NBSIR 76-1041

Security Analysis and Enhancements of Computer Operating Systems

Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, D. C. 20234

April, 1976

Final Report

U.S. DEPARTMENT OF COMMERCE
NATIONAL BUREAU OF STANDARDS

NBSIR 76-1041

SECURITY ANALYSIS AND ENHANCEMENTS OF COMPUTER OPERATING SYSTEMS.

R. P. Abbott
J. S. Chin
J. E. Donnelley
W. L. Konigsford
S. Tokubo
D. A. Webb

The RISOS Project
Lawrence Livermore Laboratory
Livermore, California 94550

T. A. Linden, Editor

Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, D. C. 20234

April 1976

Final Report

U.S. DEPARTMENT OF COMMERCE, Elliot L. Richardson, *Secretary*
James A. Baker, III, *Under Secretary*
Dr. Betsy Ancker-Johnson, *Assistant Secretary for Science and Technology*
NATIONAL BUREAU OF STANDARDS, Ernest Ambler, *Acting Director*

Foreword

This is one of a series of documents prepared as part of a project on computer security and privacy at the Institute for Computer Sciences and Technology of the National Bureau of Standards. This document is intended primarily for use by those who are responsible for managing and operating government data processing installations. It provides an understanding of the types of security problems that arise in current computer operating systems, and it suggests ways in which the security of these operating systems can be enhanced. The document may also be of use to: (1) those engaged in the development of computer security techniques, (2) the manufacturers of computer systems and software, and (3) those responsible for managing and operating computer systems in the private sector.

This document concerns the security problems that arise in computer operating systems. In order to develop a balanced set of security safeguards, one should use it in conjunction with documents that treat other specific aspects of the security problem. Other NBS publications on computer security that may be of particular interest for this use are:

Computer Security Guidelines for Implementing the Privacy Act of 1974, Federal Information Processing Standards Publication 41, U.S. Government Printing Office, Washington, D.C. 20402, Catalog No. C13.52:41, $0.70.

Guidelines for Automatic Data Processing, Physical Security and Risk Management, Federal Information Porcessing Standards Publication 31, U.S. Government Printing Office, Washington, D.C. 20402, Catalog No. C13.52:31, $1.35.

Exploring Privacy and Data Security Costs - A Summary of a Workshop, NBS Technical Note 876, U.S. Government Printing Office, Washington, D.C. 20402, Catalog No. C13.46:876, $0.85.

Proposed Federal Information Processing Data Encryption Standard, the Federal Register, August 1, 1975.

Computer Security Risk Analysis Guidelines, to be published.

This report is applicable to most general purpose computer operating systems; however, it discusses, in detail, the security features of three operating systems. These systems are: IBM's OS/MVT, UNIVAC's 1100 Series Operating System, and Bolt Beranek and Newman's TENEX system for the PDP-10. They were chosen for their illustrative value--two of them because they are the most commonly used large systems in the Federal Government inventory, and the third because a detailed analysis of its security was available, and because many of the specific security flaws found in the system can be used as detailed examples of typical security flaws. Most known TENEX flaws have been corrected in all currently used versions of the system.

Guidance is provided for specific security enhancements; however, the amount of detail contained in this report is constrained by the danger that excessive detail could be harmful. Excessive details about current security flaws might be used by someone intent on penetrating security. On the other hand, those responsible for security must be made aware of the security techniques that are available, and they must understand and prepare for the dangers to which they are still exposed.

The authors of this document have attempted to write it in a way that provides as much information as possible to those responsible for system security while at the same time minimizing its potential usefulness to someone who might misuse the information. It is generally acknowledged that the security provisions of most current operating systems can be broken by an experienced programmer who has spent much time working with the system and has a very detailed understanding of its inner workings. The guidance used in the preparation of this document was that it should not increase the number of people

who know all the details needed to effect a security penetration. Many details about specific security flaws have not been included in this report either because there is no reasonable enhancement to correct the flaw or because exploitation of the flaw could be carried out by someone with relatively little additional detailed information about the system.

The security enhancements suggested in this document do not provide complete protection against all the security flaws in the operating systems. The reader should not anticipate that the correction of the identified security flaws will do any more than reduce the number of avenues by which the system software might be penetrated. Whether the suggested enhancements will result in a significant improvement in a system's overall security posture depends on many factors that are unique to each computer installation; in particular, it depends on the characteristics of the data processing environment, the specific software and hardware configurations, the value or sensitivity of the information being processed, and the nature of the threats to that information that can reasonably be anticipated. It is very difficult to evaluate whether a specific security enhancement is a cost-effective way of improving a system's overall security posture; that decision can only be made by people who know the characteristics of the specific data processing installation and who are also familiar with the current state-of-the-art in computer security. Many data processing installations may have the option of relying mostly on physical, procedural, and administrative security controls so that confidence in the integrity of internal system controls is not needed.

Early drafts of this document - together with lists of specific security flaws - were made available to the different vendors. In most cases vendor action is the most efficient way to correct a security flaw. This document will be especially useful if it reduces the current tendency for the same security flaw to reappear repeatedly in different systems.

Dennis K. Branstad
Theodore A. Linden

Institute for Computer Sciences and Technology
National Bureau of Standards

Contents

Acknowledgment

This report was prepared for the National Bureau of Standards, Order No. S-413558-74, as part of the work of the Research In Secured Operating Systems (RISOS) project at Lawrence Livermore Laboratory. The RISOS project is sponsored by the Advanced Research Projects Agency of the Department of Defense (ARPA) under ARPA Order No. 2166. The work was performed under the auspices of the U.S. Energy Research and Development Administration.

The authors of this document are:

R. P. Abbott
J. S. Chin
J. E. Donnelley
W. L. Konigsford
S. Tokubo
D. A. Webb

SECURITY ANALYSIS AND ENHANCEMENTS OF COMPUTER OPERATING SYSTEMS

The protection of computer resources, data of value, and individual privacy has motivated a concern for security of EDP installations, especially of the operating systems. In this report, three commercial operating systems are analyzed and security enhancements suggested. Because of the similarity of operating systems and their security problems, specific security flaws are formally classified according to a taxonomy developed here. This classification leads to a clearer understanding of security flaws and aids in analyzing new systems. The discussions of security flaws and the security enhancements offer a starting reference for planning a security investigation of an EDP installation's operating system.

Key words: BBN-TENEX; IBM OS/360; UNIVAC 1100 Series OS; operating system security; software security; security flaws; taxonomy of integrity flaws.

1. An Overview

This document has been prepared for use by computer, EDP, and systems managers:

- To aid in understanding the issues of confidentiality, protection, and security as they apply to computer operating systems.
- To provide information that will assist in assessing how much effort is required to enhance the integrity features of their operating systems.

To meet these objectives, two operating systems, which are commercially available, were selected for analysis. The two systems were selected from those commonly used in Federal Government computer centers. A third system has also been analyzed and is presented here because of its more recent design and because the issue of security was considered during its design phase.

The material in this document is divided into three major areas. Sections 1-3 comprise the first area. Section 1 introduces the material with discussions of the motivational and technical aspects of computer security and the relative importance of operating system security. Section 2 deals with general operating system security as it applies to a range of systems. Section 3 presents a taxonomy of integrity flaws, that is, a more formal, systematic way of portraying and classifying these problems.

The second major area contains sections 4, 5, and 6, and each deals with a specific operating system: IBM OS/MVT, UNIVAC 1100 Series Operating System, and Bolt Beranek and Newman's TENEX for the PDP-10, respectively. The last area includes section 7, the summary and conclusions; a glossary; and a bibliography.

1.1 MOTIVATION FOR ENHANCING SECURITY

Initial interest in computer security came from the area of national security. It is fairly easy to recognize the need for protecting the data that relates to a nation's defense. However, privacy and confidentiality became issues as the nation's attention was focused on the increasing amount of personal information contained within computer systems. As the volume of information grew, so did the possibility that information might be used in a manner which was not intended.

In the business community and in the Government, many computerized records afford opportunities for fraud or embezzlement. Some examples of volatile and highly sensitive records are: proprietary data and programs; records of ownership --cash deposits, stock transactions, real property, etc.; and online banking. It is easy to imagine the implication of even a temporary modification of such records. A decision, based on the temporarily modified data, could have far-reaching effects.

Definitions of security (as applied to computers), confidentiality, and privacy are presented in the Glossary. Consider at this point, however, a rather legalistic and simplistic definition of these words:

Integrity is the state that exists when there is complete assurance that under all conditions a system works as intended.

Computer security is the composite protection of administrative and physical security for computer assets and data security.

Data security is protection against accidental or deliberate modification, destruction, or disclosure of data.

Confidentiality relates to data. The word confidential means entrusted with the confidence of another or with his secret affairs or purposes; intended to be held in confidence or kept secret.

Controlled accessibility is the protection provided to information and computational resources by the hardware and software mechanisms of the computer itself.

Privacy relates to the individual. The right of an individual to decide what information about himself he wishes to share with others, to be free from unwarranted publicity, and to withhold himself and his property from public scrutiny if he so chooses.

Public Law 93-579 (The Privacy Act of 1974) is not necessarily a justification for enhancing the security of a computer's operating system; however, it does focus attention on the protection of data. An examination of the Privacy Act is in order so that an appropriate level of effort may be directed toward operating system security as it affects the confidentiality and privacy of data.

The first portion of the "Privacy Act of 1974" reads:

"Sec. 2. (a) The Congress finds that —

(1) the privacy of an individual is directly affected by the collection, maintenance, use, and dissemination of personal information by Federal agencies;

(2) the increasing use of computers and sophisticated information technology, while essential to the efficient operations of the Government, has greatly magnified the harm to individual privacy that can occur from any collection, maintenance, use, or dissemination of personal information;

(3) the opportunities for an individual to secure employment, insurance, and credit, and his right to due process, and other legal protections are endangered by the misuse of certain information systems;

(4) the right to privacy is a personal and fundamental right protected by the Constitution of the United States; and

(5) in order to protect the privacy of individuals identified in information systems maintained by Federal agencies, it is necessary and proper for the Congress to regulate the collection, maintenance, use, and dissemination of information by such agencies."

Another excerpt from the Privacy Act of 1974:

"Sec. 3. (m) Government Contractors. When an agency provides by a contract for the operation by or on behalf of the agency of a system of records to accomplish an agency function, the agency shall, consistent with its authority, cause the requirements of this section to be applied to such system. ...any such contractor...shall be considered to be an employee of an agency."

Personal information about an individual must be protected against misuse. That is, the person's privacy must be safeguarded by maintaining the confidentiality of data related to the individual. If that information has been placed in a computer system, that computer system must maintain the confidentiality of the information. Therefore, that computer system must be secure against the misuse of information on individuals.

The law not only mandates the protection of information but requires agencies to implement security safeguards as stated in Section 3 of the law:

"(e) Agency Requirements.-Each agency that maintains a system of records shall—...

(10) establish appropriate administrative, technical, and physical safeguards to insure the security and confidentiality of records and to protect against any anticipated threats or hazards to their security or integrity which could result in substantial harm, embarrassment, inconvenience, or unfairness to any individual on whom information is maintained; and..."

1.2 TECHNICAL ISSUES IN ENHANCING SECURITY

Data security is the protection of data against accidental or deliberate destruction, modification, or disclosure. If a remote-access, timeshared system crashes and causes confidential information to be displayed randomly on one or more terminals, it may be considered to be an accident. If, however, someone causes the crash for the purpose of gathering such information, then that is a deliberate disclosure of confidential information. Neither case is desirable.

From a software point of view, both the operating system and each application program bear responsibility for maintaining data security. It is, however, the operating system that controls, assigns, allocates, and supervises all resources within the computer system. Core space, input/output (I/O) channels, peripheral units, data files, the master file index, and the CPU are accessible to an application program only after appropriate dialog (i.e., system calls) with the operating system. Should the operating system be tricked, subverted, controlled, or compromised by an application program, the confidentiality of information may be violated. The end result is the same regardless of whether the act of subversion was accidental or deliberate.

The ideal situation is one in which operating system security is a major design criterion. Even then, consideration must be given as to whether the design is correct, the design is correctly interpreted, and the interpretation is correctly implemented. Unfortunately, computer science has not advanced to the point where it is possible to prove that a sizable program has been correctly designed, interpreted, and implemented. It may well be that an incorrect design, an incorrect interpretation of that design, and an incorrect implementation may appear to provide a satisfactory operating system. Other combinations of correct or incorrect designs, interpretations, and implementations may also appear to be satisfactory.

For the most part, the operating systems that are in use today have not been designed with security and controlled accessibility as significant design criteria. In view of the desire to protect an individual's right to privacy, it may be a violation of the right to privacy to wait for an occurrence of system compromise. Therefore, an operating system must be examined for weaknesses, by knowledgeable systems analysts (programmers), with the objective of implementing corrections for any and all observed weaknesses.

1.3 OPERATING SYSTEM SECURITY WITHIN TOTAL EDP PROTECTION

Operating system security is only one aspect of the total integrity-privacy-confidentiality protection picture and needs to be viewed within a comprehensive cost-risk analysis. In some cases, the integrity of the operating system is a minor element; however, in other cases the operating system is critical and can be the weakest link of the complete EDP system.

Overall protection of a computing installation encompasses three protection areas:

- Physical.
- Information.
- Service.

Physical protection is the safeguarding of installation facilities against all physical threats; that is, protection against damage or loss from accident, theft, malicious action, or fire and other environmental hazards. Physical security techniques involve the use of locks, personal ID badges, guards, security clearances, sprinkler systems, etc. Physical protection is a prerequisite for information and service-level protection.

Information (data) protection is the safeguarding of information against accidental or unauthorized destruction, modification, or disclosure. This requires the use of both physical security (including procedural and administrative) and controlled-accessibility techniques. The software mechanisms that control access are operating systems, application programs, and utility or service programs.

Service-level protection is the safeguarding of a computer system's services from degradation or failure (i.e., crashing). A reliability failure or malicious action can cause this service degradation. The nature of the applications at a given installation normally indicate the importance of security measures for this protection.

A blend of different security measures is used to achieve the desired degree of protection:

- Personnel security — credit checks, security training, and reminders.
- Management policies — standard operating procedures which reflect a constant and committed desire to protect computer-contained information.
- Physical security — controlled physical access locks, guards, and fire protection.
- Operating system security — protection of system tables, checking of all arguments, and verification of parameters.

It is important to understand that operating system security is only one aspect of the total security area needed for insuring integrity, privacy, and confidentiality protection. The operating system can be used (or misused) in very sophisticated and subtle ways to effect a security compromise. Also, the detection of security misuse can be prevented or covered up in some instances. (Later sections will discuss several of these security flaws and what action can be taken to prevent or eliminate them.) These techniques demonstrate that an installation's operating system is a critical avenue through which data and service can be compromised. However, in the overall protection of an installation the "weakest link" concept is relevant and must be considered. This document addresses

only operating systems and should be used as a starting reference for planning a security investigation of an installation's operating system. Publications that cover other elements of security are referenced in the Bibliography.

1.4 AN EXAMPLE OF A SECURITY FLAW

Sections 4, 5, and 6 examine general and specific flaws. Before presenting this material, it will be useful to consider an example from an interactive timeshared system. Even though the example is somewhat technical, the conclusions and lessons are important and illustrative of several specific problems discussed later. The example has been chosen because:

- It is specific to a number of systems and may be generalized to apply to most systems, as will be noted.
- It is relatively harmless in the event that a misguided reader should invoke it against an unsuspecting and unprotected system.
- It serves to illustrate the point that computer security is a function of the environment in which the computer operates.

Assume the following sequence of actions on an interactive time-shared or multiprogrammable system:

- A program is started.
- The program activates an I/O activity.
- The program terminates after the I/O request is issued, but before the I/O request is completed.

The preceding sequence will result in the permanent elimination of the program's core space as a further system resource (i.e., a memory lockup). In other words, the system will have to be stopped and then reloaded before the core space that was used by the program may be reassigned to any other user. Although the example is specific to a number of present-day systems, its generalized form has a much broader application: any system that permits two or more asynchronous events to take place is susceptible to resource lockup. Those systems which perform a periodic or conditional collection of all memory space (or resources) not attached to an active process will be immune to this example.

The inner workings of the operating system and how the above sequence results in a memory lockup requires further explanation. An operating system can be viewed as consisting of two sections: an interrupt handler and a housekeeper. In this example, the housekeeper, upon receipt of the ending request, severs all connections between itself and the program. An exception is made for the memory map table because of the outstanding I/O request. When the end of I/O action is processed by the interrupt handler, it removes all traces of the outstanding I/O request, but does not clear the memory map entry. Thus, the memory map is left with a setting which indicates that a portion of core is occupied.

A number of different observations may be drawn from the memory lockup example:

1) Although the actions that must occur are specified, there are any number of ways a program can be written to produce the same end-result. The I/O device activated can be tape or disk. The program can be written in assembly language as well as some higher-level languages.
2) The example does not state which operating system will be affected by the procedure. In fact, it will work on a number of operating systems controlling the hardware of different manufacturers. This suggests that there is a commonality among operating systems with regard to the types of errors to be found in each.

3) Taken together items 1) and 2) suggest that a) there may be a set of generic classes of errors that are applicable across manufacturer product lines and b) each generic error may be expressed in a variety of programming styles. A more thorough treatment of this point may be found in section 3, Taxonomy of Integrity Problems.

4) This particular example is time dependent. The command to terminate the program must occur after I/O is started, but before it is completed. Operating systems are vulnerable to both time-dependent as well as time-independent sequences.

5) What is the impact of this example on security? If the computer system has a real-time component, it is possible that critical real-time programs will not be able to find space to fit in core. Whatever the real-time programs are supposed to do, they may lose their timeliness. If the system has no real-time component, revenue may be lost either as a result of the machine not producing revenue from clients or because the job queue is not exhausted at the end of the time period.

6) Any overt action that forces an abnormal reaction from a computer operator may be a masking action to hide or to bring into being a more devastating set of circumstances. It should be noted that there is ample opportunity in the example to erase all but a few lines of the culprit code. This erasure makes it difficult if not impossible to trace accountability in an audit-trail sense.

A more powerful point can be established as a result of items 5) and 6). The decision as to whether a particular operating system flaw affects security, and ultimately privacy and confidentiality, is a function of the environment in which the computer operates and the mission to which it is assigned. A flaw that has catastrophic consequence at one installation may have no impact at another installation.

2. Security Enhancements of Operating Systems

This section discusses general controls and actions that can be taken within a computer installation to enhance the integrity of operating systems. These security enhancements can serve as either a detection (audit) or a corrective-preventive control. Depending on the nature of the problem and the proposed action, different enhancements may be implemented by users, systems programmers, or installation managers.

The security flaws discussed here are formally classified in the taxonomy in section 3. General and specific examples of integrity problems and their enhancements are described in sections 4, 5, and 6, where specific operating systems are analyzed.

2.1 DETECTION CONTROLS

If data are examined or changed by an unauthorized user, an integrity compromise has occurred. This compromise is magnified when the compromise goes undetected. If this action is not detected and reported, then neither corrective action nor preventive measures will be taken. Thus, an integral part of operating system security is the inclusion of detection controls or audit trails.

Most operating systems have some audit-trail facilities. In a transaction-oriented system, the auditing can be complete enough to allow 100% reproduction of all operations for a given time period. This level of reporting does provide information for detecting misuse of the system. However, recording all system actions does not mean that integrity problems are necessarily reported to the proper people (e.g., a security officer). This is an administrative step that must be taken in addition to the initial recording.

For systems that are not transaction-oriented, the detection control is much more complex. It is quite common to log statistics as to what jobs and users are running and what resources are being used. Normally, this log information is sent to a console typewriter, but an administrative step is still required to report any discrepancies to the proper person. The information can be helpful in detecting resource exploitation such as system degradation or system crashes. However, detection controls are often inadequate to detect information exploitation such as the destruction, reading, or altering of data. This is because the file access information is normally not maintained.

Commercial audit packages are available from most of the large certified public accountant (CPA) firms [1]. These packages, however, only extract and report data from previously created files (quite often files of financial information). The data are checked to see if they are accurate, but there are no facilities to determine what user program read or modified the data. What is required is an audit trail of not only who is on the system, but what data files are available, what files are referenced, how files are referenced, and in general, any exception to a predefined standard of normal processing.

2.2 CORRECTIVE-PREVENTIVE CONTROLS

Numerous corrective-preventive controls can enhance the security of operating systems. These controls can affect the operating system directly, as with code modifications, or indirectly, as with procedural and administrative changes. Basically the controls are measures designed to prevent users from 1) executing in a supervisor state or master mode, 2) reading or altering data or files to which authorized access has not been granted, and 3) degrading system performance by crashing the system or using resources without corresponding accounting charges.

These controls are implemented through hardware, software, user actions, or administrative/physical steps. The ease and cost of these enhancements vary considerably. Hardware and software changes usually require more effort and cost than those taken either by users or by enacting administrative controls. However, the effectiveness of each enhancement must be considered.

a. Hardware

Some hardware controls are originally built into the system; others can be added as required. In both cases, some amount of software coordination is usually required to derive the full protection benefits.

- Built-in controls

With a multi-user environment, it is necessary to protect against unauthorized reading, modification, and execution of sections of memory. Several hardware designs can provide this protection and are usually fundamental to the computer architecture. One, physical memory can be originally divided into sections and a key assigned to each section. The key indicates the type of access allowed, if any. Two, protection can also be provided on a logical section of memory via base and bounds registers. Three, virtual storage can be used that requires the hardware to perform paging and segmentation.

In addition to memory protection, control protection is also normally a designed-in hardware feature that involves the restricted use (execution) of certain instructions. Examples of actions that a system protects against are: modifying program status words, halting execution, and issuing direct I/O commands. This protection is often implemented by the machine having two modes, or states, of operation: system and user. System mode is a privileged state in which any instruction may be executed; user mode is a restricted state in which certain instructions can not be executed.

- Add-on controls

Protection can also be provided by adding a new piece of hardware or modifying existing hardware. However, specific changes are limited by the hardware configuration in question, and most computer installations do not have the resources to effect these changes. Three examples of add-on hardware are as follows:

One, an encryption device can be used for protecting data. For example, the encryption algorithm recently described by the National Bureau of Standards [2] can be implemented in hardware. This device could then be attached to I/O or storage devices. With this configuration, data are transmitted or stored in an encrypted ("unreadable") form. Even if an unauthorized user accessed the data, it could not be decoded without also obtaining and using the key that originally encrypted the data. Thus, the key must be protected to protect the data, but this is a much easier task. Currently, encryption appears to be the most reasonable hardware addition for providing data security. The National Bureau of Standards intends both to submit the encryption algorithm to the Secretary of Commerce for consideration as a uniform Federal ADP Standard and subsequently to publish guidelines for implementing and using the algorithm [2].

Two, a hardware monitor can be attached to the existing hardware to record (or trap) execution actions. This monitoring records and evaluates how a system is being used in terms of efficiency. Also, the monitor can be used to log references to resources such as I/O channels and disk drives.

And three, a device can be added that provides permanent write protection for a physical section of core. The section of core could contain sensitive control or status information needed only by the operating system. Although this feature has significant security implications, it is not available for all systems.

b. Software

Software controls are the most common and direct security enhancements for operating systems. However, they are often costly as they require installation implementation and can introduce new integrity problems. Some software controls are as follows: 1) removing a function or module from the operating system, 2) adding a function or module, or 3) modifying existing code.

- Removing software functions

The removal of routines or parts of routines can directly increase the level of security of the operating system. The functions of these routines may be useful but permit unintended results. Also, some routines may have been originally included to circumvent protection features. Two examples of removing software are as follows:

The removal of a checkpoint/restart routine can enhance protection. This routine takes periodic dumps of program status data and intermediate results so that in case of a system crash, the program can be reinitiated at the last checkpoint as opposed to a complete restart. Security can be compromised if critical status data in the checkpoint dump are altered and used in a reinitiation of the program.

Removing system programmer traps can also enhance protection. When a system is implemented, traps or "hooks" are often included to allow special operating privileges to system programmers. The traps are intended for debugging or legitimate system maintenance. However, their usefulness depends on the secrecy of their existence, and secrecy is a very poor security protection method. Thus, the use of traps should be strictly limited or they should be removed.

- Adding software functions

Adding software functions to an operating system can be done either by the vendor or by the installation itself. As security is becoming more important, vendors are making available some routines to afford protection. Two examples of adding software functions are as follows:

The use of passwords can protect data files and user accounts. This function deals with the problems of authorization and authentication. The quality of the password software mechanism and the manner in which the passwords themselves are administered are critical and demonstrate the multi-dimensional nature of security enhancements. Short passwords (e.g., only four characters), passwords chosen for ease in remembering (e.g., name of user's spouse), or lack of exception action (e.g., not reporting several incorrect password tries) can lead to compromise and a false sense of security.

A monitor, or control routine, could be used as an audit tool to record resource usage or data accesses. In addition to recording this information, a check can be made against a predetermined authorization list to see if the action is valid and to prevent its completion if not.

- Modifying software functions

Modifying existing operating system code is a nontrivial task. Systems are normally very large and the interaction among modules is complex so that a change may produce an undesired and unexpected "ripple" action. However, code changes can significantly enhance the security of a system.

The following are two examples of system problems that can be corrected by modifying software:

1) Coding in which parameters are not adequately checked. Some system routines do not validate input parameters because of the assumption of either a benign environment or that another system routine made the validation. This can lead to routines being used for unintended purposes and security compromises.
2) Coding in which system routines store data in user storage area or execute in master (or privileged) mode when not required. These practices are not direct security flaws, but they allow users to modify data being used by the system and gain special privileges — either of which can then be used to compromise integrity.

c. User Action

Individual users can take some direct action. The most obvious is to use existing security controls, such as passwords and proper tape-labeling techniques. Also, system routines should be used in the intended manner without using tricks that may have unintended consequences.

The user must be aware of possible integrity problems and take direct action to counter them. For example, in some installations user-to-user scavenging may be a security problem. That is, code and data are left after a program terminates and a subsequent user, resident in the same core area, can read the unaltered information. In this case, a user could scrub or zero-out all buffer and data areas before terminating the program.

Another instance of possible user security action deals with terminal sign-on procedures. It is the user's responsibility to determine that he is interacting with the system and not with another user's program imitating the system. Entering passwords or accounting information on a terminal without first verifying that one is communicating with the operating system can compromise the entered information. Another user could be imitating the operating system and recording the entered information (e.g., passwords) for later unauthorized use. To prevent compromises of this type, users must interact with the system in a way that can not be duplicated by a user's program (e.g., using a terminal control key to sign off prior to initiating the sign-on procedure).

Also, users should always sign off properly when finished processing. This may involve destroying all programs and work files when through. This avoids the problem of leaving data files or programs on the system and available to anyone who happens to subsequently use the terminal.

d. Administrative-Physical

The installation manager or person designated with security responsibilities can take direct action to enhance operating system security. This action normally is to prohibit or mandate certain user actions by policy decisions or by physical actions (often some hardware or software action must accompany the administrative decision).

From a practical point of view, administrative or physical security enhancements are very important. Usually they are the first enhancements made. They can be implemented in a relatively easy and cost-effective manner and provide a significant amount of security. These measures will not prevent the very determined individual from compromising security, but it does increase the difficulty of compromising and the risk of detection. An added benefit can be a more disciplined and orderly installation.

Items that fall into this class include restricting terminal access, requiring all tapes to be labeled (with the corresponding software checks), standardizing log-on procedures, requiring passwords, using system-generated passwords, using encryption devices for data transmission, limiting actions an operator may perform in response to console log messages, and using guards and some form of badge identification around the computer facilities.

A final administrative enhancement concerns a procedure for recording all changes made to the operating system. A formal procedure should be set up to document and account for each change implemented. This is an audit-type control that fixes accountability, restricts the number of modifications, and ensures that someone understands the modification. The approval ("sign off") for each step in modifying an operating system (requesting, implementing, and verifying correctness of changes) should be done by different people.

3. Taxonomy of Integrity Flaws

3.1 INTRODUCTION

In this section, a system of arranging integrity flaws into related groups is presented, and one class of integrity flaw — operating system security flaws — is examined in detail (Sec. 3.8).

3.2 TAXONOMY OF INTEGRITY FLAWS

Table 3-1 presents a taxonomy (i.e., a system of arrangement) of integrity flaws. Table 3-1 is divided into two segments and an example. Segment one, the syntax portion, clarifies that the mere existence of a flaw renders an installation vulnerable. This is analogous to the engineering concept of "unavailable" potential energy. When an individual (or group) becomes aware of a flaw, an active potential to violate installation integrity is achieved — analogous to "available" potential energy. With adequate motivation, skill, resources, and opportunity, this potential is transformed into kinetic energy, and an installation's integrity is penetrated. This penetration of integrity provides the individual with potential access to one or more classes of resources — items of value to an installation or its users. If the individual now chooses, this access may be exploited to produce a loss for the installation (such as a loss of information, service, or equipment) and/or a gain for the individual.

Table 3-1. Taxonomy of integrity flaws

Syntax

A [*Class of User*] user acquires the potential to compromise the integrity of an installation via a [*Class of Integrity Flaw*] integrity flaw which, when used, will result in unauthorized access to a [*Class of Resource*] resource, which the user exploits through the method of [*Category of Method*] to [*Category of Exploitation*].

Syntax Elements

[*Class of User*]

- Applications
- Service
- Intruder

[*Class of Integrity Flaw*]

- Physical Protection
- Personnel
- Procedural
- Hardware
- Applications Software
- Operating System

[*Class of Resource*]

- Information
- Service
- Equipment

[*Category of Method*]

- Interception
- Scavenging
- Pre-emption
- Possession

[*Category of Exploitation*]

- Denial of Possession/Use
  - Steal equipment
  - Destroy equipment
  - Degrade service
  - Interrupt service
  - Destroy data
- Denial of Exclusive Possession/Use
  - Read/Transcribe data
  - Steal service
- Modification
  - Alter data
  - Alter equipment

Example

An "applications" user acquires the potential to compromise the integrity of an installation via an "operating system" integrity flaw which, when used, will result in unauthorized access to an "information" resource, which the user exploits through the method of "scavenging" to "read/transcribe data."

Each classification depicted in the syntax can be divided into subclassifications and each of these subclassifications can be further divided into subclassifications and so on — in descending order from most inclusive to most specific. Segment two depicts the first levels of classification for each of the syntax elements. In the following paragraphs, each classification will be briefly discussed. However, because this document is principally concerned with operating system security flaws, only that class of flaw will be fully expanded and discussed (Sec. 3.8).

3.3 CLASS OF USER

A user may have various capabilities at various times, and similar users may be granted differing sets of capabilities. However, it is useful to classify users in terms of broad sets of capabilities.

a. Applications Users

Under this approach, applications users are those users who have not been specifically granted special capabilities beyond permission to use the system. They are subdivided into consumers and producers. Consumers are the authorized recipients of information products from a computer-based application. Producers are the analysts and applications programmers who design and implement specific

applications which produce information products for consumers. (Producers may or may not be part of the consumers' organization. Producers require access to the computer system to develop products; their programs require access to data in the system.)

b. Service Users

Service users are subdivided into systems and administrative servicers. Systems servicers are members of a computer servicing staff that includes the operators, systems programmers, and maintenance engineers who are responsible for the maintenance and availability of computer system resources. Because systems servicers have physical access to the computer, the operating system code, or the data storage volumes, they have the capability to access any information in or on a system. For example, an operator can replace the installation's protected operating system with a non-protective one or may use computer console switches to alter main storage contents. The hardware vendor's maintenance engineer, in another example, is equipped with a set of diagnostic aids which can be utilized as integrity penetration tools.

Administrative servicers are members of the systems staff who do not have physical access to the computer room or operating system, but who have special software privileges, which, for example, permit access to privileged hardware instructions and special operating system services, or permit special operations on data. Such users frequently have the capability to access any information in a system.

c. Intruder

An intruder is an unauthorized user, he is an outsider. This term applies to individuals or organizations who have no authorized access to a computer installation or its products and who have a possible malicious interest in obtaining unauthorized access.

3.4 CLASS OF INTEGRITY FLAW

The classes of integrity flaws have been mentioned in sections 1 and 2. Briefly, physical protection flaws include: telecommunications interception, mixed-security-level access to terminals, unauthorized access to a computer room, and exposure to natural disasters. Flaws involving personnel security include acts such as sabotage, collusion, and user error. Procedural flaws are, of course, installation-dependent. Examples of such flaws involve tricking (or "spoofing") a system operator into making unauthorized data available to a user; inadequate tape-labeling procedures at an installation; and "Trojan Horse" subversion of an operating system. As used here, "Trojan Horse" refers to covertly implanting computer instructions in a trusted (system) program so that the trusted program executes its intended functions correctly, but with illegitimate side effects. Hardware integrity flaws include problems such as a flaw in which a user's terminal disconnect signal is not passed on to the operating systems software, or a flaw in which all users are permitted access to an instruction such as "disk diagnose," which should have restricted access. Flaws involving applications software include problems of inadequate user-user isolation, insufficient control over access to data, and exploitable flaws in program logic. Almost all applications software flaws have direct analogies with operating system flaws. Operating systems flaws are discussed in detail in section 3.8.

3.5 CLASS OF RESOURCE

The resources of value to an installation or its users are information, service, and equipment. Information includes all the system's files (programs, data, and file directories) and all user files.

Service represents the unimpaired operation of the installation. Service resources include all the capabilities of the operating system. If an applications user obtains access to the hardware wait/idle mode, monitor/master mode, or unauthorized disk-storage space, then a valuable resource has been compromised. Equipment resources include all installation equipment relevant to the unimpaired operation of its computers.

3.6 CATEGORY OF METHOD

Interception is the interruption of communication or connection. For example, a user program masquerading as the system could intercept an unwary user's sign-on password. Scavenging is the searching for something of value from discarded information or supplies. For example, if reading of (scratch) tapes is not prevented, a user could search through the data left by a previous user in an attempt to find some valuable information. Pre-emption involves taking something to the exclusion of others such as a user pre-empting CPU cycles. Possession is taking control of property such as stealing a magnetic tape containing valuable information.

3.7 CATEGORY OF EXPLOITATION

Because the categories of exploitation (Table 3-1) are self-explanatory, they are only listed here for ease of referral and completeness.

- Denial of Possession/Use
  - -Steal equipment
  - -Destroy equipment
  - -Degrade service
  - -Interrupt service
  - -Destroy data

- Denial of Exclusive Possession/Use
  - -Read/Transcribe data
  - -Steal service

- Modification
  - -Alter data
  - -Alter equipment

3.8 DETAILED DESCRIPTION OF OPERATING SYSTEM SECURITY FLAWS

Operating system integrity is concerned with the assurance that the operating system works as intended. Thus, an operating system integrity flaw is any condition that would permit a user (or his programs) to cause the operating system to cease reliable and secure operation. Integrity is thus concerned with reliability (fraud and error) problems and with security (resource and privacy protection) problems.

In this section, the seven major categories of operating system security flaws are discussed and examples of each are given. The seven categories of operating system security flaws are:

- Incomplete parameter validation.
- Inconsistent parameter validation.
- Implicit sharing of privileged/confidental data.
- Asynchronous-validation/Inadequate-serialization.

- Inadequate identification/authentication/authorization.
- Violable prohibition/limit.
- Exploitable logic error.

Associated with the general text description for each of these operating system security flaws is a table in which that flaw is further divided into sub-categories along with a brief, descriptive example for each sub-category. To conserve space, not all of these sub-categories and examples are discussed in the text. A complete description can be found in A Taxonomy of Integrity Problems [3].

a. Incomplete Parameter Validation

At a high level of abstraction, whenever a process (or program) with one set of privileges requests service from a second process with another set of privileges, the preservation of system integrity requires that the request be thoroughly validated. For most operating systems, the boundary of greatest relevance to system integrity is that boundary between a control program, with complete hardware and software capabilities, and user programs, with a limited subset of capabilities. This separation is usually enabled by hardware facilities (such as control/monitor state and storage protection) but is enforced through software.

In general, user programs invoke control program services in a manner similar to subroutine calls, using many parameters. Only the control program has the capabilities to perform the requested services. The purpose of creating this separation or isolation between user programs and the control program is to prevent any user from compromising the functioning of the control program that is performing services for all users (e.g., I/O operations, program initiation, date and time, etc.). If the checking mechanism for each of the requested parameters is not rigorous or complete, it is possible to "fool" the control program into executing the request in a manner which is detrimental to secure operations. To be validated rigorously, parameters must be checked for permissible:

- Presence or absence.
- Data types and formats.
- Number and order.
- Value ranges.
- Access rights to associated storage locations.
- Consistency among parameters (e.g., storage locations).

As an example, three dangerous results can occur if a user succeeds in getting the control program to accept a parameter consisting of an address outside the memory space allocated to that user:

- The control program may obtain unauthorized data for that user.
- A set of conditions can be generated to cause a system crash.
- Control may be returned in control/monitor state to the user.

A penetration attempt illustrating the return of control in control/monitor state to a user program is described below and in figures 3-1 and 3-2.

1) An instruction which, when executed, will transfer control to a predetermined point in the user's program is loaded into a register.
2) A system call is then made which causes the registers to be saved by the control program in Register Save Area (Fig. 3-1).
3) Upon return of control to the user, another system call is made. Among the parameters for this system call, is a pointer (address) that has to point to a location in the control program. This address will be used in transferring control to the appropriate control program service routine. Naturally, the address supplied is the location in the Register Save Area where a transfer back to the user's program had been planted by the previous system call (Fig. 3-2).

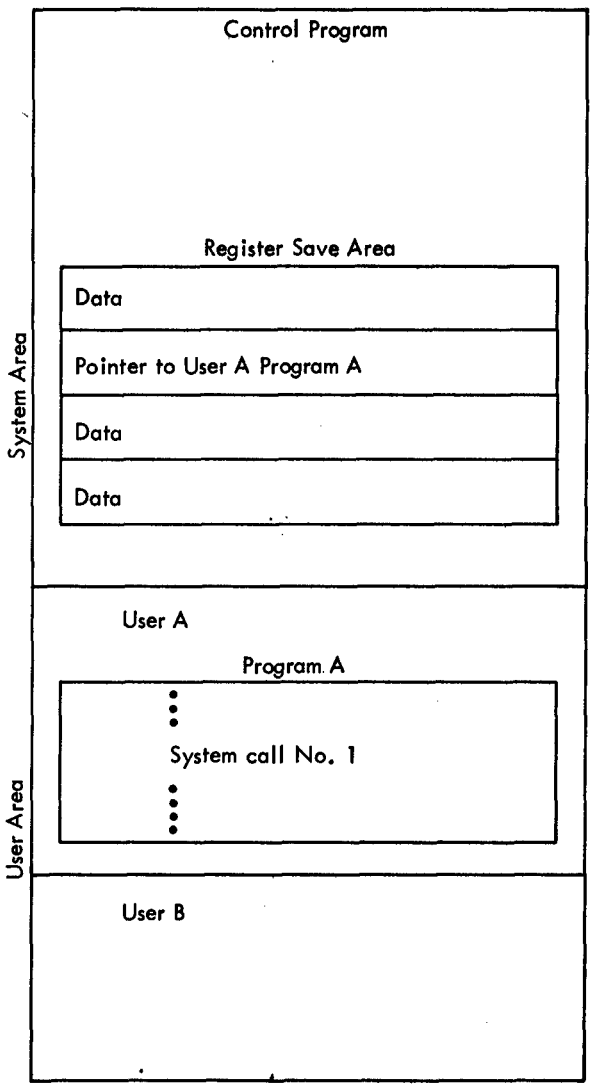

Figure 3-1. Layout of memory after first system call.

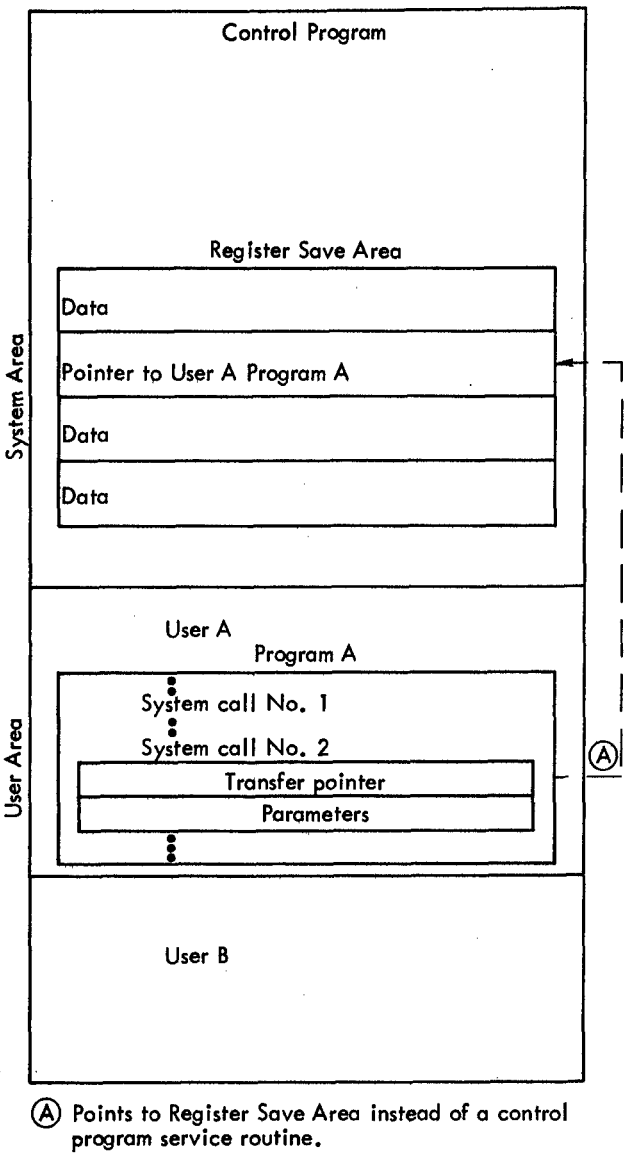

Figure 3-2. Layout of memory when preparing to issue second system call.

4) All parameters are checked and approved; and during execution of the second system call, control is returned in control/monitor state to the user giving the user control of the system.

Table 3-2 further describes the categories of incomplete parameter validation.

Table 3-2. Incomplete parameter validation: categories and examples

1. System routine does not adequately validate parameter attributes.

   Example:

   - The control program does verify an initial I/O transfer. However, it does not verify that the initial I/O transfer will not cause illegal modification to subsequent I/O transfers.

2. System routine does not properly reiterate parameter validation.

   Example:

   - Only the first I/O command or all but the last I/O command in a chained list of I/O commands is verified.

3. System routine validates a parameter under some conditions but not under all conditions of invocation.

   Example:

   - A "confused-deputy" control-program service routine adequately verifies parameters when directly invoked by a user, but not when a user's parameters are indirectly passed to the first service routine by a second service routine.

b. Inconsistent Parameter Validation

Whenever there are multiple definitions of the same construct within an operating system, there exists the possibility that inconsistencies among these definitions will create a security flaw.

This design error goes beyond the incomplete parameter validation error. A situation may exist in which each of several control program routines checks completely for conditions it considers valid; however, the multiple sets of validity criteria (i.e., conventions) are not completely consistent. An example of this category of flaw follows:

Operating systems maintain directories (e.g., catalogs) of the data files used by the system and its users. The contents of these directories are often accessed by as many as half a dozen interface programs. Each of these interface programs makes assumptions as to what constitutes a valid condition in the file system.

Consider something as basic as the characters in the parameters representing the name(s) of users to be given permission to access a file. The routine that creates a master-file-index entry may accept a character (such as an embedded blank) as valid in a specific permission name; whereas all of the other interface programs that modify/delete master-file-index entries assume blanks will never be valid and thus do not accept them. Under such conditions, specific file permissions could be created (such as shared access to a file) which could not thereafter be deleted.

Table 3-3 summarizes inconsistent parameter validation.

Table 3-3. Inconsistent parameter validation: categories and examples

Two or more systems routines perform adequate parameter verification for their purpose, but the multiple sets of validity criteria are mutually inconsistent. Example: • The routine that creates a master-file-index entry permits embedded blanks, but all of the other routines which modify/delete master-file-index entries treat an embedded blank as an error. Thus, once granted, a user may be unable to revoke shared access to a file.

c. Implicit Sharing of Privileged/Confidential Data

To ensure integrity, an operating system must be able to isolate each user from all others and from the control program. This isolation involves both control flow and information. Whenever information isolation is not complete, the system may allow information of greater privilege to become accessible to a lesser privileged user or may allow one user to access another user's information against that user's wishes.

In many operating systems the control/program portion of the operating system shares memory space with user programs, either as work space or as a convenient place to put information associated with that user program. This is a deliberate design policy to facilitate charging individual users directly for resources that they use. If the user requires file operations or other kinds of system resources, the system maintains the information and the work space for his requirement in an area that will be uniquely chargeable to that user. Because the workspace is shared, but in a mode not normally available to the user, operating system implementors have often been careless with regard to the state in which the workspace is left after receiving a user request.

For example, the control program may use such a workspace to read in the master index of user files along with their associated passwords as part of a search for data requested by a given user. This function is necessary in order for the system to determine that the request is properly formed and authorized to the user making the request. If the control program should find that the request is improper, it returns control to the user program originating the request, with an indication of the nature of the error in the request. However, in this example, the control program does nothing about the information remaining in the shared workspace. As a consequence, the user can now access the workspace and obtain from it other user identifiers and authenticators (passwords) which he can then use to masquerade to the system (Fig. 3-3). As shown below, even if the system erases the information before returning control to the user's program, the information can be obtained by the user through some form of concurrent processing, such as an independent I/O operation which reads from the workspace in question. There are other variations of this flaw. Sometimes work files and workspace are not erased when a user releases them, and another user can scavenge this "unerased blackboard" when the uncleared file space or buffer space is next assigned.

Sometimes the full implications of information made available to a user are not realized by the system's designers. For example, control programs frequently acknowledge the disposition of user service requests by setting a return-code/status-flag. Various return conditions (such as: "illegal parameter", "segment error", "password OK", etc.) and other forms of interprocess communication (e.g., SEND/RECEIVE acknowledgment) may connote intelligence that enables a user to breech security.

Table 3-4 summarizes and gives examples of the categories of implied sharing.

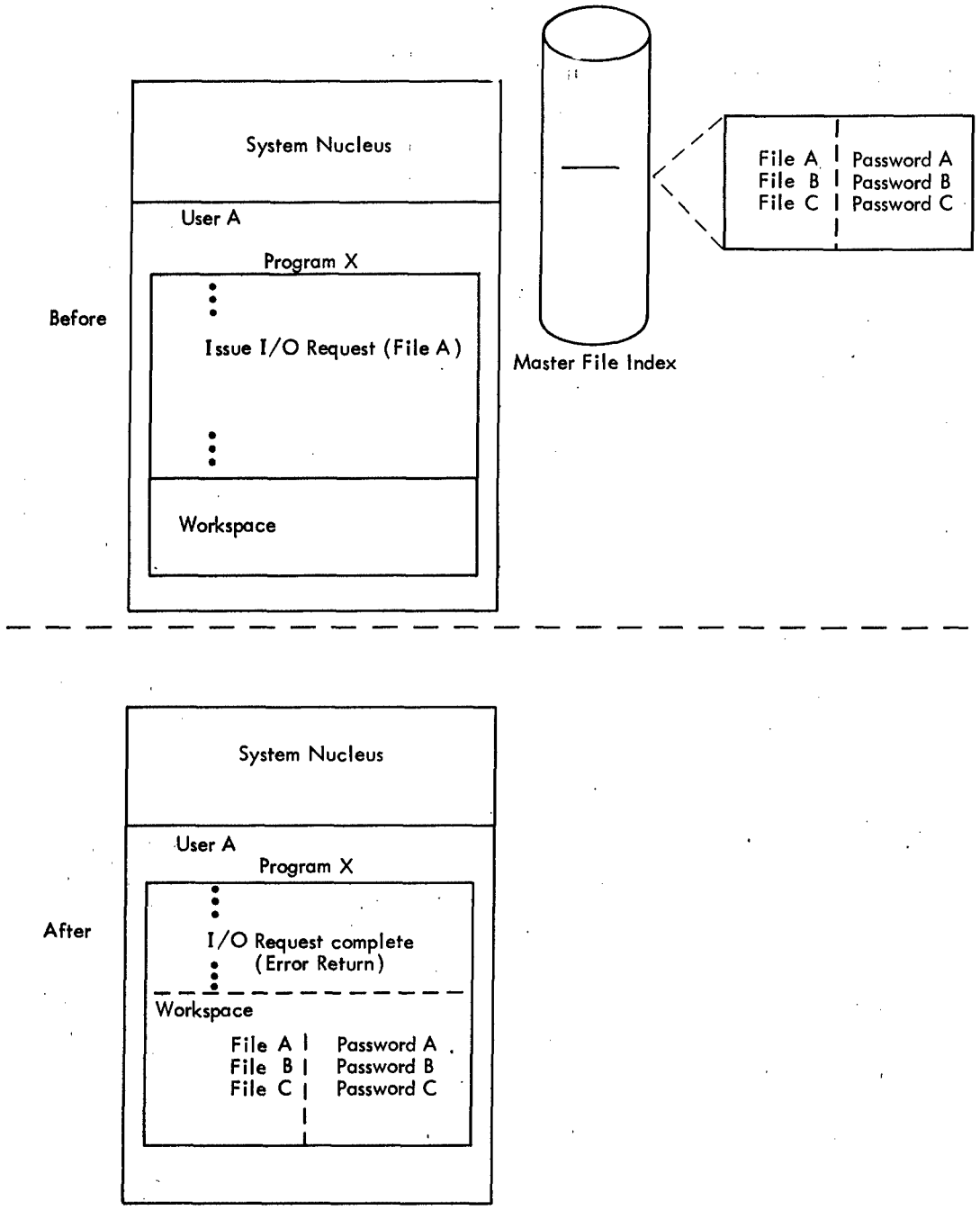

Figure 3-3. Layout of memory before and after issuing requests to read master file index.

Table 3-4. Implicit sharing of privileged/confidential data: categories and examples

1. Explicit transfer of information.

   Examples:

   - While servicing a user request, the control program uses a user-accessible buffer to scan master-file-index entries. While this activity is in process, the user asynchronously reads this buffer and obtains another user's file-index password.

   - The control program does not erase blocks of storage or temporary file space when they are reassigned to another user ("unerased blackboard").

   - A user's password is still legible through the overstrike characters on the user's terminal printout, or a user's password is listed on his batch output when his job command is flushed due to incorrect syntax.

2. Implicit transfer of information.

   Example:

   - A user piecewise decomposes a password by locating it on a page boundary and noting page faults or by precisely timing variations in the execution time required by a password checking routine.

d. Asynchronous Validation/Inadequate Serialization

System integrity requires the preservation of the integrity of information passed between cooperating processes or control program instruction sequences. If serialization is not enforced during the timing window between the storage of a data value and its reference (or between two sequential references), then the consistency of such a data value may be destroyed by an asynchronous process.

Control information is especially susceptible to modification whenever it is located in storage accessible to a subordinate process. This is sometimes called the "time-of-check to time-of-use" problem. As described under the implied sharing of privileged data flaw, an operating system may frequently share memory space with user programs. This space may not only be used for the passive storing of information, but also may contain system or user parameters that represent data upon which future actions will be based. Whenever there is a "timing window" between the time the control program verifies a parameter and the time it retrieves the parameter from shared storage for use, a potential security flaw is created. This is because contemporary operating systems allow a user to have two or more activities (processes) executing concurrently and sharing that user's memory allocation. For example, a user may initiate an I/O operation and then continue executing his program while the I/O operation completes.

In another example, a timesharing user may temporarily suspend one operation by pressing the "attention" or negative acknowledgment (NAK) key on his terminal, perform a second operation, and then return control to the first operation for completion. Some systems permit "multitasking," in which

two or more programs are sharing a single user's assigned memory (address space) and are executing concurrently – perhaps each being simultaneously executed by separate CPU's of a multiprocessing computer system.

The following steps describe an asynchronous validation flaw, which is depicted in figure 3-4.

- In time frame 1, a user issues an I/O request to the control program. The control program validates all of the I/O parameters (including the address pointer to a valid buffer within the memory legitimately assigned to the user), enqueues the I/O request (which must wait until the appropriate device is no longer busy), and then returns control to the user.
- In time frame 2, the user replaces the valid address pointer to his buffer with an address that points to a location within the control program.
- When the I/O is performing in time frame 3, the data requested by the user is read into (or out of) the control program instead of his valid buffer. Instructions within the control program can thus be overlayed with instructions supplied by the user, or privileged control program information can be read out to the user's file.

In some systems, the control program may use an overflow register save area, located in user accessible storage, whenever the control program's primary save area is filled. This saved information generally contains program status and control information.

This situation can give rise to another variation of the asynchronous validation flaw, should a user be able to modify such control information. An example of such a penetration attempt follows:

- A user constructs an I/O record that simply contains an address pointing to a desired location in one of the user's programs.
- Multiple copies of this record are then output as a file.
- The user next initiates an I/O operation to read these records repeatedly into that area of the user's memory utilized by the control program as overflow storage for registers.
- Then the user issues a system service request that causes the control program to make a number of nested intra-monitor calls, thus overflowing its primary save area. (The repeated issuing of certain service requests may also accomplish this aim.)
- The registers saved by the control program in the overflow save area will be overlayed by the input records that contain the address pointing to the user's code. (Some timing adjustments may be required for the user to accomplish this.)
- When the control program eventually restores registers and status from the overflow area, it will transfer control to the user's program in monitor/control state – thus giving the user full control over the operating system.

An operating system may store information over a period of time in shared auxiliary storage as well as in main memory. For instance, an operating system may have a checkpoint/restart provision to record the state of a running program at convenient restart points as "checkpoint" dumps. These checkpoint dumps contain both user data and control information which specifies the control status to be assigned if the program is restarted. The checkpoint dumps are recorded in a file specified to the system by the user and are accessible by that user for manipulation. Through such manipulation, the user could cause his program to be restarted with modified state information that gives his program greater privileges than that originally specified. This can, for example, result in the user gaining control/monitor state privileges.

Table 3-5 further describes the categories of asynchronous validation and serialization flaws, with examples.

Table 3-5. Asynchronous validation/inadequate serialization: categories and examples:

1. Asynchronous modification of user (inferior process) storage.

Examples:

- A user performs asynchronous I/O into his parameter list to modify illegally a previously validated parameter.
- A user performs asynchronous I/O into a checkpoint/restart file so that his process is given additional unauthorized privileges when restarted.

2. Inadequate serialization/control of protected storage.

Examples:

- A user issues a system call which, in part, stores values in an I/O control table and then returns control to the user. The user then issues a second, different system call which also, in part, stores values in the I/O control table — thus overlaying a portion of the previously set values in such a way as to gain unauthorized I/O privileges for the I/O performed in conjunction with the first system call.
- A system routine is induced to overlay its own parameter/storage area by a user. The user supplies an address where a return code is to be stored by the system routine upon return of control to the user. This user-supplied address overlays the initial word of a buffer where the system routine has stored a return jump instruction.

e. Inadequate Identification/Authorization/Authentication

Identification, authorization and authentication are the essential components of the concept of controlled access. Authorization — the controlled granting of access rights — is ultimately based upon authenticated, unique identification of individuals and resources. An operating system is essentially a resource manager. Thus, an operating system is subject to integrity problems whenever 1) it does not require authorization for an individual or process to access any data or to use any resource that should not be available to all, or 2) it does not uniquely identify the resources with which it is dealing.

A flaw is created whenever a system permits a user possessing one set of privileges/capabilities to legitimately bypass (controlled access) security mechanisms and perform an action only permitted to users with differing privileges/capabilities or whenever it permits all users to perform an action that should be restricted only to users of greater privilege.

An inadequate identification/isolation flaw can be created whenever one system routine relies upon mechanisms (implemented elsewhere in the system) to ensure the isolation of system resources and, hence, the adequacy of their identification. This may be a bad policy if the mechanisms are not, in fact, adequate.

For example, to be identified uniquely a program must be identified both by program name and by the name of the library from which it was loaded. Otherwise, it is very easy for a user to preload a

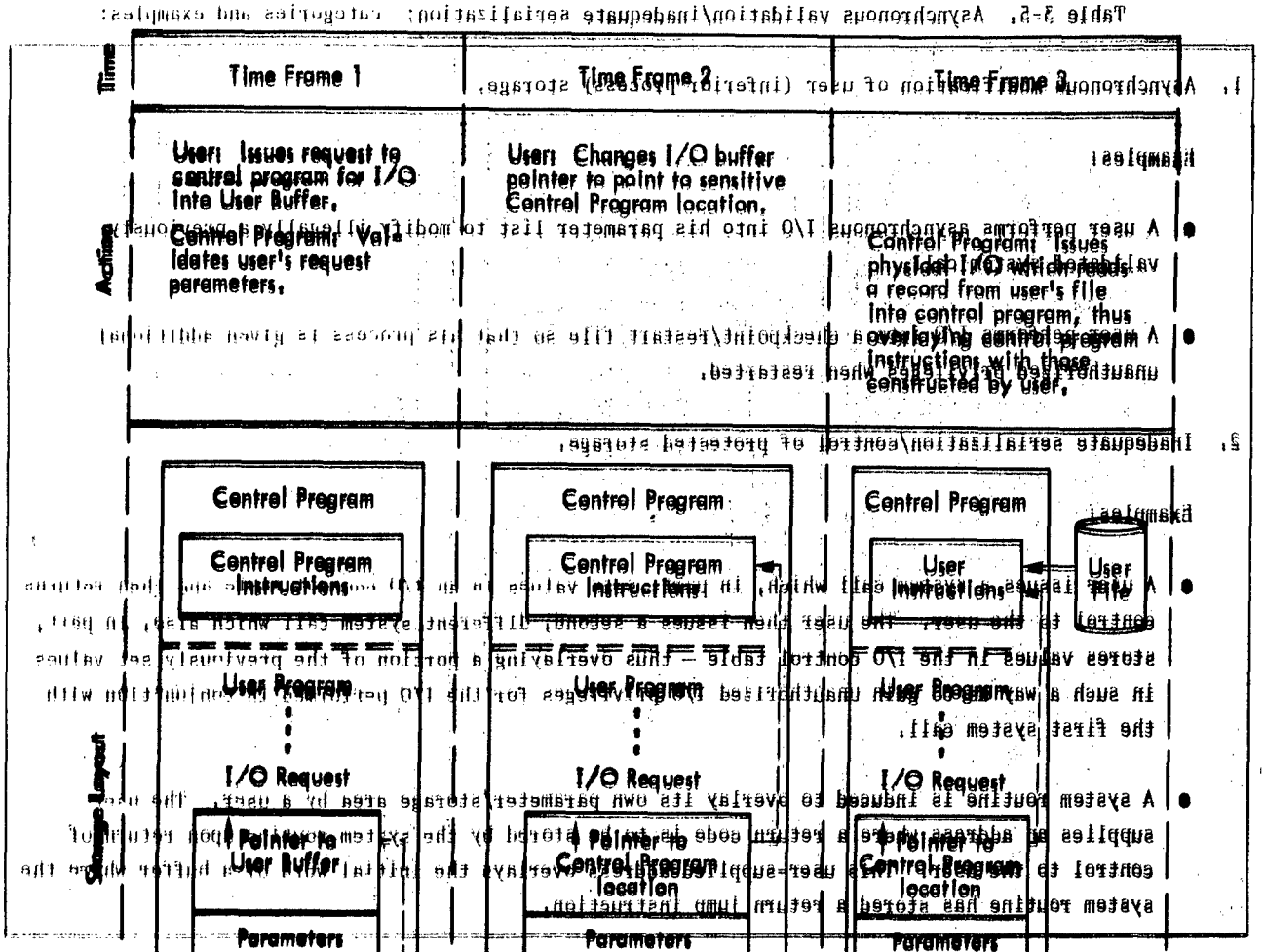

Figure 3-4. An example of an asynchronous validation flaw.

counterfeit program whose name is the same as some control program routine (which must be dynamically loaded when required) and to have this counterfeit routine used by the control program in place of the authentic routine.

To accomplish this, the user generates an activity that will result in the control program requesting this routine. The loader will see that the named (counterfeit) routine is already loaded (which is legitimate) and will set up the control program to use the counterfeit program.

As another example, the user-ID or password-checking mechanism may be circumvented if it does not effectively limit the number of times a user can attempt to log into the system or if it does not limit the elapsed time permitted for completing a login. It may be possible, under such circumstances, for a user to utilize another computer to exhaustively enumerate all password bit combinations and thus break password security.

Some systems have extensive authorization checking associated with most, but not all, of the file-access methods and do not restrict use of those access methods which do not perform authorization checking. Any user who obtains documentation for these latter access methods (unrequired capabilities) has simply to use them to access any file in the system. This is an example of the bypass of controlled-access mechanisms.

Table 3-6 summarizes the categories and presents additional examples of the inadequate identification, authorization, and authentication flaw.

f. Violable Prohibition/Limit

An operating system is described both by its embodiment in computer instructions and by its external documentation. Whenever these two descriptions differ, an integrity flaw may exist. A security flaw is created whenever a documented operating system limit or procedural prohibition is not enforced.

For example, those who implement an operating system may not treat the situation in which the upper limit in size of tables or buffers is reached or when queue space becomes saturated. Documentation may specify precisely these upper limits and prohibit exceeding the limits, but should a user deliberately or accidentally cause an overflow or overload, then various results may occur — sometimes a system crash may result, sometimes system operation is degraded, sometimes sensitive data can be lost, and in some instances such data could be compromised.

Table 3-7 gives examples of this flaw.

g. Exploitable Logic Error

In any major operating system, there are — at any point in time — some "bugs" or logic errors. Many of these errors depend upon statistically improbable timing situations and are not under the control of any individual user. Some of these logic errors can, however, be intentionally exploited by a user to compromise the integrity of a system.

One example involves incorrect error handling. The system may, for instance, perform an illegal action before signaling an error condition. Consider, for example, that a user requests a series of modifications be made to the file directory entry of another user, to which the first user has read-only authorization. If the system performs the requested actions and then determines that the actions exceed the requesting user's authorization, the security of the system has been compromised through a logic error. This also may happen if a system service such as a storage dump is initiated concurrently with the checking of a user's authorization to request the specified service for the specified storage areas. By the time the error is detected, forbidden areas may have already been listed.

Table 3-6. Inadequate identification/authorization/authentication: categories and examples

1. Inadequate resource identification/isolation.

   Examples:

   - A user program with the same name as a system program is preloaded by a user and is then accepted and used by the system.

   - A system routine assumes the validity of a system control table whenever the control table is located in system storage to which users do not have direct write access. In fact, it is possible for a user to create a counterfeit control table and have it copied into system storage by certain control program service routines, such as the storage deallocation routine.

2. Bypass of controlled-access security.

   Examples:

   - A user legally bypasses the file initialization (open) routine and its security mechanisms by utilizing a basic file access method.

   - A user obtains system privileges by taking a legal exit from abnormal job termination (i.e., abort) processing.

   - A user obtains system privileges by discovering and using a "trap door" exit to the system meant for system maintenance programmer use.

   - An operating system, which does not prevent a user from simulating its logout and login functions, permits an unattended (hardwired) terminal to simulate a logged out terminal and obtain another user's password during a simulated login process.

On a more subtle level, a user may discover that half-word arithmetic instructions are used to improperly process a half-word return-address parameter. If the largest possible half-word number is used as an address, an unanticipated overflow may occur, resulting in an address pointing to location 0001 in control program memory, which may cause a system crash.

In another situation, by pressing the "attention" (or NAK) interrupt button on his terminal during the printing of the login error message, a user may be able to cause the system to erroneously accept a new login attempt without advancing the counter set to record the number of previous login attempts. This system error permits automation of an exhaustive enumeration of passwords, with no indication to system operators that this is taking place.

In a last example of incorrect error handling, it sometimes occurs that protection mechanisms are disabled or modified as a result of a (deliberate) user error and may not be reset when the control program later returns control to the user. This can result in the user obtaining unauthorized privileges.

Table 3-7. Violable prohibition/limit: categories and examples

1. Violable system limit.

   Examples:

   - A user is supposed to be constrained to operate only within an assigned partition of main storage, while in fact, the user may access data beyond this partition.

   - A user is supposed to be constrained in the amount of system queue space available to his process, when in fact, the user may create an uninterruptible, endless loop on a system call that eventually uses up all of the control program's queue space. This causes the system to crash.

2. Violable system procedural prohibitions.

   Example:

   - A user is able to obtain unauthorized privileges by omitting notification to the operating system of an exit from an I/O error-processing routine, although the documentation requires such notification.

Table 3-8 lists three categories of logic error flaws and presents some additional examples. There are two additional categories of exploitable logic flaws. These are listed here for completeness and without examples:

- Incorrect process/function initiation or termination.
- Control-state software error trap.

Table 3-8. Exploitable logic error: categories and examples

1. Incorrect error-handling sequencing.

   Example:

   - The operating systems fails to update a count of unsuccessful login attempts if a user presses the interrupt key (NAK) on his terminal just after submitting a password guess.

2. Instruction side-effects.

   Examples:

   - The operating system uses full-word arithmetic on a half-word return address supplied by the user. If the value supplied is -1, this causes an overflow into the index field of the word, and a return to the user in control state.

   - An operating system uses a particular indirect-addressing instruction in user space to access some parameters. The user substitutes a similar indirect instruction which increments an index register after each execution and thus creates a flaw.

3. Incorrect resource allocation/de-allocation.

   Example:

   - The same tape or core block is assigned to two users at the same time.

4. IBM OS/MVT

4.1 INTRODUCTION

The IBM System/360 family of computers was developed in the early 1960's to consolidate the divergence and incompatibility among IBM's three or more existing families of second-generation computers. The System/360 design provides for a large main memory, a uniform treatment of input/output (I/O) functions, non-stop operation under an operating system, and multisystem operation.

The central processing unit (CPU) has the following characteristics: 16 general registers (15 of which may be used as accumulators, index registers, or base-addressing registers); binary addressing; fixed and variable field lengths; decimal and hexadecimal radices; bit manipulation; automatic indexing; and floating- and fixed-point arithmetic. There is a selectively maskable interruption system, main-storage protection keys, and four alternative CPU states: stopped vs operating; running vs waiting; masked (uninterruptible) vs interruptible; and supervisor vs problem states. The regularity and clarity of the System/360 instruction architecture reduces the probability that system programmers will misunderstand or forget about an instruction effect. This in turn diminishes the possibility of integrity flaws arising from instruction side effects. On the other hand, instruction limitations in the

management of base addressing and in I/O programming and error sensing have tended to complicate some elements of systems programming, which increases the possibility of integrity flaws arising from programming complexity.

The operating system discussed in this report is the OS/MVT system (Operating System for Multiple, Variable Number of Tasks) which consists of a set of service and control programs, including a master scheduler, job scheduler, and supervisor. The master scheduler handles all communications to and from the operator, whereas the job scheduler is primarily concerned with job-stream analysis, I/O device allocation and setup, and job initiation and termination.

Central control resides in the supervisor, which has responsibility for storage allocation, task sequencing, and I/O monitoring. Provision for this control is embodied in the following concepts:

- Supervisor mode (CPU supervisor state) with associated privileged instructions.
- Storage protection to ensure the supervisor's survival.
- Hardware monitoring of program instruction violations.
- A CPU wait state available to the supervisor (as opposed to a stop/halt instruction available to the applications programmer).

4.2 OVERVIEW OF OS/MVT HISTORY

OS/MVT was not designed to prevent deliberate user-tampering with the operating system. Instead, an "accidental-error" philosophy was implemented, specifying that the operating system would attempt to protect itself and other users of the system from common "accidental user errors," but there would be no explicit attempt to protect against a user deliberately trying to interfere with the operation of the system. Concern about this philosophy was evinced as early as 1964, when the Systems Objectives Requirements Committee [4] of the IBM SHARE users group stated:

> "The Committee is concerned with the problem of maintaining security of data, particularly for those systems which allow for multiple program execution. This encompasses both governmental and corporate security, the former represented by classified and top secret data, and the latter by management information systems.
>
> Members of the Committee have investigated this problem both inside and outside of IBM, and very little seems to be being done for a variety of reasons."

In addition to the accidental-error philosophy, OS/MVT (in common with most other third-generation systems) does not have a clearly and systematically defined interface with the user. The variation in such an interface tends to increase the probability that integrity flaws will be introduced during implementation and to increase the difficulty of systematically correcting such flaws.

Several installations, in the ensuing years, have made extensive modifications to OS/MVT to enhance its integrity in the presence of deliberate user attempts to penetrate the system. Two such installations are McDonnell-Douglas Corporation and Cornell University. IBM developed an experimental Resource Security System (RSS) which was field tested at the Massachusetts Institute of Technology; TRW Systems, Inc.; and the Management Information Division of the State of Illinois. System integrity was considerably enhanced in this system, although some known integrity flaws were not addressed [5]. The OS/VS2 Release 2 (OS/MVS) for the newer IBM/370 line of computers has attempted to close systematically all known integrity flaws and has made a committment to fix any integrity flaws uncovered in that system. The design approaches used in the MVS system are relevant to installations concerned with OS/MVT integrity flaws. These design approaches are discussed by W. S. McPhee [6]; portions of which are paraphrased in Operating System Integrity Flaws, section 4.3.

The following paragraphs briefly discuss the IBM/360 and OS/MVT security approaches in the categories of prevention (isolation and controlled access) and integrity monitoring.

4.3 IBM/360 AND OS/MVT PREVENTION CONCEPTS

The first step in preventing impairment of system integrity is to isolate users from each other and from the operating system. The isolation features of the system are storage protection, program interrupts, tape/disk write protection, and privileged instructions.

a. Hardware Isolation Features

Storage protection prevents currently operating CPU or I/O channel programs from intruding into other programs and their associated data areas. A number of different main storage regions should be accessible and distinguishable from each other because I/O channel operations may be related to latent CPU programs rather than the program currently being executed by the CPU. Storage protection is realized by providing each of the 2048-byte blocks of storage with an 8-bit register (3 bits unused). The monitor may store any 4-bit combination into any one of these registers. The communicants with storage (namely the CPU, each selector channel, and each multiplex subchannel) are provided with independent 4-bit key combinations by the monitor. These key assignments are divided into two classes, zero and nonzero. The zero key, considered as the master key to all locks, is assigned only to appropriate sections of the monitor. The protection function applies only to operations that store into a block. Storage takes place only if the key and lock combinations match or if the master key is used. Otherwise, the store is inhibited, and a program error interruption occurs. Fetch (read) protection is supported by the hardware-storage-protect feature (as the fifth bit in the register), but is not used by the operating system. (This permits any user to read all the contents of main storage.)

There are a number of basic control instructions, such as those initiating I/O or changing program status, which can be executed only in the supervisor state. Any attempt at execution by an application program results in a hardware "program error" interruption.

Instruction and data formats are checked for correctness as the instructions are executed. This policing action distinguishes and identifies instruction errors (such as addressing or operation exception errors) and machine errors. Thus, instruction errors cannot cause machine checks; each of these types of error causes a different type of interruption.

The way an application program requests services from the control program is through an interruption. The preferred method is through execution of the supervisor call instruction which causes a supervisor-call hardware interruption. Although the supervisor call instruction is preferred, some IBM and installation subsystems have implemented other methods of generating interruptions to request control program services.

Magnetic tapes have a detachable plastic ring whose presence or absence is sensed by the tape read-write unit to permit or prevent writing on the tape. Some direct-access devices may be equipped with a manual switch to achieve the same results. The operating system uses a file mask I/O order (i.e., channel command) to prevent an I/O operation from reading or writing beyond assigned limits.

There are a number of operating system integrity flaws that permit subversion of most of these hardware isolation features.

b. Control Access Features

Although there are installations whose computing requirements can be satisfied within an environment of complete user isolation from each other, it is far more typical for users to have requirements to interact with each other — sharing resources such as programs or files. This ability to share selectively through authorized access to specified resources is called controlled access. Controlled access relies upon an effective system of isolation, without which access can not be controlled. In

common with many other third-generation operating systems, the OS/MVT design philosophy introduces very few constraints upon unlimited access, leaving most such constraints to be implemented by administrative and operator procedures.

The OS/MVT file system includes a Volume Table Of Contents (VTOC) on each direct access volume and of an optional tape label for tapes. These labels provide for password controlled access and for expiration-date controls that require permission from a system operator for a user to write into the file before its expiration date. There is a central file index (CATALOG) which lists the name of each cataloged file and gives its volume identification (serial number). Any entry in this CATALOG is available for legitimate modification by any user.

The IBM-supplied, data-set (file), password-protection scheme is not used by most installations because it is procedurally cumbersome to use and because of limitations in the security it provides. A number of installations (e.g., Yale University) have implemented their own version of data-set password protection.

c. Integrity Monitoring and Surveillance

In keeping with the OS/MVT accidental-error philosophy, integrity monitoring (assurance that the system is working as intended) and surveillance (the monitoring and recording of system activity) are not oriented toward the detection and identification of deliberate user-tampering with the operating system or with other user's files.

The most comprehensive of these monitors is the System Management Facility (SMF). It is possible to generate more than a hundred types of records; the installation using system generation parameters can specify those types of records it wishes generated. Most installations use purchased software packages or installation-written programs to format and summarize the thousands of records SMF produces daily. SMF records can be produced for the following categories of data: job accounting, data set (file) use, disk space use, device use by job step, program calls, and installation-captured data. While SMF appears to offer a detailed and useful security audit trail, its usefulness is compromised by three factors. First, the SMF output files are no better protected than any other system file, thus permitting their unauthorized modification by a user. Second, a user who has penetrated the integrity of the operating system can prevent the SMF programs from recording information about his activities. Third, a system crash will prevent the recording of, and cause the loss of, the most recently produced records. While SMF data may thus be compromised, it will not necessarily be compromised. Therefore, SMF data must be reviewed by an installation for any indications of unusual activity — such as a statistically significant increase in the occurrence of invalid timesharing passwords (e.g., Time-sharing Option passwords).

4.4 SUMMARY

The security of OS/MVT can be enhanced in three ways:

- Operational: An installation can choose to execute applications in a single-thread, stand-alone mode in which it is assumed that each program being executed has full access to all data and programs in or stored on the system, including accounting information.
- Applications: All access to the system is through one or more designated application subsystems that the installation has audited and is confident can not be subverted. No other activity (such as program testing or compiling) can take place concurrently, and all data to be protected (including the trusted version of the operating system) are removed from the system before other activity is allowed.

- Systems: Extensive programmed modifications are undertaken to eliminate most known integrity flaws and to monitor system integrity. This will increase the effort and risk required to subvert the system.

4.5 OPERATING SYSTEM INTEGRITY FLAWS

Installations running OS/MVT should be aware of the following two specific integrity flaws. These flaws are discussed because it is relatively easy to counteract the effects of these flaws. Together with similar integrity flaws described in Section 5.6 and 6.6, these flaws also serve as specific examples of the classes of integrity flaws described in the taxonomy (Section 3). The generic description of the flaw is taken from the taxonomy. The reader should not assume that these two flaws are typical of the other integrity flaws that are known to exist in OS/MVT. Easily exploitable flaws are not described in this document unless they are no longer applicable to currently operating systems.

- Generic Description: Asynchronous validation/inadequate serialization.
  Specific Description: The checkpoint data set produced by the checkpoint/restart facility contains sensitive system data. This data set is not protected from modification (or counterfeiting) by the user prior to its use by the restart facility.
  Security Enhancement: One option, if feasible, is for the installation to remove the checkpoint/restart capability from the system. If this is not feasible, control mechanisms must be developed to require:
  - System-operator validation of checkpoint files.
  - External labeling procedures for checkpoint volumes.
  - Off-line library control over access to checkpoint files.
  - Prohibition of I/O to checkpoint data sets, except via the checkpoint SVC and authorized system utility programs.

- Generic Description: Inadequate identification/authorization/authentication.
  Specific Description: It is possible to bypass tape label processing even if this function is included in the system by specification during system generation. For example, a user may change the label-type, which is specified in a system control block, before the tape file is opened and the label accessed.
  Security Enhancement: The system can be modified so that important parameters which describe user files are maintained in memory protected from the user. This may be combined with an automatic-volume-recognition package acquired commercially or from another installation. Alternatively, naming conventions and logs for external tape labels can be adopted such that the system operator will only mount tapes authorized to the requesting user.

5. UNIVAC 1100 Series Operating System

5.1 INTRODUCTION

The UNIVAC 1100 Series Operating System operates any of the three 1100-series computers, the 1106, 1108, and 1110. It combines multiprogramming, multiprocessing, timesharing, communications, and real-time systems into a complete set of software — from basic service routines to compilers. The major subsets (with some examples shown in parentheses) are:

- Executive system.
- System processors (collector, FURPUR, SECURE).
- Utility system processors (FLUSH, CULL, DOC).
- Language processors (FORTRAN, COBOL, ASSEMBLER).
- Subroutine library (FORTRAN library, SORT/MERGE).
- Applications programs (GPSS, PERT).

The executive system is responsible for controlling and coordinating the functions of the internal environment. By using multiprogramming and multiprocessing, it handles batch processing, demand processing (timesharing), and real-time processing. Since installations do not have the same needs and requirements, each capability supplied by the executive may be eliminated by an installation during system generation. Permanent data files and program files are kept on mass storage devices with some security measures to ensure that files are not subject to unauthorized use.

The executive system consists of several classes of routines, grouped by the functions provided.

- A set of control statements are provided for directing the execution of individual tasks of a job and for relaying operational information about the job to the executive system.
- The supervisor component controls the sequencing (coarse scheduling), setup (dynamic allocation of storage space), and execution (central-processor-unit dispatching) of all jobs.
- Available facilities are assigned by the executive as needed to fulfill the requirements of all runs entering the system.
- File-control routines allow file manipulation without concern for the physical characteristics of the recording devices.
- Operator-communication functions display system status information to the operators.
- Input/output device handlers and symbionts control the activities of all I/O channels and peripheral equipment attached to the system.

5.2 DESIGN CRITERIA OF THE OPERATING SYSTEM

Because of the flexibility of the hardware and software configurations, the 1100 Operating System can handle both scientific and business jobs. Jobs may be submitted via any one of three ways: central job-entry terminals, remote job-entry terminals, and demand terminals. Once submitted, a job may be handled in any of three modes: real-time, batch, and demand processing. The differences of the three operating modes are not in the way data is handled but rather in the priority and queuing of the job's tasks for dispatching.

UNIVAC (in the 1100 Series Operating System Programmer Reference Manual) lists the following as fundamental design criteria:

- Individual components operate reentrantly whenever possible.
- Components and data areas are permanently resident in main storage only if nonresidency is either impossible or would impose an unacceptable overhead.
- The central processor units (CPU's) are in general treated equally.
- Components must be able to execute on any CPU.
- Wherever feasible, real-time requests are given top service priority.
- Interrupt lockouts and software interlocks on interrupt-related data must be kept to a minimum.
- Most executive system components operate as ordinary activities and are managed in the same way as user activities.
- Service requests by an individual user are to be validated to whatever extent is necessary to eliminate undesired interaction with the system and other users.
- Code for optional hardware components and software capabilities should be written such that it need not be generated if the associated component or capability is not configured.
- Each type of peripheral (e.g., teletypewriters and printers) has an associated symbiont, a peripheral-handling routine. The symbionts buffer large amounts of data from low-speed peripherals to main storage to minimize the number of data transfers.

5.3 1108 ARCHITECTURE

The CPU's are interfaced to the I/O peripherals by I/O controllers (IOC). As a result of the IOC, the CPU does not concern itself with the physical capabilities and limitations of I/O devices. The CPU and the IOC are connected to the main storage device. Data transfers are usually buffered to increase efficiency. There is a hardware option, back-to-back block transferring, which allows data manipulation to be handled completely within main memory without the involvement of the CPU once the process has been initiated. This leaves the CPU completely independent during long data transfers. (See figure 5-1.)

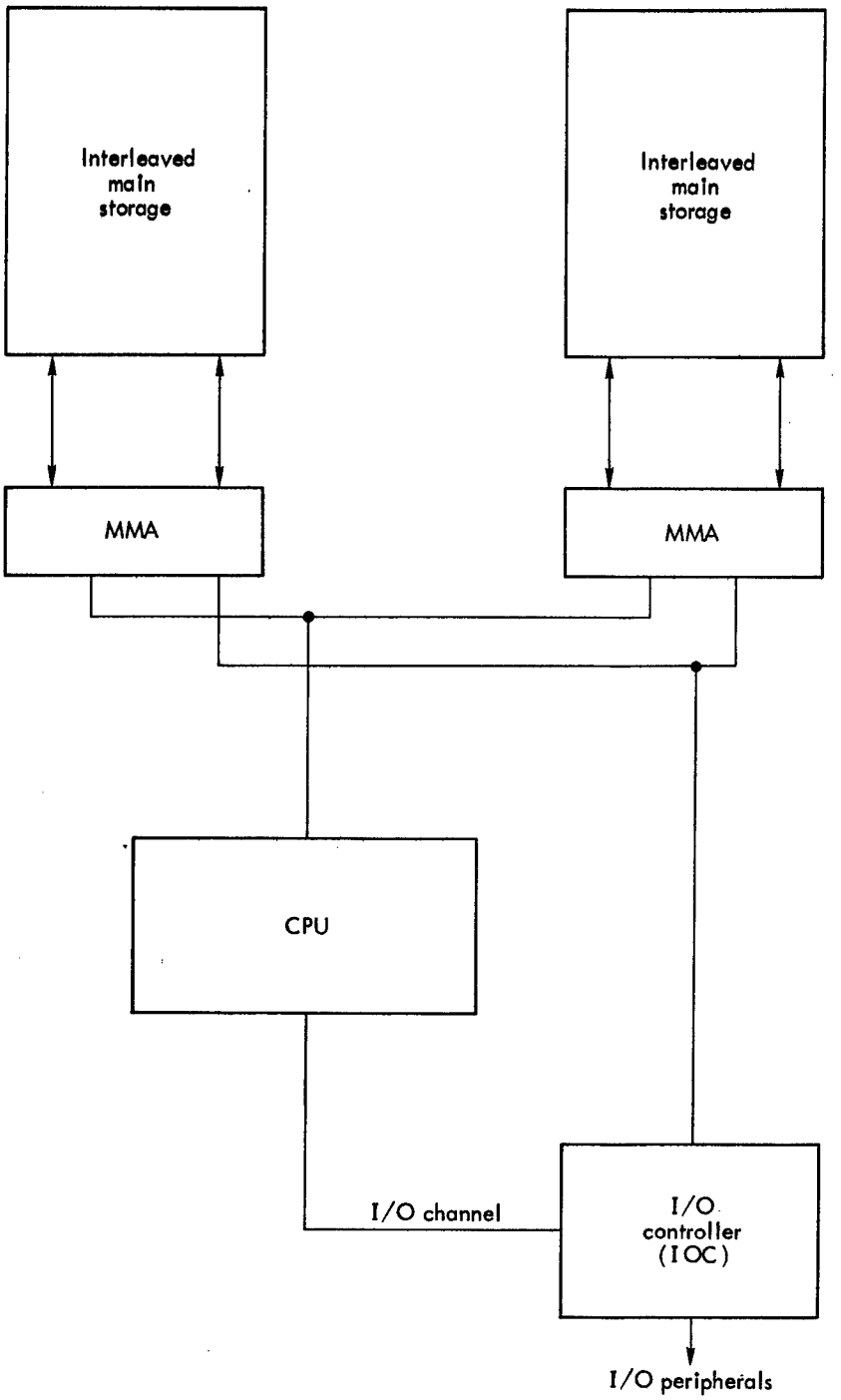

Figure 5-1. CPU, IOC, and interleaved main storage with MMA's for UNIVAC 1108.

a. Memory Interface

The CPU's and the IOC's are interfaced to the storage units by the multimodule access (MMA) units. There is one MMA for each storage unit. The UNIVAC 1108 may have from one to four storage units. A storage unit provides 64K words of main storage (36-bit words). Hardware implementation exists to provide address interleaving for greater efficiency. An MMA is used by the CPU's and IOC's to request and receive access to the corresponding storage unit. When a conflict occurs between two or more access requests, the MMA services the requests on a priority basis.

b. System Control

The majority of the control activities are handled through the 128 control registers, the interrupt system, and the queuing system. Dynamic allocation is serviced in order of priority and by the queue ordering within each priority. The first request arriving at priority x will be handled the moment there are no requests of a higher priority. The dispatcher, controlling CPU usage, operates in a similar fashion, i.e., switching consists simply of taking the highest priority activity from a list (called the switch list) of all activities currently requiring CPU service. When an I/O interrupt occurs, the type of interrupt and the channel on which it occurred is either queued or routed to the appropriate processing routine. When the interrupt is queued, the priority of the interrupt helps determine the length of the time before servicing.

The availability control unit (ACU) provides hardware configuration control. It interfaces with the CPU's, IOC's, storage units, and MMA's for up to 24 peripheral subsystems. The ACU provides for:

- Dividing the entire system into independent systems.
- Disabling CPU's or IOC's whenever a power failure occurs.
- Taking a unit off line for maintenance without impacting the operation of the remaining units.
- Initiating automatic recovery upon system failure.
- Maintaining an availability table for all units.

5.4 INTEGRITY FEATURES

a. User Control

The basic program entity for which storage allocation is required is called a bank. When in main storage, a bank occupies contiguous physical addresses. In the logical program sense, a bank is a collector-defined portion of the program that is specified by a single bank-descriptor word (BDW). (The collector functions as a loader or linkage editor.) By this collector definition, a program is logically organized into two banks: instruction banks (I-banks) and data banks (D-banks).

The profile of the entire task with all of its associated banks is maintained in the program control table (PCT). The portion of the PCT controlling the banks is called the bank-descriptor table (BDT). In the BDT are the BDW's describing each bank belonging to the program. The BDW's contain bank control information such as whether or not the bank is read-write protected and what are the upper and lower boundaries of the bank area.

A program area is thus described and controlled by the PCT. Within the PCT are two control words: the program state register (PSR) and the storage limits register (SLR). The PSR contains the guard mode and storage protection designators. Main storage protection can be established in either one of two modes. One mode (exec mode) affords write, read, and jump protection; the other (user mode) affords protection only when writing. A program may run in either of two modes. Exec mode allows the program to use privileged instructions; user mode does not. The SLR contains the upper and lower absolute

address limits of main storage in which the active I-bank and D-bank of the currently operating program are located. Thus the type of storage protection and whether or not it is enforced depends on the condition of the guard mode and the write-only storage protection designators in the PSR.

b. States of Execution

There are four states in which a program may execute.

- The program runs in exec mode with no storage limits enforcement.
- The program runs in exec mode with storage limits enforced.
- The program runs in user mode with no storage limits enforcement.
- The program runs in user mode with storage limits enforced.

The last state is the only "safe" state with regards to security. A user job getting into any of the first three states can cause a security compromise.

c. Protection of Permanent Files

Cataloged files are protected by the concept of keys: read-key required, write-key required, both read and write keys required. In core, files are protected by storage limits, read-only designator, and write-only designator. Files, in general, can be manipulated by using the file utility routines (FURPUR) supplied by the system. A user can also write his own file manipulation routines.

Physical security of cataloged files on mass storage is provided by the SECURE processor which produces backup tapes. Most files can be saved by SECURE. Certain files are marked with "unload inhibit" which will not allow them to be removed from mass storage. These unload-inhibit files are determined by real time or other special considerations. There is also a guard option for reading that even prevents the privileged read necessary to make the backup copies. This option is required for certain special files which are internal to the system and are either highly transient or highly classified. A checksum is optionally provided to check the backup copy. With these backups, no file will lose more than a certain amount of processed data if the system crashes with some kind of data loss. SECURE tapes are handled by SECURE routines only. These tapes should be stored separate from user tapes. In other words, there must be a SECURE tape vault, exclusive of user tapes.

d. Protection of Magnetic Tapes

All magnetic tapes are "exclusively assigned." That is, two different jobs should not be able to access the same tape at the same time. Each tape is identified by a series of header labels. The volume header identifies the reel number and the owner of the reel if it is private (as opposed to public). The file header contains information such as filename, file generation and version numbers, creation and expiration date, and access control designators. When the operator mounts and loads a tape, the system should check the headers against the tape request if the tape is labeled. (The earlier releases of the 1100 Operating System did not perform an adequate check; however, the later releases do.)

A tape-labeling system is necessary to offset any operator error which may occur. If such a system is not available, more procedural checks must be added to double check for human error. One possibility is for the operator to always copy the user identifier and the reel identifier into a tape log book, then to take the tape log book to the vault librarian who then must initial the log book before giving the tape to the operator. (Ownership of tapes must also be checked.) There could be a log book at the vault for the operator to initial. With the incorporation of all of these procedures,

the possibility of using a wrong tape is greatly reduced. The overhead involved with these procedures is time and the cost of log books. Alternate methods may be more reasonable for an installation with few tapes.

e. Audit Trails

While the log books record tape usage, the system maintains logs to record system usage. Audit trails (log entries) record the following: 1) job initiation and termination statistics, 2) user-specified log control messages, 3) console messages, 4) I/O errors, 5) symbiont activity, 6) tape-labeling information, 7) checkpoint/restart information, 8) facility usage, and 9) cataloged mass-storage file usage.

The total amount of time used by a job is broken down into the various subsystems used. A log entry is created and subsequently inserted into the master log whenever the configuration of a job is changed by assigning or freeing a tape or an arbitrary device file, or whenever a cataloged file is created, assigned, or freed. The actual usage, or lack thereof, of a file or tape is not logged since the mere act of assigning a file does not imply usage. In fact, a user may assign any file or tape. The access control is not checked until the access is performed. Thus, the "usage" entries in the master log list the reservations made by the job for various types of storage and facilities and the amount of time spent using each type. These entries do not adequately describe the actual access made by the job.

The system master log is updated at intervals. Until update time, the logging information is kept as a temporary file, and temporary files are destroyed by a system crash. Therefore, a user can get on the machine, perform unauthorized actions and be undetected (i.e., not recorded on the master log) if a system crash occurs. Some installations have incorporated their own audit trails into the 1100 Operating System. If a larger overhead is acceptable, more specific information must be added to detect an intended breach of security. The presentation of the information should be directed toward the person (not necessarily a system programmer) responsible for examining audit information.

The use of the audit trail raises the questions of who and how often the log entries are examined. Administrative procedures need to be implemented to ensure that the audit trail information is examined in a consistent and timely manner and that appropriate actions are taken. An example of such a procedure is the existence of a "security officer" whose console output are the highlights of the log. The important entries (as defined by the administration) are printed onto the security console. The officer's task is to use this information in detecting breach attempts. The presentation of log entries to a security officer should be easier to read than are the current log tables. A list of times and files used may not be significant to the security, while a message indicating a guard mode fault from user xxx at terminal yyy would get his attention.

Implementing these administrative procedures will require a redesign of not only the logging system but also part of the interrupt system. Currently, a user program may supply an interrupt-handling routine for software errors so that information would not be passed to the log routine. Thus, a guard-mode fault would not be logged. The interrupt must be logged before control is passed to the user's interrupt handling routine.

f. Role of System Console Operator

Just as the tape vault procedure can be used to identify a user's access rights to a tape, a system console operator could have a procedure to determine when or when not to honor a request for operator intervention. Messages from the system would be formatted so that they contain flag characters which a user could not duplicate onto the system console as a spoofing mechanism. Once the operator

is sure that the system is the requestor, he can safely comply. If the requestor is a user, the operator must check two points: 1) the user is really who he says he is, and 2) the user is authorized to request that particular action. An example of this process is the authorization of a job that requires operator interaction. The operator would have a list of all users allowed this privilege and would confirm that the request came from that user rather than from another user masquerading as one of the authorized persons. These procedures will differ depending upon the installations' needs and requirements. Each installation should decide upon a set of rules and enforce them.

g. Impact of System Degradation

Another installation-dependent procedure is the handling of a system crash. Certain installations do not consider a system crash a security problem. Other installations view a crash as a reliability problem, a matter of poor throughput. Still others worry about system crashes and recoveries because of time-sensitive data processing or the loss of information.

Assuming that a system crash is serious with respect to security, a procedure should be established for system recovery. Before recovery begins, steps must be taken to identify, as specifically as possible, the reason for the crash. This is not an easy task. The system itself may be in control at the time of the crash even though it was caused by a user. (For example, he requested a confusing sequence of tasks, supplied bad data, etc.) Where possible, the user job responsible for the crash must be identified. A decision by a "person of responsibility" is made as to whether malice was meant or an accident happened. The security officer's console log might be one tool used for deciding. It must be pointed out that this procedure will make system recovery take longer.

It is possible for a user to cause a system crash after making spurious files. There is a chance, therefore, that the recovery system may be fooled into giving these spurious files more privileges, more data, etc. This would be a security breach.

Solving this problem involves the redesign of the recovery system and possibly of other portions of the normal system.

5.5 SUMMARY

The UNIVAC 1100 Series Operating System was not designed with operating system security in mind. There are a number of known flaws that can be corrected only by redesigning parts of the operating system. Certain parts involve a large amount of code, and changes in these areas would greatly impact the design of other areas. One such area is the file management complex of routines.

Although not all areas lend themselves to redesign, a large number of the problems are fixable. Consider as an example the problem of file protection in this system. To protect the files from unauthorized user access, an access table can be implemented. Because the table size must be limited, not all user files can be protected in this way. However, not all files are sensitive in nature. Thus, a user can assess each file's protection requirements and submit his sensitive files for this kind of protection.

Problems that would involve complicated software changes may have solutions in hardware applications. For example, bank protection can be write-protected by introducing an add-on hardware unit into the system.

Still other problems lend themselves best to procedural solutions. Restriction of access is indeed the most straightforward of solutions. Assuring that all users with access to the system are legitimate (by physical restrictions, by imposing heavy identity checks, or by a combination of the

two) allows the system to relax its mistrust of users. The ultimate in procedural protection mechanisms is the restriction of computer operation to serial (batch) service. That is, only one job is allowed on the system at any instant of time.

Timesharing and concurrent job processing introduce complications that the standard UNIVAC system is currently unable to adequately provide protection. Procedural, hardware, and software fixes can be introduced (and some UNIVAC sites have done so) to reduce the probability of computer mismanagement to a level of "acceptable" risk during concurrent processing.

5.6 OPERATING SYSTEM INTEGRITY FLAWS

The following examples have been carefully chosen because they illustrate security problems yet involve a minimum amount of exposure or risk to installations. The first two integrity flaws can be fixed at the option of each installation. The final three flaws appeared in early versions (before level 31) of the operating system and have been corrected by Univac (as of three or more years ago). They are listed to illustrate the types of security problems that occur in computer operating systems.

- Generic Description: Inadequate identification/authorization/authentication.
  Specific Description: The protection mechanism of reentrant processors (REP's) cannot write-protect the REP's I-bank.
  Implications of Flaw: If the I-bank of a REP is not write-protected, a user program can attach to the REP, install a Trojan horse, and then release the REP back into circulation. Future calls to the REP will trap the callers.
  Security Enhancement: UNIVAC has developed a hardware attachment (two cards per CPU) which implements proper write protection for the I-banks. A switch is also provided to allow access by systems personnel to run debug shots.

- Generic Description: Implicit sharing of privileged/confidential data.
  Specific Description: Portions of core are not cleared before the user has access to the areas in question.
  Implications of Flaw: The user given uncleared storage can inspect the residue left by the system or by another user. Information he can get may include file access words, passwords, keys, EXPOOL information, etc. There is an option for the collector that allows the user to specify core sections not to be cleared before release to him. There are also at least two instances of EXPOOL information leakage. Both are recoverable by the user, allowing him to scavenge for system control information.
  Security Enhancement: This collector option (B-option) must be removed. The default of "clear" must be the rule. CLRCOR is a configuration parameter which can be set at system generation so that the system always clears core.

- Generic Description: Inadequate identification/authorization/authentication.
  Specific Description: The check for the system file name is inadequate.
  Implication of Flaw: Access to the system file SYS$*DLOC$ specifies the run as being "privileged." To determine if a job is privileged, the system concatenates the first three characters of the qualifier name with the first three characters of the filename. This six-character result is compared against the string "SYSDLO". If the comparison yields equality, the job is assumed to be privileged. Therefore, a job with access to the file SYSA*DLOA is considered privileged. Any privileged run can access the text and the master file directory information of any file catalogued in the system without supplying any of the access keys.
  Security Enhancement: The entire name of the file and its qualifier must be checked. This check must be expanded to include the entire qualifier (SYS$) and the entire filename (DLOC$), instead of just the first three characters of the qualifier and filename.

- Generic Description: Implicit sharing of privileged/confidential data.
  Specific Description: In systems released before level 27, register R1 contained the access key for the I/O instructions.
  Implication of Flaw: A user can write into register R1. An all-blank keyword is the access key that allows any request to be honored. A user who writes blanks into register R1 can then issue I/O requests to gain unauthorized access to system information and to data belonging to other users.
  Security Enhancement: A control word can be defined in system space to hold the access key. All checks on the I/O requests should be compared against this control word.

- Generic Description: Exploitable logic error.
  Specific Description: A user can load a bank before his access rights to it are checked.
  Implication of Flaw: A file pointer is updated too early. The user can access banks to which he has no legal rights.
  Security Enhancement: The user's access rights must be checked before updating the file pointer. The code is already in the LIJ/LDJ section of the system. The two sets of instructions need to be interchanged.

6. Bolt Beranek and Newman TENEX

6.1 INTRODUCTION TO TENEX

The TENEX operating system was designed by Bolt Beranek and Newman Inc. (BBN) with support by the Advanced Research Projects Agency (ARPA). The original intent was to develop a medium cost system employing state-of-the-art virtual memory hardware and software, user-controlled multiprogramming, and a flexible user-controlled software interrupt system to serve the research groups at BBN. The one need that forced the development of the new system more than any other was the need to run large LISP programs, which typically require a rather large and scattered working set.

To obviate the need for developing a completely new set of utility programs, TENEX was designed to be hardware and software upward compatible with Digital Equipment Corporation equipment, the DEC 10/50 system. By having DEC UUO's (DEC system calls) cause loading of a user interpreter to perform the equivalent TENEX JSYSes (TENEX system calls), TENEX was able to implement a complete new set of system calls and still maintain the DEC 10/50 compatibility with a minimum addition to the security kernel of the operating system.

6.2 TYPICAL USE OF TENEX

Figure 6-1 shows a system status listing for a "typical" TENEX system. The listing gives the reader a visual picture of the typical use of TENEX. A description of how the listing was generated and what the various fields represent should help it to convey the maximum amount of information. This description follows:

1) To generate the listing, a user on one TENEX system ran a TELNET program that allowed him to connect to the "typical" TENEX system. The TELNET program has a feature (discussed in section 6.6.b with respect to a security problem) that allows it to accumulate the terminal output from the remote system in a disk file. This disk file, the TELNET typescript file, was later edited into this report.
2) Starting at the top of the listing is a line of header put in by the TELNET program. The "#" is the first character of terminal output and is the prompt character from the TELNET program. The user then typed "tenex" followed by a carriage return (<CR>).* The TELNET program completed the connection through the ARPA network as it typed "is complete" and the initial herald by the remote system follows the "#" delimiting the end of the TELNET programs' command output. In the case of TENEX, the herald indicates the TENEX version number (here 1.33.16) and the EXEC version number (here 1.53.21).
3) The "@" is the prompt from the remote TENEX's EXEC language processor (EXEC). The user then typed "login username password <ESC><CR>". The password doesn't echo, and the system recognizes the default account string for this user when the user typed the ASCII escape character (<ESC>).
4) The user was then logged-in, and he got some information about his status, a note as to when he last logged in (to detect password leakage problems) and was given the prompt "@" by his now logged-in EXEC (discussed in section 6.4.c, Process Protection). He then typed "ld<CR>" which generated the status listing, and he typed "logout<CR>" after the last "@" prompt to logout. An ASCII SUB (control-Z) tells the original TELNET that the user wants to talk to it, again eliciting the "#". The user then disconnects and quits leaving the typescript file.
5) As far as the load status itself goes, after some load statistics (e.g., an average of 6.31 processes are currently requesting the CPU), the load status lines are generally broken up into four fields: job number, terminal number, user name, and a variable length field. The variable length field includes: a) the subsystem in use (default is EXEC); b) the user's local network site, shown in parentheses "()" (default is a local user); and c) the user's connected directory, shown in angle brackets "<>" (default is the user's login directory). Most of these terms are discussed in detail later where they relate to security issues and are noted here simply for later reference. The only lines that are not of this format are lines that do not start with a number (the job number). These lines indicate active ARPA net connections and are not pertinent to this discussion.

*Characters in angle brackets (<>) indicate ASCII standard characters.

```
TELNET typescript file started at TUE 18 MAR 75 1937:27 EDT
#tenex is complete.#
TENEX 1.33.16. EXEC 1.53.21
@login risos  LLL
 JOB 9 ON TTY113 18-MAR-75 16:39 PDT
 PREVIOUS LOGIN: 18-MAR-75 14:12 PDT

@ld
 Load   6.31  5.19   5.05
 Up 32:44:36  25+2+5 jobs  12% idle
Jb TTY  User
34 114* not logged in                          :Discussed with EXEC.
 9 113  Risos     LD.(LLL-RISOS)               :The load status program itself.
20 112* Smith     RSEXEC.(AMES-TIP#40)         :An RSEXEC problem is discussed.
46 111* Jones                                  :Blank indicates EXEC use.
42  53* Lab       TELNET                       :A TELNET problem is discussed.
 OPND #3/2 <> BBN-TENEXB:2#10/1. B8            :An active network connection.
48  52  Jones     NLS                          :A fancy tree structured editor.
25 50*  Risos     FTP.(LLL-RISOS)              :The file transfer program.
 OPND  102/3 <> BBN-TENEX:7 3001/0. B8
 OPND  104 <- BBN-TENEX:7 3003. B36
30  45  Friend    BSYS.<Isis>                  :The backup archival system.
40  43* Taylor    LISP.<Speech>
 3  42* Campus    SPELL                        :A spelling correction program.
17  40* Sysjock   PRINTR.<System>              :System printer control.
28  37  User      (PRIV)                       :A user's unnamed program.
21  36* Aistuff   LISP
13  32* Edit      TECO                         :A simple powerful editor.
38  26* Assembl   MACRO                        :An assembler.
41  24* Fast      TVEDIT                       :A fast display type editor.
12  23* Report    RUNOFF                       :An old report generator.
45  22* Nbs
36  21* Risos     EXERCI                       :The RISOS system Exerciser.
29  17* Number    F40                          :The Fortran compiler.
 8  16* Bart      SOS                          :An unknown subsystem.
16  14* Alongname.XOS
19  11* Bart      READMA                       :A subsystem to read messages.
24  10* Friend    SNDMSG                       :A message sending facility.
23   2* Macll     MACN11                       :A PDP-11 cross assembler.
39  Det Batch                                  :A detached job.
26  Det Wait      TELSER.<Net>
 5 (LDINF)           <Loadstat>
 4 (RSSER)                                     :RSEXEC server discussed.
 CLZW #0/1 <> BBN-TENEX:40#3/2. B36
 2 (BATCON)          <Batch>
 1 (PRINTR)          <Printer>
 0 (SYSJOB)
 OPND #3000/1 <> BBN-TENEXB:35#103/2. B8
 OPND #3002 <- BBN-TENEXB:35#105. B8
@logout
LOGOUT JOB 9, USER RISOS, ACCT LLL, TTY 113, AT  3/18/75 1641 PDT
  USED 0:0:5 IN 0:2:14
#disconnect
#quit
```

Figure 6-1. Example of a system status listing for a "typical" TENEX system.

6) Anything after and including a ";" on a line was included as a comment and was edited in for this report. Also some subsystem use was modified to further typify the status. Some names were changed, and some lines were deleted to shorten the listing.

As an example in hand of typical TENEX use, the initial entering, editing, spelling correction, storing, and network mailing of this section were done with TENEX.

6.3 OVERVIEW OF TENEX HARDWARE ARCHITECTURE AND INTEGRITY FEATURES

a. CPU

The TENEX operating system can currently run on a DEC KA10 processor with a BBN pager or on a DEC KI10 processor. (The KA10, KI10, and KL10 are all DEC PDP-10 processors.) The initial implementation was on the KA10 for which BBN designed the pager. The KI10 version is selected at assembly time and is implemented by translating between the KI10 format and the BBN pager format for the page tables. The system upon which this report deals with is the KA10 version, so only the KA10 system will be considered. There is little loss of generality in thus restricting the report to the KA10 version of TENEX.

Probably the most important feature of the PDP-10 CPU that aids system integrity is the fact that the instruction set and I/O structure were optimized more for simplicity than for speed. The mechanism for computing the effective addresses used by instructions is uniform throughout the instruction set, thereby freeing the programmer and automatic analysis routines from the need to consider many special cases. Included in the instruction set are a complete set of Boolean operators, stack operators, variable-length byte operators, and flexible bit-testing and masking instructions. Many of these instructions are almost never used and many perform identical operations, but the instructions are organized in a way that makes them easy to understand and use. The I/O structure is also very simple and doesn't contain the sophisticated channel-program mechanisms that appear in many computer systems of about the same vintage.

One disadvantage of the PDP-10 is that its architecture does not admit a practical virtual machine monitor. There are several instructions that allow a program running in monitor mode to determine that it is really in monitor mode. Such instructions would have to be emulated in a virtual machine monitor, and no mechanism is available for singling out the particular troublesome instructions. This fact requires that a virtual machine monitor for the PDP-10 simulate all instructions, a mechanism that is impractically slow on the PDP-10 itself. A TENEX emulator was implemented on the MAXC at the Xerox Palo Alto Research Center (PARC) which made some otherwise difficult measurements possible, but this is a one of a kind system and the additional measurements are very limited. A virtual machine monitor for the PDP-10 itself would make TENEX development much smoother and make much more complete testing of the operating system possible.

The hardware modifications necessary to allow the KA10 to support a virtual machine monitor are fairly minor. The software task of writing a virtual machine monitor for the PDP-10, however, would be a major undertaking.

The JSYS mechanism that was introduced with the BBN modifications to the KA10 processor greatly simplifies the Monitor call mechanism. The most important part of the JSYS mechanism is the JSYS vector located optionally at a fixed location in real core or in a reserved page in the per-process region (discussed in section 6.3.b) of the Monitor's virtual space. (This option is a recent addition for JSYS trapping which is also discussed in section 6.4.c.) The JSYS vector contains the addresses of the routines used to handle the JSYS calls and the addresses of words to store the information necessary to return to the calling process. When coupled with the per-process mapping (Sec. 6.3.b

below), this mechanism makes monitor entry and exit from user space very simple. Unfortunately the Monitor recursion mechanism (using JSYSes within the Monitor) is still awkward. This recursion is once removed from the user, however, and is therefore somewhat less security sensitive.

b. Virtual Memory Hardware

TENEX is a virtual memory system with separate virtual spaces for the Monitor and the user. This fact reduces some security problems. For example, it is not possible for a user to put his program into monitor mode by having the Monitor return by mistake with the monitor mode bit set. If the TENEX Monitor did so, it would simply return to somewhere within the Monitor. This would likely cause a system crash, but would not allow the users program to run in monitor mode. The user can still attempt to put a program segment into the Monitor space somehow and then try to get the Monitor to jump into it, but this is a slightly more difficult problem that also exists on any system with write-modifiable or pageable code.

Another advantage of having a virtual memory map for the Monitor is that it is possible to write-protect the monitor code. TENEX currently makes some use of this feature.

The fact that part of the Monitor virtual space is mapped on a per-process basis makes the Monitor simpler and eliminates a class of security problems. Associated with each TENEX process is a page called the Process Storage Block (PSB). Contained within the PSB is the map for the Monitor's per-process area (the last 128 pages of the monitor virtual space). The PSB is pointed to by a pager reister, so it is very easy to change when rescheduling a new process. The kinds of security problems that this feature eliminates are those which can happen when the monitor confuses references to or fails to protect from the user-process specific information (e.g., the implied sharing of privileged data class of flaws discussed in the taxonomy in section 3).

Still another hardware feature that simplifies system coding and therefore makes it easier to secure TENEX is the operation of the monitor mode instructions which reference user space. On the PDP-10, the 16 general registers can be referenced as the low 16 words of a virtual space. These same registers are used for both Monitor and user spaces. Therefore, when the Monitor is entered, it has to save any user registers that are needed by the Monitor routines. To reference user space, the Monitor is given a special set of instructions. When the Monitor references a user address which happens to be between 0 and 15, the pager automatically causes this reference to go to a special saved register area of the PSB. In this way, if the Monitor has a user address to reference, it need not make a special check to see if it is between 0 and 15.

c. Peripherals

The most security sensitive TENEX peripherals are the paging drum and the disks, which are used for permanent file storage. Failures in these units have caused security problems in the past, but generally such failures only result in system crashes. The little fault tolerance built into TENEX for these contingencies should be augmented.

TENEX supports a wide variety of online terminal devices. The software that associates these devices with processes is fairly simple. No security problems were discovered in its implementation. Intrinsically, however, mechanisms are designed into TENEX which allow processes to assign a non-logged-in terminal device and to accurately simulate a non-logged-in EXEC. The way the mechanisms work, there is no way that a user starting to use a hard-wired, non-logged-in terminal can assure himself that the terminal is not assigned to a process which is simulating on EXEC. Terminal-like connections that are effected through the ARPA network do not have this problem because the network "terminals" are created and destroyed every time a connection is set up.

TENEX supports an extensive set of communications software. There are problems discussed in section 6.6.a concerning the auto-answer telephone facilities and the interface to the ARPA computer network. The ARPA network software presents a unique problem both intrinsically and in terms of the implementation of the interface multiplexing.

Intrinsically, the ARPA network (or any network of the same type) represents some unique problems because it allows arbitrary processes to communicate with arbitrary processes on other host systems. Traditionally, computer systems have kept a fairly tight control as to which processes were allowed to handle specific devices. Because the ARPA network is logically multiplexed to look like a very large number of devices, access to these pseudo devices is given out very freely. Any process (especially privileged processes) handling one of these pseudo network devices is maintaining a security perimeter and must be assured of correct operation to assure system integrity. (See the Privileged subsystem problem discussed in section 6.6.b for example.) Even an interactive terminal user using a process to connect to another site (e.g., the TELNET connection, in figure 6-1) is incurring the security risk that the remote system might spoof his local process into releasing sensitive information. If the user explicitly gives sensitive information to an ARPA network connection, he has no assurance that its integrity will be protected. All such network communication must assume malicious tapping and noisy communication channels, and therefore, combative techniques like encryption and error checking must be used.

The Network Control Program (NCP) that performs the logical multiplexing for the ARPA network connections is a complex program making auditing and correct implementation difficult. Also, there are many situations where it is natural for the NCP to trust the Interface Message Processors (IMP's) handling the communication network functions and even the remote hosts to follow network protocols and/or to perform in a generally reasonable manner. This trust can lead to problems.

There were several times during the course of the TENEX study that problems of one kind or another were traced to the NCP. The NCP is a complex and rapidly changing portion of the TENEX monitor, however, so it was not included in the main thrust of the TENEX security study and will not be directly discussed in the remainder of this report.

TENEX also supports a complete assortment of tape devices, line printers, card readers, paper tape equipment, etc. There are minor intrinsic security issues associated with many of these devices, but these issues are generally applicable to such devices on any system, and there were no outstanding novel implementations on TENEX.

There is one other peripheral-associated mechanism that deserves mention and that is the TENEX Backup System (BSYS). This is a magnetic storage system that is used as archival storage for TENEX. The system uses ordinary magnetic tapes and requires manual operator tape manipulation. This system is sensitive in that it has to multiplex the physical tapes among users. An incorrect archival retrieve could give one user another user's file. However, this system was not looked at in detail for the TENEX study, and no problems associated with it appeared after the initial bugs were shaken out.

6.4 OPERATING SYSTEM DESIGN AND INTEGRITY FEATURES

There are some terms that need to be mentioned before a discussion of the TENEX software design and integrity features can be continued. Included here are only those terms and associated information pertinent to the security and integrity discussion:

Item	Environment	Associated Items
Directory	system	password, directory protection, default file protection, directory group membership word.
File	directory	protection bits.
Job	system	login directory, connected directory, job process structure, job file structure.
Process	process, job	process capabilities, inferior process structure.
User	system	login directory (and associated password), user group access word, initial capabilities.

The discussion of TENEX software security and integrity will be broken into three parts: file protection, directory protection, and process protection.

a. File Protection

In TENEX, any file can be named and access-requested by any user. Five kinds of access are recognized: read, write, execute, append, and list. List access is requested when a user asks for the name of a file from the system (e.g., by trying to list the file's name in a directory). Append access is requested when a user attempts to write beyond the end of a file. The others are self explanatory.

Each file contains 18 bits of protection information. This information is broken up into three fields called SELF, GROUP, and OTHER. Each field has one bit to denote each of the five accesses recognized (plus one unused bit). When a user attempts to access a file, the appropriate field is chosen as follows:

1) If the user is connected to the directory containing the file, then the SELF protection field is used to validate access. A user is initially connected to his login directory and can only connect to other directories by owning them (see the discussion of directory protection below) or by knowing their password.
2) TENEX has 36 user groups that are set up administratively (the user has no direct control over them). Associated with each user is a 36-bit user group access word. The user's group access word contains a 1 bit for each group that the user has GROUP access to. Associated with each directory is a 36-bit directory group membership word. The directory group membership word has a 1 bit for every group that the directory belongs to. If, when attempting to access the file, the user does not have the SELF relationship to the file's directory, but does have group access to a group which the file's directory belongs to, then the GROUP protection field is used to validate access.
3) If neither 1) nor 2) hold, then the OTHER protection field is used to validate access.

b. Directory Protection

In addition to the file protection, there is a facility for directory protection within TENEX. The directory protection mechanism, like the group mechanism, is administratively controlled and is inaccessible to the user. Associated with each directory is a protection word that is broken up into

fields analogous to those for files (SELF, GROUP, and OTHER). The first bit in this field is used to limit all accesses. If this bit is off, a user can't even know of the existence of the protected directory. If this first bit is on, then the other bits are used for governing OWNER, OPEN, and APPEND access. OWNER access allows connection to the directory without a password, control of file accounting, and some other control functions. OPEN access allows the user to open files in the directory according to their file protection. APPEND access allows the user to add files to the directory.

This protection scheme is flexible, but its particular form of flexibility also admits various incompletely defined and at times seemingly inconsistent situations. The directory protection and group mechanisms are poorly documented. Typical TENEX users have nothing to do with any of the protection mechanisms. A good deal of sophistication by users is required to change the file protection word. The group and directory protection mechanisms are often difficult to use even for experienced system users. The lack of user manipulatory facilities for the directory protection and group facilities seriously limit the utility of these facilities.

c. Process Protection

Processes within a TENEX job occupy a tree structure with the TENEX EXECutive language processor (EXEC), usually at the root. TENEX processes have always been able to protect themselves from inferior processes. (Note: The terms superior and inferior are used to indicate the relative position of processes in the tree. Process A is superior to process B if A is on the shortest path between B and the root.) With the implementations of the JSYS trapping facility, however, superior processes can now protect both themselves and the user's resources from inferior processes. This property has important applications to debugging, Trojan-horse problems, and extensive programming.

The cases of the debugging problem and the Trojan-horse problem are quite similar. In the first case, one is generally worried about protection from a well-meaning but possibly misdirected process and in the second case from an unknown and possibly malicious process, but the basic problem is the same. In each case, the user wants to protect himself from a process that might try to adversely affect his environment.

Early in the TENEX development an invisible debugger, IDDT, was developed which monitors execution of a user's process in a way invisible to the process. This debugger is quite useful for monitoring, but it cannot protect the user from arbitrary system calls by the process being debugged. The new JSYS trapping feature for TENEX allows a superior process to handle system calls for an inferior. This mechanism allows the superior to completely protect itself and the user's resources from the untrusted inferior process.

Though this theoretical ability to solve the classical Trojan-horse problem exists, it is awkward to implement and has not yet been programmed. The problem is interpreting the inferior's JSYS calls and deciding whether or not to allow them. Until a complete security system is implemented, however, a user's files can be protected with the simple mechanism outlined below.

The user could put whatever files the untrusted program is to be allowed to reference in a spare directory (this scheme requires that such a directory exists) and then, after connecting to the spare directory, he could run the untrusted program under a trivial monitor that simply traps any attempts to connect to another directory. This would crudely solve the classical Trojan-horse problem and allow users protection from undebugged programs.

The JSYS trapping facility is very new — new enough that at the time of this writing, no complete documentation exists. The JSYS trapping facility was designed, however, largely to aid implementation of the TENEX RSEXEC system, so the RSEXEC makes heavy use of JSYS trapping and is a good example of the ways in which extensive monitors can be written for TENEX.

The RSEXEC system is designed to give the user access to TENEX on a network virtual basis. To this end, it allows users to acquire directories on other host computers and to use the files in those directories as if they were local files. This requires that the RSEXEC trap JSYS calls and make references to remote files look like references to local files.

A set of capabilities are maintained with each TENEX process. These capabilities are used to denote special privileges of the process. Typical privileges are:

- o CTRLC allows a process to enable "control-C" as a PSeudo Interrupt (PSI, software interrupt) condition. "control-C" is usually reserved as the break character to escape from a process in TENEX. A process with the "control-C" capability can handle its own "control-C's".
- o SUPMAP allows a process to perform memory mapping operations on its superior.
- o WHEEL allows various privileged JSYSes. This capability is an all-powerful capability given to system programmers.

Processes in TENEX are also allowed flexible access control over the virtual memory pages of processes and files that they have access to. This is not particularly pertinent to security except to note that this mapping scheme correlates with the file protection mechanisms in the sense that virtual memory mapping allows at most those accesses allowed to the file or process being mapped.

One other TENEX feature which fits most naturally here is the way in which the TENEX executive language processor was essentially kept out of the TENEX security kernel. When a user initially connects to TENEX, he is given an EXEC. This EXEC is not logged in, however, and is only able to interpret the user's commands and perform simple operations. When the user tries to login, the EXEC simply performs a login JSYS supplying the name, password, and account given by the user. If this login is successful, the EXEC is given access to the user's login directory, the user's capabilities, etc. At this point the EXEC is acting like any other process running in the usual environment for the user. It can only access files accessible to the user and has only those privileges granted to the user by the monitor. In this way, the EXEC need only be trusted by the user with his resources to the extent that any other program which is running unprotected must be trusted. The EXEC can no more degrade the Monitor than can any other user process. Also, users cannot be affected by another user's EXEC unless there is a problem in the TENEX monitor. For this reason, except for possible Trojan-horse-like problems, the TENEX EXEC need not be considered part of the TENEX security kernel.

6.5 SUMMARY

Of the security problems that have been discovered in TENEX, the majority of them have been eliminated from more recent versions of the system. The more significant known problems that remain occur in the area of the peripheral interfaces. In TENEX it is possible, though difficult, to handle the classic Trojan-horse problem using the system call trapping feature (JSYS); TENEX does not address the general problem of controlled communication between mutually suspicious processes that is solved by experimental capability-list (C-list) systems.

Much of the TENEX implementation is relatively easy to analyze. Its major weaknesses from the point of view of code obscurity are in the file system, the Network Control Program, and the scheduler. Portions of the TENEX process and software interrupt handling code have been extensively analyzed and exercised. These sections of the monitor now appear to be implemented correctly.

6.6 OPERATING SYSTEM SECURITY FLAWS

During the more than 2 years of the TENEX study, 20 to 30 flaws were found in the system that could loosely be classed as security flaws. Of these, some 13 or 14 were analyzed in depth. These flaws were observed in TENEX releases between 1.29 and 1.32. Some of the existing flaws have been fixed in releases 1.33 or 1.34.

Here we present a few of the flaws that we have investigated. All of the known problems that still exist in TENEX are presented along with methods for minimizing their effects. Furthermore, since the other flaws have been fixed at all current TENEX installations, we have included representative examples of these earlier flaws. These examples are fairly typical of the range of security problems that occur in computer operating systems; however, the ease with which some of these problems were fixed is not typical. The flaws are categorized within the scheme presented in the taxonomy in section 3 in order to display the TENEX flaws in terms of global operating system problems. The associated taxonomy category is shown in parentheses.

a. Existing Flaws

- List-access failure during file recognition. (Inadequate identification/authorization/authentication)

There is a feature of the TENEX monitor which fills in the remainder of a recognizable name for a file at the user's request. The monitor code which handles this name recognition fails to check to see if the requestor of the recognition has list access. Users must be aware that this problem exists and not depend on list-access protection.

The needed fix is to add the appropriate access check. In the meantime, users who really have a need to hide the names of files in their directory can get someone to protect their directory using the directory-list protection.

- Crash on drum overflow. (Violatable prohibition/limit)

If users create too many private pages, the Monitor crashes. TENEX does not use any paging scheme for its secondary drum storage. If the drum fills up, the system crashes.

There are two schemes that have been used to alleviate this problem. One is to make the drum larger by logically including some disk storage. This scheme could work because there is an upper limit on how much drum storage can be demanded by users. This upper limit is very large, however, so the initial scheme of reserving disk space for the drum overflow is too impractical to be used to solve the problem completely. What is needed is a paging scheme for the "drum" storage.

Another scheme that has been used to prevent system crashes is the triggering of a machine-size-exceeded software interrupt if the drum is close to filling up. This does protect the system from crashing, but probably destroys a user program which was not doing anything wrong.

- Circular mapping causes hung process. (Exploitable logic error)

This problem appears if a user sets up a circular set of indirect map pointers in a process and then tries to kill the process. The killing process hangs irrevocably in the KFORK (kill fork) JSYS.

Having this problem occur simply degrades system performance and uses up the user's resources (the process hangs in an unterminating loop). There is currently no certain way known to get rid of such hung processes short of restarting the system.

This problem could be fixed by having the KFORK JSYS limit the number of indirect page pointers that it will follow in the same way that the pager does.

- Lack of scanner hang up and recall signal. (Hardware)

As mentioned earlier, some of the telephone scanners used with TENEX do not have adequate signals to notice when an auto-answer phone call is hung up and another call on the same line is initiated. This allows new callers to get old jobs if the old job was hung up without a proper logout.

To avoid this problem, the system should be configured with the proper scanner signals that are available. Users on systems without the proper hardware can minimize this problem by logging out properly whenever possible. This problem occurs only with telephone calls, so a system without telephone access (e.g., only hardwired and/or network access) does not have the problem.

b. Flaws That Have Been Fixed

- Skip return problem. (Exploitable logic error)

This error occurred because of the way the monitor implemented the mechanism which returns to the user's program counter (PC) + 1 rather than directly to the user's saved PC (called a skip return). The mechanism was simply to add 1 to the user's return word. If the PC happened to be -1, this addition would overflow into the index field of the word. In this case, because of the way in which the return instruction works, the return would be done to the location specified in general register 1. This return location also controls whether return is to user or monitor space. Under certain conditions, the user could control what was in register 1 and in some other registers. This control would allow the user to return to the registers that he had set up in the monitor's space. These registers could contain a program that bootstraps in a program designed by the user to take over the complete control of the TENEX monitor.

This was the only case encountered during the study of TENEX in which a complete take over of TENEX could be accomplished in this somewhat fundamental way. The password check problem noted below was as serious in its consequences, but was not as fundamental in nature.

The problem was patched the afternoon that the bug was encountered and was distributed to all TENEX sites through the ARPA network the same day. The fix amounted to masking out the indirect and index bits in the user return word before returning to the user.

- Password information leak. (Implied sharing of privileged/confidential data)

This problem combined several features of TENEX to produce a security threat in a clever way. The first feature is that a user can find out when a page in his virtual space which was unmapped has been referenced. The second feature is that the TENEX password-checking routine did character-at-a-time checking of a user's submitted password. The third feature was the fact that user processes can themselves submit passwords for checking in a flexible way.

What a user could do to exploit these features was to submit a candidate password in user space in such a way that the first character which was not known to be correct is 1 byte before a page boundary with an empty page to follow. Then if the character is correct, the password checker will reference the next page thereby telling the user that the password character was correct. As long as the character is incorrect, the user can simply change it and try it again. This reduces the number of guesses required to guess an N-character password from on the order of 64**N to on the order of 64*N. This allows passwords to be discovered in a reasonably short amount of time.

The fix to this problem was to have the password checker reference the full length of the submitted password regardless of where the check failed.

This flaw is mentioned because it is an instance of a very stubborn type of problem that is not directly attacked by any of the current program-verification or program-proving techniques. The problem is one of having the system leak sensitive information to a user process through timing or other unrelated information channels (in this case the fact that a page fault had occurred). With this problem, the password checker could have been proven correct as could the page mapper, but between the two of them they leak out some critical system information. There are many other ways in which a system can leak information from one process to another without directly transferring any data. However, because this general problem is not directly pertinent to TENEX, it is not discussed further.

- Incorrect communication instruction usage. (Exploitable logic error)

This is a case where inappropriate use was made of one of the special user-to-monitor space communication instructions. Here the instruction was intended to fetch a byte from the user space. It would do this properly, but would also do any indirecting that was requested by the user in the monitor space. When doing indirecting on a PDP-10, new memory words for the address calculation are accessed from addressed locations until an address is found in which the indirect bit is turned off. (For details, see DEC System 10 Assembly Language [7].) If an unmapped Monitor page was referenced during this address computation, TENEX would crash.

The fix was to use the proper communication instruction that also did its indirection in the user space. The actual fix which was initially implemented was to mask out the indirect bits of the user submitted pointer. This initial fix unnecessarily limited the user's flexibility.

The instruction set for the TENEX machine, even with these few user-monitor communication instructions, is so simple that bugs of this kind are rare.

- Accounting problem. (Inadequate identification/authorization/authentication)

This is a case where an add-on mechanism was not properly tied into the monitor's usual access control mechanisms. Originally in TENEX, user accounts were not validated. They were only used for system information. Later it was decided to charge according to the accounts. This required account validation. Initially this was implemented by having the EXEC check accounts if an account change was requested through the EXEC. This didn't suffice, however, because users could still change their accounts directly with the change account JSYS, thereby bypassing the EXEC's check.

In TENEX, version 1.32 changes were made to the change-account JSYS to have it validate accounts directly.

- TELNET typescript file problem. (Implied sharing of privileged/confidential data)

This is an example of where the Trojan-horse problem can still plague unwary TENEX users. The TENEX TELNET program is a program that can be used to connect to other sites on the ARPA network. This program has a feature for accumulating the user's terminal printout in a file, called his typescript file. This facility is turned on by default, but the file generated is made temporary so that is disappears if the user logs out without having explicitly saved it. Before this problem was noticed, the protection for the typescript file was set to the default protection of the directory that it was in. This was generally set to allow all users read access. This meant that after a user had completed a TELNET transaction and until he logged out, his typescript file, containing possibly sensitive information like passwords, etc., was up for grabs.

The fix was to set the protection of the typescript file to self only. The moral is that, even though TENEX has facilities for combating the Trojan horse problem, users must be wary of leakage when running any programs that are allowed to reference sensitive data.

- Privileged subsystem problem. (Inadequate identification/authorization/authentication)

This problem, which was alluded to earlier in the ARPA network discussion (Sec. 6.3.c), demonstrates the need to keep privileged processes to an absolute minimum (zero if possible).

TENEX has a facility called linking which allows two users to communicate by sending all terminal output generated by either user to both user terminals. When one user attempts to link to another, he is allowed to do so only if the other user is accepting links.

The RSEXEC system, which was mentioned earlier, implements as one of its services a mechanism for host-to-host network links. In the earlier stages of the RSEXEC development, it was found desirable to have the process at each site which listened for these network links to be a privileged process. The way it was implemented, the network link was put through in spite of the fact that the receiving party might be refusing links. This left potentially sensitive information exposed.

The point here is that any privileged processes running on TENEX are in the TENEX security kernel. If TENEX is ever to be proven correct then any such processes must be proven correct. The easiest way to do this is to have no such processes.

- Unchecked file deletion access. (Inadequate identification/authorization/authentication)

The DELNF JSYS is a monitor call that deletes all but some number of versions of a file. Such a JSYS was initially thought convenient, but in actual practice is almost never used. It was probably the fact that the DELNF JSYS is so seldom used that caused this integrity problem to go undetected for so long.

In coding DELNF, the file protection check was simply omitted. This allowed any user to delete any file in any directory with simply open and list access to the directory. This includes almost all files.

The important thing to note about this problem is the fact that there could be an access check omitted in DELNF even though the proper check was made in DELF (the JSYS usually used for deleting files). This dual implementation situation is caused by ineffective organization and causes comparable problems in many systems. TENEX generally has very few problems of this sort.

There is a feature of TENEX that effected the DELNF problem that can be most appropriately discussed here. When TENEX deletes a file as with DELNF, it simply sets a "deleted" bit in the file descriptor. Such deleted files can afterward be undeleted if desired. These deleted files are only really destroyed if the user explicitly asks for them to be expunged or if he logs out. If, before deleting and expunging any files or logging out, the user had first taken the precaution of undeleting all files, he would have been safe from all but the most insistently malicious DELNFing programs.

There are system files that would cause a system crash or serious degradation if deleted (for example the TENEX EXECutive), so this wasn't really an acceptable solution, but with it users could at least protect their own files fairly well.

7. Summary and Conclusions

The protection of computer resources, data of value, and individual privacy has motivated a concern for security of EDP installations. Because operating systems are such an integral and critical part of large installations, this concern for security has extended to operating systems. This document reports some of the security flaws and security enhancements developed from a research project.

This material is a starting reference for planning a security investigation of an EDP installation's operating system. Its intended use is at Federal Government EDP installations with large commercial systems. However, the results can be applied to most EDP installations.

To meet this objective, three commercial operating systems are analyzed and security enhancements suggested. The specific security flaws discussed are formally classified according to the taxonomy developed here. Flaws can be classified formally because of the similarity of operating systems and their security problems. This classification leads to a clearer understanding of security problems and aids in analyzing new systems.

The operating systems chosen for analysis represent a major segment of the current Federal EDP installations but not the entire spectrum. The choice of systems implies neither evaluation nor recommendation but only prevalence of use and interest. Also, no attempt at completeness of flaw description is presented, only a representative set of flaws for which some amount of security enhancement can be done.

Several conclusions are drawn from the overall analysis:

- Security is not an absolute but is relative for each installation, depending on the data, resources, and mission of the facility as well as on the potential hazards. Each installation must determine its own cost-risk trade-off.
- Operating system security is not a binary, yes-no, condition. Primarily because of its size and complexity, no large operating system can be said to be completely secure. However, many security enhancements can be incorporated into systems to make them harder and costlier to penetrate or compromise.
- Software security, which includes operating system and user applications programs, is only one aspect of the total security of an EDP installation. Administrative, physical, and hardware security need to be considered and kept in perspective.
- Operating systems are not necessarily the most critical point of security or the point that requires the first enhancements. Also, they may not offer the highest return in terms of the cost-protection trade-off as compared to physical or administrative security.
- There are a limited number of basic security flaws (5 to 15 depending on the way they are described). There can be numerous different exploitations, but the number of basic flaws remains fairly constant. These basic flaws tend to re-occur in different systems because of the similar architecture and design of the machines and operating systems.
- With the expected technical growth of EDP installations (e.g., in the area of networks), the security of complex systems will require continuing analysis. Current security flaws and enhancements may not remain valid with the technological changes that are taking place.
- Today's commercial operating systems were not designed with security as a critical design factor. Efficiency, flexibility, and cost were more important, not the demand for security. This orientation led to security via retrofitting, and this is shown to be a poor method as changes often introduce new and subtle security problems.

- There are some enhancements that can be instituted by installations to increase operating system security and increase the difficulty of a penetration. But these are often in the area of software modification and thus require a system expertise that is not always readily available at installations. However, some operating system security problems can only be "fixed" through a redesign of the entire system.
- Continuing research and development in operating systems security are being performed in universities, research institutions, commercial firms, and government agencies. An informative summary of sites and security work being done has been published [8]. The list of sites includes: MIT, Carnegie-Mellon, Lawrence Livermore Laboratory, Information Sciences Institute, MITRE, TRW, Systems Development Corporation, IBM, Honeywell, National Bureau of Standards, and the Air Force Electronic Systems Division.
- If security modifications are to be made to operating system code, it is very desirable to have "clean" coding. (That is, coding written in a straightforward style, unencumbered with tricks or complex instructions.) Easily readable coding does not add to security per se, but does aid in understanding the code, and reduces the chance that a routine can be used in a manner other than what was originally intended.

Glossary

This glossary defines terms that may not be defined in the text or that require special emphasis and ease of referral. The items in brackets [ ] specify the context of the terms.

Many of the definitions have been modified to make them consistent with the definitions that are to appear in the Glossary of Terminology for Computer Systems Security [9].

Access

The ability and the means to communicate with (input to or receive output from), approach or make use of. Data access is often categorized by combinations of read, write, or execute.

Asynchronous [event/process]

Events or processes which occur at an unknown time or execute at an unknown rate with respect to each other. An example is an I/O process which proceeds at a rate independent of the program which initiated it.

Audit trail

A chronological record of system activities which is sufficient to enable the reconstruction, review, and examination of the sequence of environments and activities surrounding or leading to each event in the path of a transaction from its inception to output of final results.

Authentication

The act of verifying the eligibility (i.e., authorization) of a user and his agents (e.g., programs, terminals) to access specific categories of information.

Authorization

The granting to a user, a program, or a process the right to access.

Capability [process/user]

The right to access granted to an individual, program, or process. In a capability system, this right to access is signified by a protected bit pattern or by inclusion on an access list.

Certification [operating system]

Proving or measuring the integrity of a system. Certification is the act of authoritatively confirming (via an effective methodology) that the protection capabilities or characteristics of a system comply with a particular set of requirements.

An assurance, based on defined objectives and arrived at through a closed process of assessment, that the probability of operating system design and/or implementation flaws is less than a specified value, and that the probability of a hardware failure is less than a specified value.

Confidentiality

A concept that applies to data that must be held in confidence. Confidentiality describes the status accorded to data and the degree of protection that must be provided for such data. The protection of data confidentiality is one of the objects of security. Data confidentiality applies not only to data about individuals but to any proprietary or sensitive data that must be treated in confidence.

Controlled access

The concept that each authorized user of a system be permitted access to that information and resources to which he is authorized, but to no more.

Limiting access to the resources of an automated data processing system to only authorized users, programs, and processes or (in computer networks) other authorized data processing systems.

Control program

That part of an operating system which directly interfaces with the hardware and which initiates and guides the execution of all other programs and processes. A control program frequently consists of an interrupt handler and a housekeeper component. Other terms used synonymously include: supervisor, monitor, and executive.

Control state

One of two generally possible states in which a computer system may operate; the other is the user state. In the control state, certain privileged instructions are permitted execution. Privileged instructions are not permitted to execute when the system is operating in the user state. Other terms used synonymously include: supervisor state, monitor mode, and executive state. (The architecture of some computer systems supports operation under fewer or more than two hardware states.)

Cost-risk analysis

The assessment of the cost of providing a given degree of protection vs the potential risk of not protecting a resource. (This is a function of the economic consequences of a loss and the threat probability.)

Encrypt [data]

The coding of information to conceal its meaning; to convert plain text into an unintelligible form by means of a cryptosystem.

Flaw [operating system]

An operating system integrity flaw is the state that exists whenever a user (or his programs) has the potential to cause the system to cease reliable and secure operation. An integrity failure exists when this potential has been exercised and the reliable and secure operation of the system is breached.

A reliability flaw represents the potential to cause a system to cease correct operation (e.g., to crash or degrade the operation of a system); a security flaw represents the potential for one user to access (i.e., read, modify, manipulate, or destroy) another user's information or programs against that user's wishes or to gain control of the operating system.

Integrity

Integrity is the state that exists when there is complete assurance that under all conditions a system works as intended. That is, the system reflects the logical correctness and reliability of the operating system; the logical completeness of the hardware and software that implement the protection mechanisms; and the consistency of the data structures and accuracy of the stored data. Integrity is concerned with reliability (fraud and error) problems and with security (resource and privacy protection) problems.

Interactive [computer system]

Use of a computer such that the user is in intimate control of the execution of his work and may make modifications or enter data between execution steps.

Isolation [user]

The containment of users, data, and resources in an operating system such that users may not access each other's data and resources and may not manipulate the protection controls of the operating system.

Multiprogrammed [computer system]

A system which executes numerous processes or programs concurrently, by overlapping or interleaving their execution. For example, permitting more than one process to timeshare computer peripheral devices.

Operating system

The aggregate of control and maintenance software that is used to support user interface functions.

Paging

A procedure for moving standard-size blocks of information (pages) between main storage and auxiliary storage units. This is generally used to permit several programs to share main storage concurrently.

Preventive mechanism

A preventive mechanism is a software or hardware mechanism that implements all or part of the elements of protection in a system. In an operating system, these elements are: isolation and controlled access.

Privacy [information]

The right of an individual to self-determination as to the degree to which he will interact with his social environment; this is manifested by an individual's willingness to share information about himself with others; and may be compromised by unauthorized exchange of information about the individual between other parties.

Process [computer]

A process (or task) is a computation that may be executed concurrently with other computations. This term may represent either a sequence of instructions or an entire (independently dispatchable) work unit. (An operating system generally represents the latter by a control block containing an address-state and a processor-state description. A process may involve several programs and several jobs. A user may create more than one process. Two or more processes may share programs and data.)

Protection

The defending or guarding of the hardware, software, and information resources of a computer system.

Real-time process

See time-dependent.

Reliability

A measure of the ability to function without failure.

Remote-access [computer system]

A hardware and software system which permits input to be made from a location other than the central computer room. Usually, such a system provides remote output as well; operates over telecommunications circuits; and manages a number of remote-access stations or terminals.

Scavenging [data]

Searching through data not erased from storage after use (i.e., residue), without necessarily knowing its format or content, in an attempt to locate or acquire unauthorized information.

Security

Security is the realization of protection of data, the mechanisms and resources used in processing data, and the security mechanism(s) themselves. Data security is the protection of data against accidental or unauthorized destruction, modification, or disclosure using both physical security measures and controlled access techniques. Physical security is the protection of all computer facilities against all physical threats (e.g., damage or loss from accident, theft, malicious action, fire, and other environmental hazards). Physical security techniques involve the use of locks, badges (for personnel identification), guards, personnel security clearances, and administrative measures to control the ability and means to approach, communicate with, or otherwise make use of, any material or component of a data processing system.

Security kernel [operating system]

That portion of an operating system whose operation must be correct in order to ensure the security of the operating system. Ideally, this involves the isolation of all hardware and software functions, features, and data which form the basis of protection of programs and information in one protected, centralized part of the system (i.e., kernel). The rest of the operating system is linked to this kernel in a manner such that the kernel is invoked by all references to information in the system. Only the protection mechanisms themselves are placed in the kernel, and the policy-making code which directs these mechanisms is placed elsewhere in protected compartments.

Spoof

To deliberately induce a system user or operator to take an incorrect action.

Suspicious processes

In a multiprogramming environment, the concurrent use of the system for sensitive data or programs by users who mutually distrust one another or where one distrusts the other. Such processes normally grant only the minimal number of required capabilities to each other.

Symbiont

Small routines that buffer large amounts of data from low-speed peripherals to main storage to minimize the number of data transfers.

System call

An instruction that acts much like a subroutine call but transfers control to the operating system rather than one of the user's subroutines.

Taxonomy

A system of classification which proceeds from the most inclusive classification to the most explicit.

Time-dependent [sequence]

A sequence of computer instructions which can produce different results depending upon the rate at which it is executed. A real-time process involves time-dependent instruction sequences, which must be executed within specific (usually small) periods of elapsed time for the process to produce valid results.

Timing window [execution]

Any unit of time (generally at least the length of time it takes a computer to execute a single instruction) between two sequential processes, where the second process is dependent upon information supplied by the first process.

Trap door

A breach intentionally created in a computer system for the purpose of collecting, altering, or destroying data.

Trojan horse [flaw]

A trusted program which contains a trap door.

Validate

To perform tests and evaluations to determine compliance with security specifications and requirements. Validation is usually considered to be less comprehensive and rigorous than certification.

Virtual storage

An organization of memory (based on automatic swapping of data and programs as required between operating memory and secondary memories) that allows addressing as if a very large executable memory existed, although the executable memory available to the processor is generally much smaller than the range of addressing which may be referenced by a program.

Working set

The area of a processes's virtual storage that is referenced frequently. Generally, this is taken to be the portion that is referenced enough to require presence in primary memory.

Bibliography

AUDITING

Krauss, L. I., SAFE: Security Audit and Field Evaluation for Computer Facilities and Information Systems. (Firebrand, Krauss and Co., East Brunswick, N. J., 1972).

An auditing handbook that has several hundred checkpoints of security related items for computer facilities. The book is designed to aid in a do-it-yourself field investigation of security measures and safeguards. The user estimates the importance of each item and his facilities compliance, and then calculates a security index value (a weighted average). The eight classifications of checkpoints are: personnel; physical; data, programs, and documentation; operational; backup; development; insurance; and security program.

Computer Control Guidelines, Canadian Institute of Chartered Accountants, (Auerbach Publishers, Princeton, N. J., 1971).

A book presenting the results of a study performed by the Canadian Institute of Chartered Accountants. The book deals with 25 control objectives, giving the minimum control standards and specific control techniques.

Kuong, J. F., Computer Security, Auditing and Controls. -- A Bibliography, Management Advisory Publications, Wellesley Hills, Mass., 1973.

This non-annotated bibliography covers several hundred publications in the interrelated areas of auditing, controls, and security. The bibliography is classified into six main subheadings and numerous subclassifications. The main subheadings are: EDP Auditing and Controls, computer security and privacy, EDP planning and operations control, EDP management review and evaluation, online and real-time systems, and checklists and guidelines.

BIBLIOGRAPHY

Abbott, R. P., et al., A Bibliography on Computer Operating System Security, Lawrence Livermore Laboratory, Rept. UCRL-5155 (1974).

This bibliography has over 750 entries on the subject of computer security. The emphasis is on software and, in particular, operating system security. The first part of the bibliography is a key word out of context (KWOC) index and the second part a master listing of each entry.

GENERAL SECURITY

AFIPS System Review Manual on Security, American Federation of Information Processing Societies, Inc., Montvale, N. J., 1974.

This manual is the first AFIPS System Review Manual. It is intended to be used as a guide for reviewing existing systems and as a checklist during system development. The intended audience includes managers, EDP auditors, and systems designers. The chapter topics include: personnel, physical security, operating systems, access controls, programs, communications, storage, and input/output. Each chapter contains "General Principles" to be followed and a checklist of related questions.

Van Tassel, D., Computer Security Management (Prentice-Hall, Inc., Englewood Cliffs, N. J., April 1972.)

This book is a good examination of numerous computer security topics. These topics include: past crimes and disasters, company security, embezzlement, EDP controls, auditability, program security, cryptographic techniques, disaster protection, insurance, service bureau relations, and time-sharing security. The book deals with management controls and operating procedures and has a series of checklist questions.

Data Security and Data Processing, Vols. 1 to 6 (G320-1370 to G320-1376). (International Business Machine Corp., White Plains, N. Y., 1974.)

These six volumes report the findings of a program initiated in 1972 by IBM to strengthen data security research and technology, and to identify user requirements. A wide range of specific topics are covered and presented at different levels of detail. Volume 1 is written for management and discusses data security in general. Volume 2 summarizes the findings of the study. Volumes 3 through 6 present in detail the findings of the four sites: the Massachusetts Institute of Technology, the State of Illinois, TRW Systems, Inc., and the IBM Federal Systems Center at Gaithersburg, Maryland.

OPERATING SYSTEM SECURITY

Anderson, J. P., "Information Security in a Multi-User Computer Environment," Advances in Computers, (Morris Robinoff editor, Academic Press, Inc., New York, 1972), pp. 1-35.

This short article deals with methods of exploiting flaws or weaknesses in operating systems of multi-user systems to illegally access data. Some hardware and software enhancements are suggested; however, the article is quite technical.

Conway, R. W., W. L. Maxwell, and H. L. Morgan, "On the Implementation of Security Measures in Information Systems," Communications of the ACM, April 1972, pp. 211-220.

This paper discusses the nature of flexibility in a secure system and to relate the costs of implementation and enforcement to that flexibility. A security matrix model is presented and used to explain security features of several existing systems.

PHYSICAL SECURITY

Guidelines for Automatic Data Processing Physical Security and Risk Management, National Bureau of Standards, Federal Information Processing Standards Publication, FIPS PUB 31, June 1974.

This document is part of the Federal Information Processing Standards Publication Series. The publication provides guidelines to be used by organizations in structuring physical security programs. It includes the following topics: security analysis, natural disasters, supporting utilities, system reliability, procedural measures and controls, off-site facilities, contingency plans, security awareness, and security audit. It also contains statistics and information relevant to physical security and gives references to other, more detailed, publications.

Molho, L. M., "Hardware Aspects of Secure Computing," AFIPS Conference Proceedings, Spring Joint Computer Conference, Vol. 36, 1970, pp. 135-141.

This paper reports the findings of a study of the hardware aspects of controlled access time-shared computing. It deals with the storage protection system and the Problem/Supervisor state control system of an IBM System 360 Model 50 computer. Methods of enhancing security are discussed.

PRIVACY – CONFIDENTIALITY

Renninger, C. R. and D. K. Branstad, Ed., Government Looks at Privacy and Security in Computer Systems. National Bureau of Standards Technical Note 809, National Bureau of Standards, Washington, D.C., 1974.

This publication summarizes the proceedings of a conference held for the purpose of highlighting the needs and problems of Federal, State, and local governments in safeguarding individual privacy and protecting confidential data contained in computer systems from loss or misuse. The conference was sponsored by the National Bureau of Standards in November 1973. Major needs and the cost implications of providing security measures are discussed.

Renninger, C. R., Ed., Approaches to Privacy and Security in Computer Systems. National Bureau of Standards Special Publication 404, National Bureau of Standards, Washington, D.C., 1974.

This publication summarizes the proceedings of a second National Bureau of Standards conference held in March 1974 to continue the dialog in search of ways to protect confidential information in computer systems. Proposals are presented for meeting governmental needs in safeguarding individual privacy and data confidentiality as identified in the prior NBS conference.

Westin, A. F. and M. A. Baker, Databanks In a Free Society (Quadrangle Books, New York, 1972).

This book reports the results of a 3-year study of computer databanks and civil liberties. The study was commissioned by the National Academy of Sciences. The book includes: 1) a profile of 14 organizations that use computers for record keeping, 2) a description of changes both in organizational record-keeping patterns and in civil-liberties protections that were believed to be taking place but were not observed, 3) a description of those changes that were observed taking place, and 4) the findings about accuracy and security problems in computerized record systems.

References

[1] Adams, D. L. and Mullarky, J. F., "A Survey of Audit Software," The Journal of Accountancy, 39-66 (September 1972).

[2] The Federal Register, Monday, March 17, 1975, Washington, D.C., Vol. 40, No. 52, pp. 12134-12139.

[3] Konigsford, W. L., A Taxonomy of Integrity Problems, Lawrence Livermore Laboratory Rept. (To be published).

[4] IBM SHARE GUIDE, 1620 Users Group, Appendix F, Report of Systems Objections and Requirements Committee, Rept. No. SSD 123 (June 1964).

[5] Data Security and Data Processing, Vols. 1-6, IBM, White Plains, NY, Rept. Nos. G320-1370 through -1376 (June 1974).

[6] McPhee, W. S., "Operating System Integrity in OS/VS2," The IBM Systems Journal, No. 3, IBM, Armonk, NY (1975).

[7] decsystem10 assembly language handbook, Software Distribution Center, DEC, Maynard, MA 01754, Rept. No. DEC-10NRZC-D (1973).

[8] Saltzer, J., "Ongoing Research and Development on Information Protection," Operating System Review, 8-24 (July 1974).

[9] Glossary of Terminology for Computer Systems Security, Federal Information Processing Standards, Task Group 15: Computer Systems Security, National Bureau of Standards, September 2, 1975.

RAC/lt/la

NBS-114A (REV. 7-73)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET	1. PUBLICATION OR REPORT NO. NBSIR-76-1041	2. Gov't Accession No.	3. Recipient's Accession No.

4. TITLE AND SUBTITLE

Security Analysis & Enhancements of Computer Operating Systems

5. Publication Date April 1976

6. Performing Organization Code

7. AUTHOR(S) The RISOS Project
Lawrence Livermore Laboratory

8. Performing Organ. Report No.

9. PERFORMING ORGANIZATION NAME AND ADDRESS

The RISOS Project
Lawrence Livermore Laboratory
Livermore, California 94550

10. Project/Task/Work Unit No. 640.1112

11. Contract/Grant No.

12. Sponsoring Organization Name and Complete Address *(Street, City, State, ZIP)*

National Bureau of Standards
Department of Commerce
Washington, D.C. 20234

13. Type of Report & Period Covered Final 7/74-12/75

14. Sponsoring Agency Code

15. SUPPLEMENTARY NOTES

16. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here.)*

The protection of computer resources, data of value, and individual privacy has motivated a concern for security of EDP installations, especially of the operating systems. In this report, three commercial operating systems are analyzed and security enhancements suggested. Because of the similarity of operating systems and their security problems, specific security flaws are formally classified according to a taxonomy developed here. This classification leads to a clearer understanding of security flaws and aids in analyzing new systems. The discussions of security flaws and the security enhancements offer a starting reference for planning a security investigation of an EDP installation's operating system.

17. KEY WORDS *(six to twelve entries; alphabetical order; capitalize only the first letter of the first key word unless a proper name; separated by semicolons)*

BBN-TENEX; IBM OS/360; operating system security; security flaws; software security; taxonomy of integrity flaws; UNIVAC 1100 Series OS

18. AVAILABILITY

- [x] Unlimited
- [ ] For Official Distribution. Do Not Release to NTIS
- [ ] Order From Sup. of Doc., U.S. Government Printing Office Washington, D.C. 20402, SD Cat. No. C13
- [x] Order From National Technical Information Service (NTIS) Springfield, Virginia 22151 Order No. PB257087

19. SECURITY CLASS (THIS REPORT) UNCLASSIFIED

21. NO. OF PAGES 69

20. SECURITY CLASS (THIS PAGE) UNCLASSIFIED

22. Price $4.50

USCOMM-DC 29042-P74