correct implementation and resistance to penetration. The control program is not concerned with user files or multiple access methods. It simulates a machine with a fixed number of buttons and instructions that require interpretation. The choice of System/370 Principles of Operation as a user interface provides a well understood and well documented interface. The result is a system that is smaller and simpler, with fewer opportunities for error. The simplicity enhances the probability of obtaining security. The exception area is the support of input/output.

Our investigation produced a strong conviction that VM/370 was well implemented and carefully tested, even though the study resulted in several penetrations that were the result of implementation errors. This contradiction is consistent with our observation that no system of the size and complexity of VM/370 is known to be error free. All paths could not be tested, and proof of correctness could not be demonstrated. A penetration of any portion of the system could jeopardize the entire system. Modifications, corrections, and functional enhancements were found to have a higher probability of containing flaws that yielded penetrations than the original core of the system.

The penetration methodology developed by our team has provided a systematic and reasonably comprehensive approach to testing VM/370 security strengths and weaknesses. Although there may remain serious and undiscovered flaws, the methodology provided a high payoff ratio of actual flaws found to potential flaws examined (65%). The method saved much time by preventing the researchers from examining low-probability or redundant pathways. It also provided a means for internal communication. At the same time, the team methodology required laborious hours of intensive readings of listings, system documentation, and program logic manuals. Although many hypotheses were generated, only a small number could be studied in detail. In the process of discarding large numbers of hypotheses, one or more valid ones may have been overlooked. Also, the very fact that a systematic approach was taken introduces the possibility that exposures orthogonal to the paths of investigation indicated by the methodology might more easily remain concealed.

As a final observation on the penetration study of VM/370, we occasionally contemplated the wisdom of attempting to make a virtual machine the same as its real-machine counterpart. In one case, the designers of VM/370 had ignored the operation code in a channel program using data chaining because the hardware appeared to do so. In another case, by making the storage keys freely available to the virtual machine, VM/370 was deprived of an extra measure of protection against I/O errors. Thus, permitting a virtual machine to be able to create virtual copies of itself—although essential for some of the purposes for which virtual machine systems are used—increases the difficulty of making such systems secure.

Acknowledgment

The authors acknowledge the contributions of C. Weissman and R. R. Linde of the System Development Corporation, and L. A. Belady and A. L. Tritter of the IBM Research Division. Their

participation in discussions during this project played an important role in establishing a methodology, setting goals, and speculating on probable weaknesses of VM/370. Referees' comments led us to make useful improvements in this paper.

CITED REFERENCES

1. J. P. Anderson, *Computer Security Technology Planning Study*, ESD-TR-73-51-1, Headquarters, Electronic Systems Division, Hanscom Air Force Base, Massachusetts 01730 (October 1972).
2. H. Weiss, "Computer security, an overview," *Datamation* **20,** No. 1 (January 1974).
3. R. Schell, et al. *Preliminary Notes on the Design of Secure Military Computer Systems*, ESD-MCI-73-1, Headquarters, Electronic Systems Division, Hanscom Air Force Base, Massachusetts 01730 (January 1973).
4. D. Tsichritzis, *Systems Security*, IBM Research Report RC3989, may be obtained from the IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, N. Y.10598.
5. NBS Special Publication 404, *Approaches to Privacy and Security in Computer Systems*, U.S. Department of Commerce, Washington, D.C., 1974.
6. A. Jones, *Protection in Programmed Systems*, Ph.D. Thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania (June 1973).
7. S. E. Madnick and J. Donovan, *An Approach to Information System Isolation and Security in a Shared Facility*, Alfred P. Sloan School of Management, M.I.T., Cambridge, Massachusetts (March 1973).
8. R. Bisbey and G. Popek, *Encapsulation: An Approach to Operating System Security*, Information Sciences Institute, Marina Del Ray, California 90291.
9. G. Popek and C. Kline, "Verifiable secure operating system software," *AFIPS Conference Proceedings, National Computer Conference* **43,** 145–151 (1974).
10. P. Karger and R. Schell, *MULTICS Security Evaluation: Vulnerability Analysis*, NTIS: AD/A001120, may be obtained from the National Technical Information Service, U.S. Department of Commerce, Springfield, Virginia 22151.
11. W. McPhee, "Operating system integrity in OS/VS2," *IBM Systems Journal* **13,** No. 3, 230–252 (1974).
12. J. P. Anderson, *AF/ACS Computer Security Controls Study*, ESD-TR-71-398, Headquarters, Electronic Systems Division, Hanscom Air Force Base, Massachusetts 01730 (November 1971).
13. J. Bergart, *Computer Security, Access Control and Privacy Protection in Computer Systems*, Moore School of Electrical Engineering, University of Pennsylvania (1972).
14. J. Scherf, *Computer and Data Security: A Comprehensive Annotated Bibliography*, Alfred P. Sloan School of Management, M.I.T., Cambridge, Massachusetts (1974).
15. *IBM Virtual Machine Facility/370: Command Language Users' Guide*, IBM Publication GC20-1804, may be obtained from IBM Corporation, Data Processing Division, 1133 Westchester Avenue, White Plains, New York 10604.
16. *IBM Virtual Machine Facility/370: Control Program* (CP) *Logic*, IBM Publication SY20-0880, may be obtained from IBM Corporation, Data Processing Division, 1133 Westchester Avenue, White Plains, New York 10604.
17. *IBM System/370 Principles of Operation*, IBM Publication GA22-7000, may be obtained from IBM Corporation, Data Processing Division, 1133 Westchester Avenue, White Plains, New York 10604.

18. R. P. Parmelee, T. L. Peterson, C. C. Tillman, and D. J. Hatfield, "Virtual storage and virtual machine concepts," *IBM Systems Journal* **11**, No. 2, 99–130.
19. M. McGrath, "Virtual machine computing in an engineering environment," *IBM Systems Journal* **11**, No. 2, 131–149.
20. A. Newell, J. Shaw, and H. Simon, "A variety of intelligent learning in a general problem solver," *Self Organizing Systems, Proceedings of an Interdisciplinary Conference*, Pergamon Press, Elmsford, New York 10523.
21. "OS/VS Data Management Services Guide," IBM Publication GC26-3783, may be obtained from IBM Corporation, Data Processing Division, 1133 Westchester Avenue, White Plains, New York 10604.
22. W. M. Inglis, "Security problems in the WWMCCS GCOS *Joint Technical Support Activity Operating System Technical* system," *Bulletin* 730S-12, Defense Communication Agency (August 1973).

Reprint Order No. G321-5029