NATIONAL COMPUTER SECURITY CENTER

AD-A234 055
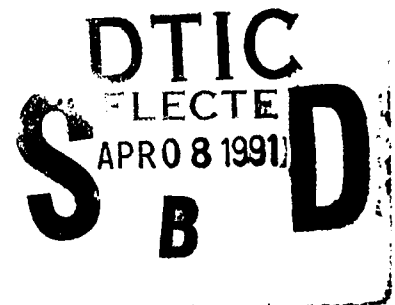
FINAL EVALUATION REPORT
OF
AMERICAN TELEPHONE AND TELEGRAPH

SYSTEM V/MLS RELEASE 1.1.2 RUNNING ON UNIX SYSTEM V RELEASE 3.1.1

DTIC
ELECTE
APR 0 8 1991
S
B
D

October 18, 1989

Approved for Public Release:
Distribution Unlimited

FINAL EVALUATION REPORT

AMERICAN TELEPHONE AND TELEGRAPH

SYSTEM V/MLS RELEASE 1.1.2

RUNNING ON UNIX SYSTEM V RELEASE 3.1.1

NATIONAL COMPUTER SECURITY CENTER

9800 Savage Road
Fort George G. Meade
Maryland 20755-6000

October 18, 1989

This page intentionally left blank.

FOREWORD

This publication, the Final Evaluation Report of American Telephone and Telegraph, System V/MLS, is being issued by the National Computer Security Center under the authority of and in accordance with DoD Directive 5215.1, "Computer Security Evaluation Center". The purpose of this report is to document the results of the formal evaluation of AT&T's System V/MLS operating system. The requirements stated in this report are taken from *DEPARTMENT OF DEFENSE TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA* dated December 1985.

Approved:

Eliot Sohmer _____ October 18, 1989

Eliot Sohmer
Chief, Office of Product Evaluations and Technical Guidelines
National Computer Security Center

Accession For	
NTIS GRA&I	☑
DTIC TAB	☐
Unannounced	☐
Justification	

By_____
Distribution/

Availability Codes	
Dist	Avail and/or Special
A-1	

ACKNOWLEDGEMENTS

Team Members

Team members included the following individuals, affiliated with the following organizations:

Karen M. Bielat
Caralyn A. Crescenzi
Mark D. Gabriele

National Computer Security Center
Fort Meade, Maryland

William N. Havener

The MITRE Corporation
McLean, Virginia

Sharon G. Kass

The MITRE Corporation
Bedford, Massachusetts

Holly M. Traxler

Institute for Defense Analysis
Alexandria, Virginia

Acknowledgement is given to Rick A. Siebenaler for his contributions to this document.

Table of Contents

EXECUTIVE SUMMARY

The security protection provided by the AT&T System V/MLS (System V/MLS) operating system, configured as described in the Trusted Facility Manual, when running on the AT&T 3B2/500 or 3B2/600 mini computers, has been evaluated by the National Computer Security Center (NCSC). The security features of System V/MLS were examined against the requirements specified by the *DoD Trusted Computer System Evaluation Criteria* (the Criteria or TCSEC) dated 26 December 1985 in order to establish a TCSEC rating.

The NCSC evaluation team has determined that the highest class at which System V/MLS satisfies all the specified requirements of the Criteria is class B1. Therefore System V/MLS, when configured in the manner described in the Trusted Facility Manual, has been assigned a class B1 rating.

A system that has been rated as being a B1 class system provides a Trusted Computing Base (TCB) that preserves the integrity of sensitivity labels and uses them to enforce a set of mandatory access control rules.

The UNIX[1] System V operating system is a general purpose time-sharing system. System V/MLS, an enhanced version of UNIX System V, was developed to meet the B1 Criteria while maintaining compatibility with UNIX System V. System V/MLS maintains a security audit trail, provides mandatory access control, and includes other security features such as a random password generator, a trusted version of the Bourne shell, and a trusted administrative interface. There is also a configuration management plan in effect for the this system for future Ratings Maintenance Phase (RAMP) participation.

---

[1]   UNIX is a registered trademark of AT&T

This page intentionally left blank.

INTRODUCTION

In October 1988, the National Computer Security Center (NCSC) began a formal product evaluation of System V/MLS, an American Telephone and Telegraph (AT&T) product. It is intended that this report give evidence and analysis of the security features and assurances provided by the System V/MLS operating system. This report documents the evaluation team's understanding of the system's security design and appraises its functionality and assurances against the Criteria's B1 security requirements. This evaluation applies to System V/MLS Release 1.1.2 running on UNIX System V Release 3.1.1.

The System V/MLS product adds mandatory access control and auditing capabilities to UNIX System V. It is integrated with UNIX System V at install time via an overlay procedure. Some UNIX System V routines were rewritten for System V/MLS, and others were added. Hooks were inserted in the underlying UNIX System V routines to call the new System V/MLS routines.

Although System V/MLS is a separate product from UNIX System V, throughout this report System V/MLS will be referred to as the whole integrated system. Statements referencing only those routines unique to the System V/MLS product are clarified to reflect the exact meaning.

Material for this report was gathered by the NCSC B1 evaluation team through documentation, interaction with system developers, and hands on use of System V/MLS.

Evaluation Process Overview

The National Computer Security Center (NCSC) was created to improve the state of computer security in computer systems processing information that is vital to the owners of that information. The Center fulfills its mission by promoting the development and implementation of Trust Technology and encouraging the widespread availability and use of trusted computer security prooducts. Through the Trusted Product Evaluation Program, the Center works with the manufacturers of hardware and software products to implement and make available to the public technically sound computer security solutions. Under this program, the NCSC evaluates the technical protection capabilities of computer security products against well-defined published evaluation criteria.

The product evaluation process culminates in the publication of a Final Evaluation Report, of which this document is an example. The Final Evaluation Report describes the product and assigns it a rating that denotes a specific level of trust. The assigned rating is independant of any consideration of overall system performance, potential applications, or particular processing environment. Rated products are listed on the Evaluated Products List (EPL), the aim of which is to provide ADP system

developers, mangers, and users an authoritative evaluation of a product's suitability for use in processing important information.

The NCSC performs evaluations of computer products in varying stages of development from initial design to those that are commercially available. The following is a description of the process by which this system was evaluated. For a description of the current evaluation process, see the Vendor Guide.[1]

This product evaluation consisted of a developmental phase and a formal phase. The primary thrust of the developmental phase is an in-depth examination of a manufacturer's design for either a new trusted product or for security enhancements to an existing product. Since the developmental phase is based on design documentation and information supplied by the industry source, it involves no "hands on" use of the system. The developmental phase results in the production of an Initial Product Assessment Report (IPAR). The IPAR documents the evaluation team's understanding of the system based on the information presented by the vendor. Because the IPAR contains proprietary information, distribution is restricted to the vendor and the NCSC.

Products entering the formal phase must be complete security systems. In addition, the release being evaluated must not undergo any additional development. The formal phase is an analysis of the hardware and software components of a system, all system documentation, and a mapping of the security features and assurances to the Criteria. The analysis performed during the formal phase requires "hands on " testing (i.e., functional testing and, if applicable, penetration testing). The formal phase results in the production of a final report and an Evaluated Products List (EPL) entry. The final report is a summary of the evaluation and includes the EPL rating which indicates the final class at which the product successfully met all Criteria requirements in terms of both features and assurances. The final report and EPL entry are made public.

Document Organization

This report consists of four major sections and four appendices. Section 1 is an introduction. Section 2 provides an overview of the system hardware and software architecture. Section 3 provides a mapping between the requirements specified in the Criteria and the System V/MLS features that fulfill those requirements. Section 4 presents the evaluation team's personal opinions about System V/MLS. The appendices identify specific hardware and software components to which the evaluation applies, list the trusted processes in the system, and provide a bibliography for this report.

---

1   *Trusted Product Evaluations, A Guide For Vendors*

SYSTEM OVERVIEW

This section begins with a brief description of the history of the System V/MLS system, and then describes the security-relevant architecture and mechanisms.

System V/MLS Background and History

In 1969 Ken Thompson, Dennis Ritchie and other employees at Bell Laboratories developed the UNIX operating system, a time sharing system implemented on a Digital Equipment Corporation model PDP-7 computer.

In 1973 UNIX was rewritten in the high level language C from the original assembler language. This made the system more readily understandable and portable to different hardware bases.

The portability of the system led to its use on numerous hardware architectures. By 1977 the number of sites using the UNIX operating system had grown to approximately 500. During the next five years Bell Laboratories enhanced the system and eventually released the UNIX System III version. The UNIX System IV version was internal to AT&T and not released commercially. By 1983 Bell Laboratories added several features and announced the System V version of UNIX.

In 1984-1985 the AT&T Federal Systems Division, which had been working towards the development of a secure UNIX operating system, developed "SecPac", a UNIX operating system add-on security package. SecPac provided rudimentary data sensitivity labeling, mandatory protection, and a detailed audit trail. SecPac was then modified and refined into the current System V/MLS product.

Hardware Architecture

The AT&T System V/MLS Trusted Computing Base (TCB) consists of two primary elements: the system hardware, and the system software. It is the purpose of the hardware to provide an environment which the software can use to implement a complete and trusted interface to the system. The following section will describe the hardware and how it provides the necessary support for the system software.

Evaluated Configurations

The evaluated configurations of System V/MLS include two hardware models, the AT&T 3B2/500 and 3B2/600, with support for the AT&T 630 Multi-Tasking Graphics (MTG) terminal, the AT&T 4425 terminal, the AT&T 605 BCT terminal, and the AT&T 5310 printer. Both of the host machine models utilize the WE 32100 Microprocessor and the WE 32101 Memory Management Unit.

The AT&T 3B2/500 is configured with 4M bytes of RAM (8M bytes maximum), 147M bytes of hard disk storage which is accessed through a Small Computer System Interface (SCSI) card, floppy disk drive, 60M byte cartridge tape drive (SCSI interface), and support for a virtual cache option.

The AT&T 3B2/600 is nearly identical in architecture to the 3B2/500, with slightly different standard components. The 3B2/600 is configured with 4M bytes of RAM (16M bytes maximum), 294M bytes of hard disk storage (SCSI interface), floppy disk drive, 60M byte cartridge tape drive (SCSI interface), and a virtual cache card. The 3B2/600 also supports a multiprocessor enhancement, which is not included in the evaluated configuration.

Both of these system models provide 32-bit data and address paths, as well as Error Correcting Code (ECC) RAM to detect and correct single bit errors and detect double bit errors. Both provide support for the WE 32106 Math Acceleration Unit.

The evaluated configuration includes three types of terminals: two ordinary terminals and an intelligent terminal, the AT&T 630 Multi-Tasking Graphics terminal. The 630 MTG incorporates 640K bytes of RAM (1M byte maximum), a Motorola 68000 microprocessor and a bitmapped display with a 1,024 x 1,024 resolution. The terminal allows a user to create multiple windows operating concurrently at different security levels, allows window-to-window operations, and provides a trusted communications path.

Hardware Components

Central Processing Unit Subsystem

System V/MLS utilizes the Western Electric 32100 Microprocessor (WE 32100 or CPU) to provide computational and security functionality. The WE 32100 is a 32-bit central processing unit which supports a 4 Gigabyte address space, via 32-bit data and address buses. The WE 32100 has a 64-word on-chip instruction cache. All memory addresses for instruction fetch and data go through the Western Electric 32101 Memory Management Unit (WE 32101 or MMU) in System V/MLS to provide a virtual memory environment.

Sixteen 32-bit registers are available on the WE 32100, consisting of thirteen general-access registers, and three kernel-privileged (for write access) registers. The kernel-privileged registers were designed specifically to support the concept of a process within the CPU. This design provides a convenient and efficient mechanism, upon which UNIX was built.

Processor Execution Modes

The WE 32100 supports four execution modes: kernel, executive, supervisor, and user. Of these, only kernel and user mode are used. The design of the 3B2 system hardware ensures that neither of the other two modes is usable.

The execution modes serve two purposes; first, they dictate which instructions are available for execution, and second, they are used by the MMU to enforce restrictions upon access to data. Kernel mode is the most privileged; in this mode all instructions are available for execution. User mode is the least privileged; all user programs are executed in user mode. Only the execution of a system call transfers the processor from user to kernel mode.

Execution Mode Switching

The system call (gate) mechanism provides a means of controlled changes of the processor execution mode by installing new Processor Status Word (PSW) and Program Counter (PC) register values. If the new PSW has a different processor execution mode than the current PSW, a transition to the new mode occurs.

The GATE and RETG instructions are used to switch processor execution modes (see "Instruction Set" below). GATE performs the actual PSW switch, from which the processor deduces its current execution mode by examining the PSW value. RETG is used to return from the environment which was "GATEd" to, and in doing so, RETG forces the new execution mode to be less privileged than or equal to the current mode.

Instruction Set

The CPU implements 179 instructions, of which 9 are provided to directly support operating system functions. Those opcodes for which there is no corresponding instruction generate an illegal opcode exception when executed.

The instructions designated for supporting an operating system are those that can change the physical state of the processor, respond to interrupts, or permit changing the process that is currently executing on the CPU. Of the nine operating system instructions, six require the processor to be in kernel mode for execution. The kernel mode (privileged) instructions are: CALLPS, DISVJMP, ENBVJMP, INTACK, RETPS, and WAIT. If these instructions are executed in other than kernel mode, a privileged opcode exception occurs. The remaining three operating system instructions are: GATE, MOVTRW, and RETG, which may be executed while the CPU is in any processing mode.

The following is a discussion of the operating system instructions:

Call Process: CALLPS performs a process switch, saving the current process image and entering a new process. CALLPS saves the context (register contents) of the current process, pushes the current Process Control Block Pointer (PCBP) onto the interrupt stack, places a new PCBP into the PCBP register, sets the Process Status Word, Program Counter, and Stack Pointer registers, and then exits.

Disable Virtual Pin and Jump: DISVJMP changes the CPU to physical addressing mode (disabling the MMU) and puts a new value into the PC register.

Enable Virtual Pin and Jump: ENBVJMP enables the virtual address pin for the CPU, allowing the MMU to perform address mapping for the virtual environment.

Interrupt Acknowledge: INTACK is used to generate an acknowledge signal from the CPU to an interrupting peripheral. This allows the system to correctly acknowledge interrupts.

Return to Process: RETPS terminates the current process (without saving its context) and returns to the process whose PCBP is on the top of the interrupt stack.

Wait for Interrupt: WAIT halts the CPU, stopping instruction fetching and execution until an interrupt or external reset occurs.

Gate: GATE is used to change the value of the PSW and PC registers of the CPU, potentially placing the CPU into a new processing mode. The instruction retrieves new PSW and PC register values from a protected memory area (write permission while in kernel mode only) to prevent unexpected processing mode changes. GATE is used in conjunction with RETG, for valid processor execution mode switches.

Move Translated Word: MOVTRW tells the MMU to intercept the virtual address sent by the processor, translate it, and return the physical address to the destination specified in the instruction.

Return From Gate: RETG is very similar to a simple return from subroutine instruction, with the exception that RETG enforces a linear ordering of execution modes. The linear ordering will not allow the new execution mode to be more privileged than the current mode. RETG is used in conjunction with GATE to switch the execution mode of the processor.

Registers

Sixteen 32-bit registers are provided with the CPU; nine of these registers are for general use (r0 - r8), while seven are special-purpose registers (r9 - r15). The intended functions for these registers are described below.

Operating System Support Registers

The WE 32100 supports the abstraction of processes through the use of three kernel-privileged registers, the Process Status Word, Process Control Block Pointer, and Interrupt Stack Pointer. In this context, kernel-privileged means that the register can only be written while the CPU is in kernel execution mode.

The Process Status Word (PSW or r11) contains status information about the microprocessor and the current process. The PSW also contains four condition code flags used in transfer-of-control instructions.

The Process Control Block Pointer (PCBP or r13) points to the starting address of the process control block for the current process. The process control block is a data structure in main memory containing the hardware context of a process when the process is not running. This context consists of the initial and current contents of the PSW, Program Counter, and Stack Pointer; the contents of the user registers; boundaries for an execution stack; and block move specifications (whether or not block moves are allowed) for the process.

The Interrupt Stack Pointer (ISP or r14) contains the 32-bit memory address of the top of the interrupt stack. This stack is used when an interrupt request is received. In addition, the stack is used by the call process (CALLPS) and return to process (RETPS) instructions.

Conventional Registers

The CPU has nine general-purpose registers (r0 - r8), the Frame Pointer, Argument Pointer, Stack Pointer, and the Program Counter registers. These registers are all accessible (for both read and write access) in any execution mode of the processor.

The general-purpose registers (r0 - r8) are used for intermediate data storage, arithmetic, data transfer, logical, and program control assembly instructions. Registers r0, r1, and r2 are additionally used for string manipulation and transfer instructions and for return code values in C programs.

The Frame Pointer (r9) and Argument Pointer (r10) registers are used primarily for support of higher-level programming languages. By convention, the Frame Pointer points to the beginning location in the stack of a function's local variables. The Argument Pointer points to the beginning location in the stack to which a set of arguments for a function have been pushed.

The Stack Pointer (SP or r12) contains the address of the top of the current execution stack, i.e. the next available memory location on the stack for data storage.

The Program Counter (PC or r15) contains the address of the currently executing instruction or, on completion, the starting address of the next instruction.

Interrupt Handling

The WE 32100 accepts fifteen levels of interrupts. An interrupt request is made to the processor by placing an interrupt request value on the interrupt priority level pins of the CPU or by requesting a nonmaskable interrupt by asserting the NMINT line of the CPU. Pending interrupts are not acknowledged until the current instruction completes, except in the case of instructions which must loop in the course of their processing, such as the MOVBLW (move block) instruction. In the case of these instructions, interrupts are enabled at the end of each pass through the loop, and disabled at the start of each pass through the loop.

The pending interrupt value is compared to the value contained in the Interrupt Priority Level (IPL) field of the PSW. For the pending interrupt to be acknowledged, its inverted value must be greater than the IPL value, except for the NMINT, which is always received.

The processor acknowledges an interrupt by placing the inverted interrupt request value on the address bus. The interrupt request value is received by the interrupting device, which then returns an 8-bit offset into the full interrupt table which then points to an interrupt handler for execution. Upon completion of the interrupt handler, the next instruction from the interrupted process is executed. For NMINT interrupts, the CPU assumes address 0x8c (hex 8c) contains the PCBP of the interrupt handling routine; thus it transfers to this PCBP when an NMINT occurs.

Math Acceleration Unit

The WE 32106 Math Acceleration Unit (MAU) is supported in System V/MLS, and is used to provide increased system performance. The MAU provides floating-point capability for System V/MLS, in single, double, and double-extended precision. Operands, results, status, and commands are transferred over an internal system bus, providing the interface to the host processor.

Memory Subsystem

System V/MLS makes use of the WE 32101 Memory Management Unit to provide a separate virtual address space for each user. This virtual environment allows the system to support multiple users, while maintaining separation between those users and the kernel code and data.

Memory Layout

The virtual address space is divided into four sections by the MMU; each section is up to 1G byte in size. Two sections are used to map kernel address space, and two are used to map user address space. Each section is then further subdivided into segments, which is in turn divided into pages. Pages are 2K bytes in extent.

System V/MLS supports virtual memory by paging. The paging scheme allows a process to exist in primary memory with a minimal memory requirement, thus allowing more processes to be active at any given time than could actually fit into memory concurrently. At any point in time, some pages for a given process may reside in primary memory while others exist on secondary storage. If a process attempts to access a page that is not resident in primary memory a fault occurs, and the needed page is brought in from secondary memory. The page replacement algorithm is called "least recently used second chance replacement". See page 24, "Memory Allocation", for additional information.

Address Translation Mechanism

The CPU generates a virtual address, which is translated by the MMU into a corresponding physical address. This translation process involves the actual virtual-to-physical translation and checks to determine that access should be granted to the requested memory page based upon the segment based access permissions.

The MMU performs address translation (see Figure 1) using descriptors that contain the information necessary for segment and page mapping. The MMU has two types of descriptors: segment descriptors (SD) for mapping the paged segments, and page descriptors (PD) for mapping pages within the paged segments.

A SD consists of two 32-bit words. The first word contains information about the segment, while the second word contains the physical address for the Page Descriptor Table (PDT) for the segment. Information within a SD identifies whether the segment is present in memory, has been modified, is cacheable, has been referenced, what the access permissions for the segment are, and other data about the segment.

A Page Descriptor Table is associated with each memory segment in the system. The PDT maintains page descriptors for each page within a segment. It is these page descriptors that ultimately reflect the physical location of a memory page. Page descriptors are composed of information such as the physical presence of the page, modified status, reference indicator, physical address of page, and similar status data.
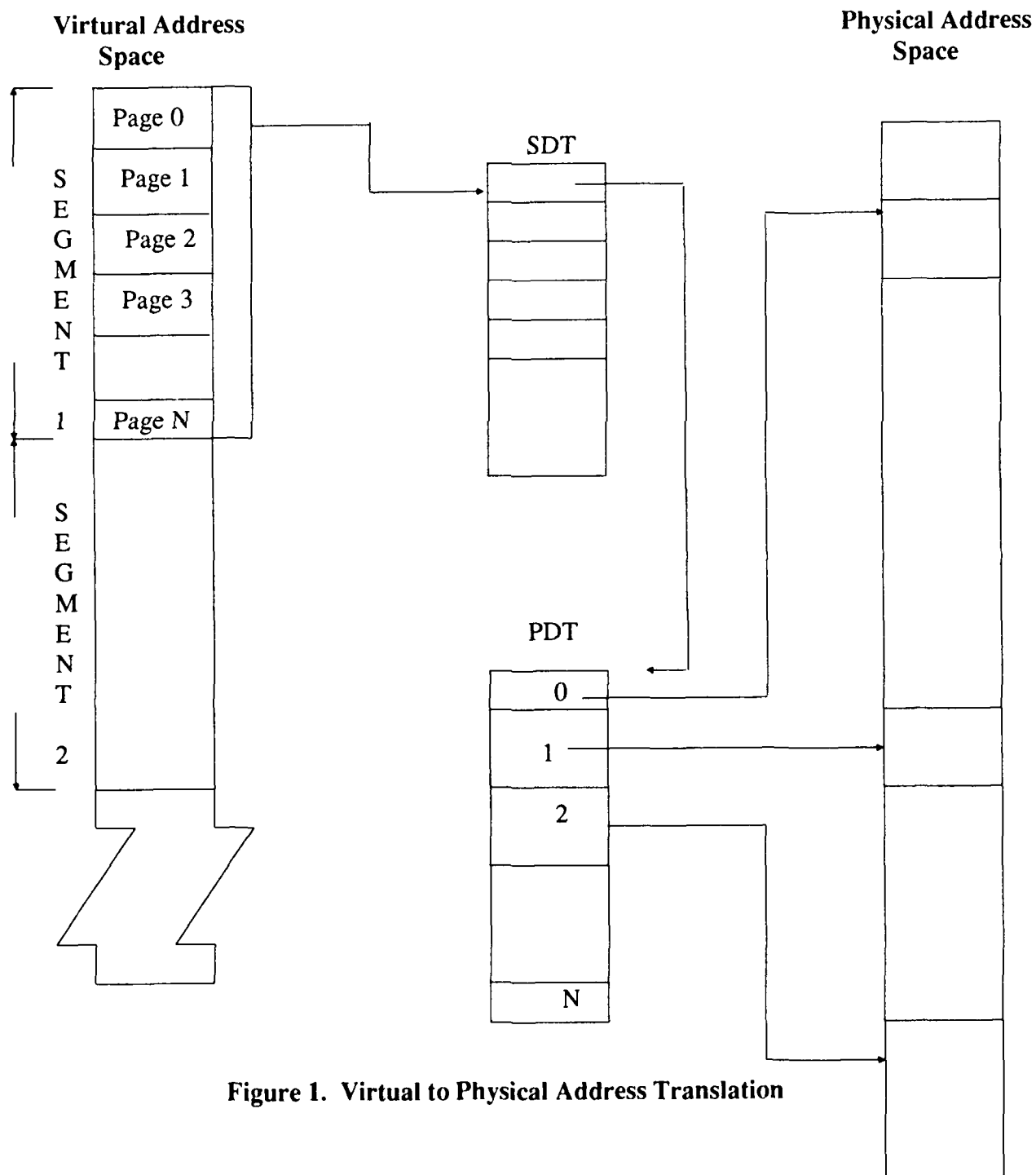
Figure 1. Virtual to Physical Address Translation

Cache Memory Support

System V/MLS is capable of supporting an optional virtual cache plug-in board (on both the AT&T 3B2/500 and 3B2/600). This virtual cache (Vcache) board examines the virtual address passed from the CPU to the MMU, and determines if the address is a valid cache entry. If valid, the Vcache performs the read/write that was requested by the CPU; if not valid, the Vcache allows the request to continue into the MMU. When the data is returned from the MMU (on read operations), the Vcache copies the data into its cache for later use; for write operations which are cache hits, the cache is written through to main memory. When a process switch occurs, the Vcache is flushed.

Memory Protection Features

The MMU controls access to memory segments through the use of an access permission field in the SD. Whenever a segment descriptor is used to perform a translation, the MMU checks the access permission fields of the SD, the type of access being requested by the CPU, and the execution mode at which the access is being requested. If the MMU determines that the access is not allowed under the given conditions, an access rights exception occurs.

The MMU uses the execution mode (kernel, executive, supervisor, or user) at which the access is being requested and the access permission field of the SD to determine whether access is allowed. Allowable permissions are read/write/execute (RWE), execute only (EO), read execute (RE), or no access (NA) permission to the segment. A segment has four sets of permissions, one for each execution mode. All processes executing in a given execution mode receive the same access to any given segment. The hardware prevents a process from operating in either executive or supervisor mode, so the permissions for these two modes are never used.

Associated with each physical memory page is a "page modified" bit which indicates whether the page has been written to since last brought into memory. If this bit is set, then when the page is deallocated, it will be written to secondary storage.

Input/Output Subsystem

System V/MLS supports many devices in its standard and optional configurations: these include floppy disk drives, fixed (hard) disk drives, cartridge tape drives, and a printer. Access to these devices by the system (or users of the system) occurs via several system interfaces. These interfaces are indirectly responsible for maintaining separation of information between multiple users, as explained below.

The interface to an external device consists of two pieces: the I/O bus of the system and the plug-in interface card. The I/O bus provides power and signal connections for the plug-in cards and simply provides a means by which information is transferred between the CPU and external devices (via the interface card). Interface cards are more complex since they often support multiple devices and must be relied upon to store and retrieve information from/to the CPU. Given this increased logic necessary on the interface card, we have examined the available interface cards to determine those which are suitable in a trusted computing environment. An exhaustive list of components included in the evaluated configuration is available in Appendix A of this report.

System Interfaces

This section of the report discusses the interface cards available, the functional aspects of each card and how the card can be used in a trusted computing environment. Also included in this discussion are plug-in cards that extend the system's capabilities, such as memory extension cards.

Ports (CM195B) Card and HiPorts (CM195BA) Card

The Expanded I/O Card (also known as the Ports card) provides four separate asynchronous serial port (RS-232C) interfaces and one parallel port interface. The maximum throughput of the Ports card is 19,200 bits per second. The throughput of the Ports card model CM195BA (also known as High Performance Ports card) is 38.4 bits per second.

The two models differ in their internal operation, with the CM195BA being capable of performing with increased efficiency.

Implementation

The Ports card can be considered as having two distinct sections. The first is the serial and parallel port interfaces. The second is the interface to the I/O bus, the Common I/O (CIO) hardware.

The serial and parallel port interfaces are implemented through the use of two Dual Universal Asynchronous Receiver/Transmitter (DUART) chips and a CENTRONICS[1] parallel interface. The DUART chips provide four asynchronous serial ports, denoted as subdevices SD0-SD3, which can operate in either polled or interrupt mode. Hardware drivers are used to interface the DUARTs to modems or terminals, per RS-232C specifications. The parallel port interface allows for polled access to an external device (printer). Ports card firmware handles all handshaking with the external device, and does not allow any interrupts from the device to be received by the system bus.

The I/O bus interface, known as the CIO hardware, is the second major component of the Ports card. This section of the card is composed of an INTEL 80186 processing unit, Signetics 82S105 I/O bus controller, several support registers, on-card RAM and ROM, and miscellaneous support logic.

The 80186 processor is responsible for the overall activity of the Ports card. This includes the relay of information between the bus controller and the DUART chips, maintenance of on-chip registers and memory, and the execution of any "pump" code that is downloaded from the system. Pump code provides a mechanism by which the actions of the card can be tailored by the host system; that is, the card executes the pumped software versus the standard ROM code.

---

1   CENTRONICS is a registered trademark of CENTRONICS Data Computer Corporation

The 82S105 is the control interface to the system I/O bus. The 82S105 responds to three activities: 80186 read/write of main memory, system board read/write of the Ports card, and acknowledgement of interrupts from the system board.

The support registers on the Ports card contain information such as the Identification Code of the card, the Interrupt Vector of the card, a 128K byte address range assigned by the system unit in accordance with its memory mapping, and control and status information of the Ports card components.

The Ports card RAM is 32K bytes in size, and accessible only by the 80186. This storage area contains any downloaded code from the system unit and any locally stored variables. The ROM is 16K bytes in size and contains the firmware executed by the 80186. It is this firmware that provides the basic capabilities and diagnostics of the Ports card.

A few routine diagnostic tests are run for the Ports card each time the 3B2 computer is powered up. A complete set of diagnostics can also be requested through the use of the diagnostics monitor (DGMON). Potential problems that are identified by the diagnostics routines include: defective Ports card, defective I/O bus slot, defective system board, or defective diagnostic code, among others.

Use in a Trusted Environment

The Ports card maintains separation between each serial port and the parallel port, ensuring that no data will be compromised. The logic within the card is designed to simply forward information between the system unit and peripheral device. The objective of this design is to ensure that any security related control sequences (such as trusted path) will be passed to the system unit for processing. The team has verified this through testing.

EPorts Card - CM195Y

The Enhanced I/O Card (also known as the EPorts card) provides eight separate asynchronous serial port (RS-232C) interfaces. The EPorts card provides the same features as the Ports card (with the exception of the parallel port) with higher throughput (38,400 bits per second) and hardware flow control.

EPorts supports the standard software flow control, as well as two methods of hardware flow control. In one method, the receiving device must use the Request To Send (RTS) and Clear ToSend (CTS) signals for flow control. In the other method, the receiving device must use the Data Terminal Ready (DTR) signal.

Implementation

The EPorts card is identical to the Ports card in its support of the eight asynchronous serial ports. See page 11, "Implementation", for a detailed discussion of the implementation.

Use in a Trusted Environment

The EPorts card maintains separation between each serial port, ensuring that no data will be compromised. The logic within the card is designed to simply forward information between the system unit and peripheral device. This design is intended to ensure that any security related control sequences (such as trusted path) will be passed to the system unit for processing. The team has verified this through testing.

SCSI Host Adapter - CM195W

The Small Computer System Interface (SCSI) Host Adapter Card provides an asynchronous, single-ended SCSI bus interface. The bus interface is capable of supporting up to seven peripheral controllers. The 3B2 computer can support up to eight SCSI Host Adapters simultaneously.

Implementation

The SCSI Host Adapter card can be considered as having two distinct sections. The first is the SCSI protocol controller. The second is the interface to the I/O bus, the CIO hardware, as discussed previously.

The SCSI protocol architecture can be viewed as consisting of four protocol levels:

Level 0 - is the electrical interface and signaling protocol defined in the ANSI specification. This level is implemented in hardware, under the control of the Host Adapter firmware.

Level 1 - is the message system that the Host Adapters and controllers use to communicate. It is used to transfer information about the bus, controller, and request status under control of the firmware in the Host Adapter and controllers.

Level 2 - is defined as the SCSI command level, to provide a means to direct the controller's activity.

Level 3 - is defined as the user interface to the target drivers. Examples of this level include user requests such as read, ioctl, etc.

Levels 0 and 1 are implemented within the SCSI Host Adapter through the on-card NCR 5386 SCSI Protocol Control Chip (SPCC). The SPCC is controlled by an INTEL 80186, part of the CIO hardware which regulates data flow between the I/O bus and the SPCC, and ultimately controls flow to external peripheral devices.

The SPCC is responsible for translating data between the CIO expected format and the SCSI bus format. This translation occurs at the request of the 80186, via CIO initiated requests and in response to interrupts from peripheral devices.

The SCSI Host Adapter is capable of receiving software from the host system, and executing that software within the 80186. Use of the SCSI Host Adapter in this manner allows the system to be customized according to the configuration of peripherals.

The SPCC is supported by several status registers, which are used to indicate the status of the SCSI card, transactions, and interrupts.

The SCSI Host Adapter has three basic types of diagnostics available. The SCSI Host Adapter performs diagnostics during power-up, system boot, and at the request of the system administrator. The diagnostics performed basically determine that the SCSI Host Adapter operation is functionally correct.

Use in a Trusted Environment

The SCSI Host Adapter maintains separation between the devices connected to its bus. This separation ensures that no data will be transferred to the wrong peripheral. The logic within the card is designed to forward data between the system unit and peripheral devices. It is the responsibility of the system unit to specify which device is to receive the data and to store any sensitivity label information with the data. The team has tested this unit to provide assurance that it is capable of appropriately separating data between peripheral devices.

Memory Extensions

The 3B2 computer supports several memory extension cards; the cards available are: 2M byte - CM523B, 4M byte - CM523A, and 16M byte - CM523D. These cards provide an extension of the random access memory used by the system, thus allowing increased system performance. Each of these cards contains 32-bit RAM with 12-bit Error Correction Codes, capable of detecting all two-bit errors and detecting and correcting all one-bit errors.

The memory cards indicated above are controlled and maintained by the operating system of the computer, and are trusted to maintain the integrity of the data which is stored within the card.

Virtual Cache Board - CM522A

The Virtual Cache (Vcache) Board is used to provide a virtual cache environment for the system unit. See the discussion on page 10, "Cache Memory Support" for additional detail.

Interface Cards not in Evaluated Configuration

The following components were not configured into the system which the team evaluated. As such, they are excluded from the evaluated configuration.

Co-processor Board - CM527A

The Co-processor Board utilizes the WE 32100 microprocessor to provide co-processing assistance to the system unit. Co-processing systems may present additional obstacles to implementation of a trusted system, and the vendor has chosen not to submit this co-processing package to the team as part of the evaluated configuration. As such, the Co-processor Board is not considered an option for use in a trusted computing environment, and not included as an acceptable feature card for the System V/MLS system in its evaluated configuration.

Network Interface Card - CM195A

The Network Interface Card is used to connect the 3B2 computer to other 3B2 computers and ETHERNET[1] compatible interfaces. The functionality provided by the Network Interface Card is such that all computer systems with access to the transmission cable are capable of accessing any data which is transmitted on the network. The Network Interface Card is not capable of ensuring that sensitivity labels associated with data will be maintained during transmission. From a security standpoint, this functionality is unacceptable and prevents the Network Interface Card from being considered an option for use within a trusted computing environment.

Network Access Unit - CM195U

The Network Access Unit is used to connect the 3B2 computer to an AT&T STARLAN Network. The functionality provided by the Network Access Unit is such that all computer systems with access to the transmission cable are capable of accessing any data which is transmitted on the network. The Network Access Unit is not capable of ensuring that sensitivity labels associated with data will be maintained during transmission. From a security standpoint, this functionality is unacceptable and prevents the Network Access Card from being considered an option for use within a trusted computing environment.

Remote Management Package/Alarm Interface Circuit - CM195AA

The Alarm Interface Circuit (AIC) card is designed to provide an interface between the 3B2 computer and certain external devices, and to allow three different specific functions: allow for a remote console terminal; generate external alarms when the system's sanity has failed; and provide panic alarm capability, via an interface to other external devices. Of these functions, implementation of the first would invalidate the system's rating by providing for an extension of the TCB hardware outside of the protective perimeter in which the CPU and peripherals reside. The other two are ineffective unless the AIC is connected to some monitoring device external to the machine. In the latter case, the monitoring equipment would not have been evaluated, and as such, cannot be trusted

---

1   ETHERNET is a trademark of the XEROX Corporation

October 18, 1989

not to violate the system's security policy. Thus, the Alarm Interface Circuit board may not be considered an option for use within a trusted computing environment.

Input/Output Devices

As with most machines, the 3B2 computer relies upon external peripherals to perform useful services. Peripherals such as fixed disks, floppy disks and tapes are widely used in the 3B2 computing environment. The previous discussion regarding I/O device interfaces, discussed the mechanism by which external peripherals can be connected to the 3B2. See Appendix A for a list of devices which may be connected to the system, and trusted to maintain the integrity of the data that flows through or into the device.

It should be noted that the system must have a "console", which is a terminal plugged into a special port reserved for that purpose on the system board. The console functionality is the same as all other terminals on the system (all users must login to access the system), except that some system software directs error messages and notification of significant events to the console. Often a printer is connected in parallel with the console, to generate a hardcopy of these messages.

The 630 MTG Intelligent Terminal

The evaluated configuration of System V/MLS includes an intelligent terminal, the 630 MTG (Multi-Tasking Graphics) terminal, which can be used as a user's terminal. The 630 MTG terminal provides capabilities that include the ability to scroll and save text in any window, as well as the ability to "cut and paste" text from one window into another.

The terminal firmware demultiplexes the communications from the host, passing the data to the terminal resident program (*wproc*) controlling the individual window. This is accomplished on the 630 MTG side of the physical port by a demultiplexer/controller, *demux*, which receives the communications from the host. Data intended for a window is simply passed to that window's *wproc* while control information (i.e., download initiation signal) is handled directly by *demux*. More information on *wproc* will befound on page 77, "wproc". The terminal firmware also interprets commands to create, delete, move, and reshape windows, and manages other terminal capabilities such as cut/paste.

There are a number of security-relevant aspects to such a device. The following description of the implementation of the 630 MTG terminal provides an outline of those issues.

The 630 MTG contains a Motorola 68000 microprocessor and 640 Kbytes of dynamic RAM (expandable to 1M byte). The 630 MTG also contains 384K bytes of EPROM, with a cartridge port on the side of the unit allowing an additional 384K bytes of EPROM to be plugged in. There is no memory management or paging functionality implemented in the terminal hardware. The video display is bitmapped, with a resolution of 1,024 x 1,024 pixels. The terminal incorporates two RS-232 ports for communications with one or more hosts. The evaluated configuration has only one host enabled, since there would be no way to partition the 630 MTG's address space for the two hosts. The terminal supports a printer port which is also disabled in the evaluated configuration,

since its output would not be labeled according to the labeling requirements.[1] The terminal also supports a mouse for easy manipulation of the terminal's many features.

The terminal runs a trusted version of the standard 630 MTG terminal emulator firmware, which implements a trusted communications path between the user and the host computer in addition to providing labeling, mandatory access control, and preventing object reuse in accordance with the B1 requirements. Communication between the terminal and the trusted host software takes place via one of the 8 virtual terminal connections (channels) which the 630 MTG terminal is capable of supporting. All control communication, such as setting up and labeling windows, is carried by channel 0 (virtual terminal device 0, referred to as "xtnm0", where n and m are digits used for terminal identification only, and will henceforth be omitted). User windows may occupy channels xt1 through xt7. The control window, xt0, is invisible to the user.

The host device driver (xt driver), which communicates directly (sends data over the physical connection) with the 630 MTG terminal, ensures data separation at the virtual terminal level, much as ports ensure data separation at the physical terminal level. The details on how this is accomplished are described on page 77, "Window Labels".

The 630 MTG does not provide memory management within its local address space, so no program which is untrusted may be downloaded and run on the 630 MTG without invalidating the system's rating. For this reason, it is only possible to download code from the directory /usr/dmd/bin, in which only trusted code resides. Downloading may only occur through the action of /usr/bin/dmdld, which is hard-coded such that it will only download programs from that directory (see page 80, "Downloadable Software"). The evaluated configuration includes two such programs: fw.mods and chk630. These are explained on page 75, "The 630init Process".

There are four major 630 MTG firmware components:

- System - contains the round-robin scheduler, system initialization logic and system processes that handle global mouse interaction, window manipulation, key translation and host I/O multiplexing.

- Application - contains application level processes that perform terminal emulation, set the terminal characteristics and program the programmable function keys.

- xt protocol - implements a communication protocol between the host and 630 MTG. This is the protocol used in the layers environment.

---

[1]   Physical security is relied upon to prevent the printer or second host port from being enabled after the initial 630 MTG terminal session connection with the host is made. These ports are initially assured to be disabled as part of the terminal initialization process.

- Libraries - contains routines used by firmware components and downloaded programs. The library routines are accessed through the firmware vector table, which is a table containing addresses of functions and global variables which are indirectly called and accessed by the 630 MTG. The firmware vector table is located after the exception vector table in RAM and is located before a4K byte pad area which exists in case the vector table (or ROM bss, which is on the other side of the pad area) needs to be extended.

630 MTG Memory Layout

The 630 MTG does not support virtual addressing and has no memory protection except that provided by using ROM to hold the terminal firmware. The 630 MTG me ury is displayed as:

Low Memory	000000
Vector Table(s)	
Libraries text	
Data Section	
Reserved	

EPROM

Cartridge (append) NOT USED	100000

I/O Registers	200000

Expansion I/O Card NOT USED	400000

Video Controller	700000

Video RAM	760000

Program

RAM

Vector Table(s)	
Pad Space	
Rom bss	
Cartridge bss NOT USED	
Available	

Expansion RAM Card	800000

NVRAM	FEE001
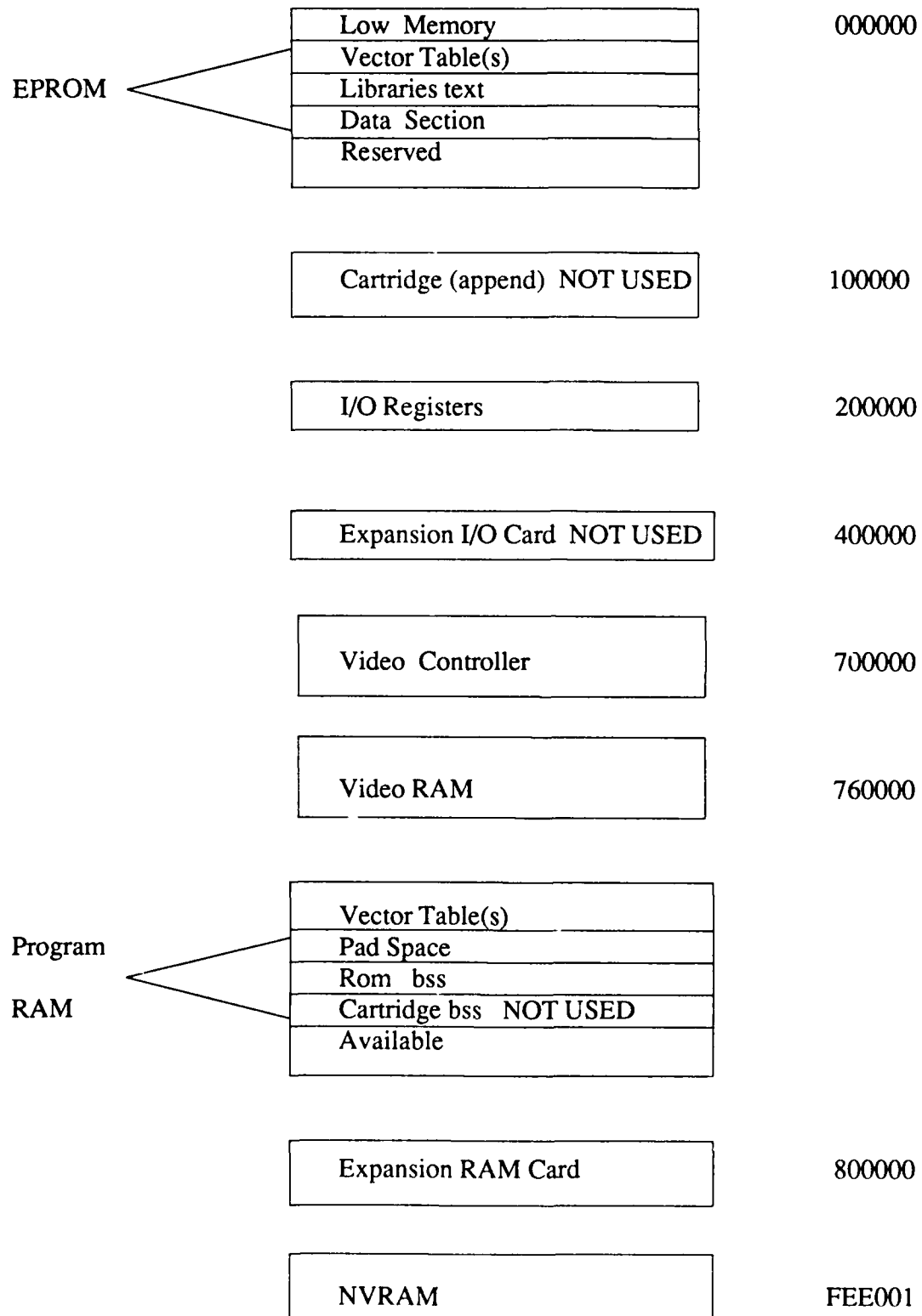
FIGURE 2. THE 630 MTG MEMORY LAYOUT

Figure 2 from top to bottom describes the memory layout of the 630 MTG. The first item in low memory is the table of interrupt/exception vectors. These interrupt vector entries point to the interrupt vector table in RAM. The RAM interrupt vectortable entries contain jump instructions to the real interrupt/exception handlers in ROM. Following the interrupt/exception table is the firmware vector table which gets moved to RAM with initialized data structures on reboot. All firmware function calls go through the RAM table, whereas interrupts/exceptions go through both tables. Following both these ROM vector tables are the text and data sections of the firmware. These are the components of the standard EPROM.

Video RAM is dual ported so that video control and CPU processing can proceed simultaneously. Following the video RAM are the RAM interrupt/exception vector table (pointing to ROM located handlers) and the RAM firmware vector table (pointing to ROM located firmware routines). The pad area exists so that the firmware can be modified and thus grow in size.

The Non-Volatile Read-Access Memory (NVRAM) is battery backed and holds terminal setup data and the character strings associated with the programmable function keys.

630 MTG Memory Management

The 630 MTG performs memory management through its firmware; there is no special hardware to manage memory. The main tasks in managing memory on the 630 MTG are to save and retrieve portions of windows which may become obscured by other overlapping windows, and to support the downloading functionality.

There are three types of memory, which are illustrated in Figure 3: memory blocks obtained from an alloc pool, a gcalloc pool and a combination of the two called a gcastray block.

alloc grows up →   ← gcalloc grows down

| alloc block | | alloc block | gcastray block | alloc... block | | gcalloc block |

alloclevel⏤
gclevel⏤
alloclimit ⏤

Figure 3. 630 MTG Memory Management

Memory allocated with *alloc* starts at the low end of user memory space and grows upward in memory. It is non-compactible and is therefore limited by some upper bound stored in the variable allochimit. *alloc* employs a first-fit algorithm, combining contiguous free blocks. *alloc* zeros out the requested memory (i.e., fills with zeros). Blocks cannot be allocated with *alloc* past allochimit or beyond the first block allocated via *gcalloc* encountered by *alloc* (the gclevel). Memory allocated via *alloc* is used for downloads, thread stacks, window structures, PFkeys, and other general dynamic storage (identified on page 73, "The 630 MTG Terminal Implementation").

gcalloc stands for garbage collectable allocated memory. Memory allocated via *gcalloc* starts from the high end of user memory space and grows downward in memory to the alloclevel b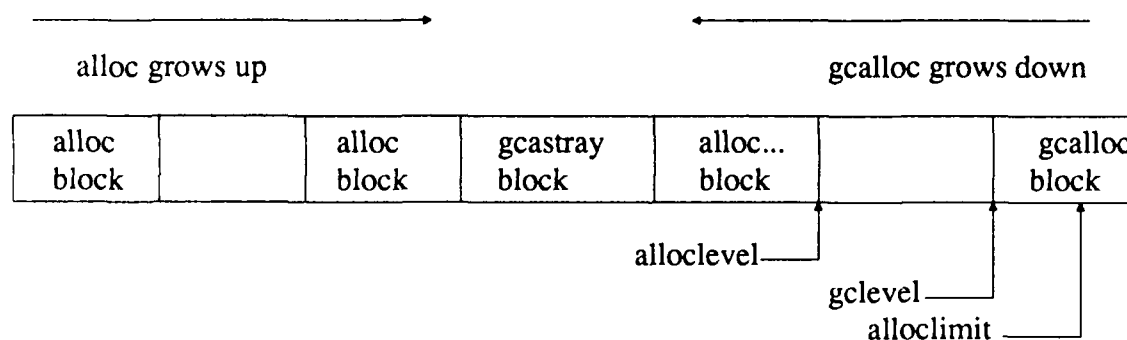oundry and is compactible. *gcalloc* allocates blocks from the lower level of the pool (i.e., it lowers gclevel). If there is not enough space at the low end of the pool (i.e., if it reaches alloclevel), the pool is compacted towards its upper boundary. Memory allocated via *gcalloc* is primarily used to store lines of text in windows.

gcalloc allocates blocks in the alloc pool when there is not enough space in the gcalloc pool after compaction to satisfy a request. A block allocated this way is referred to as a *gcastray* block.

Use in a Trusted Environment

The 630 MTG terminal allows a user to work with multiple terminal sessions operating at various labels (although two hosts are possible, only one host connection is allowed on the evaluated configuration). Only one user login session is active, although many host connected and local windows can be active at any time. The 630 MTG provides an interface which appears as a complete and independent terminal device to each host process connected to the terminal. In order to implement this functionality, the terminal firmware partitions the terminal memory to support up to eight host windows (seven windows accessible to the user; one for control information). Although the evaluated configuration of the 630 MTG terminal does not incorporate all of the functionality of the stock 630 MTG, it still retains a wide variety of useful capabilities unique among the systems evaluated to date.

Physical Security Considerations

The AT&T 3B2/500 and 3B2/600 computers, on which System V/MLS runs, have physical dimensions which are small enough so that the entirety of the TCB hardware could easily fit on or under a desk in the average office. Because of this small size, it is especially important to note here that there are certain physical security considerations involved in appropriate deployment of any computer system which is expected to process sensitive information.

Primarily, the system administrator must understand that any computer relies upon its physical security as a basis for all other security which it provides. A computer which is left unattended and unprotected in the presence of untrusted individuals is liable to be tampered with; therefore, the system administrator should take precautions that all elements of the TCB hardware are protected in a fashion appropriate for the most sensitive information on the system. The single exception to this rule is that remote devices, such as terminals and printers, should be protected as appropriate

for the most sensitive information which they are capable of accessing. For example, a terminal may have a device maximum of SECRET, although the system high is TOP SECRET; in this case, the terminal should be protected as though it is SECRET information.

The Firmware Password and The Floppy Key

As an additional security enhancement, the 3B2/500 and 3B2/600 computers require a password before allowing access to "firmware mode", from which system diagnostics and other low-level functions (including selection of bootstrap load device) may be accessed.

Since use of the firmware password is likely to be an infrequent event, it is quite possible that the system administrator may forget the firmware password. To remedy this problem, the 3B2/500 and 3B2/600 come with a device referred to as the "Floppy Key." This is a floppy diskette which is keyed to the serial number of the particular 3B2 computer on which it was generated. The Floppy Key is meant to be used in one of two cases: either the root or firmware password is lost, or the system battery is weak and parts of the system Non-Volatile RAM (NVRAM) become unreadable and must be restored.

The Floppy Key may be used to reset the system NVRAM to its default values, and to gain access to the system in firmware mode. It is therefore imperative that the Floppy Key be afforded the same protection that would be provided for the most sensitive data on the computer with which it is associated.

Software Architecture

System V/MLS enforces security through the use of hardware (as previously discussed) as well as software. The software of the system provides the interface to the TCB and is responsible for determining access to the objects controlled by the system. The following is a discussion of the system software architecture and the mechanisms provided within the kernel.

TCB Boundary

First it is important to note the boundary of the TCB. The TCB is made up of hardware, firmware, and software components. Hardware and firmware components were discussed previously on (see page 3, "Hardware Architecture") and on page 16, "The 630 MTG Intelligent Terminal". Software components consist of all trusted routines. Trusted routines are all routines running in kernel space (including 630 MTG terminal routines) as well as trusted processes running in user space. The system's kernel contains those routines running in kernel space and are listed in Appendix B. Trusted processes are defined on page 67, "Trusted Processes", and are listed as part of Appendix C. The kernel, trusted processes, and 630 MTG terminal will now be discussed.

Kernel Architecture

Overview

The System V/MLS software architecture can be viewed as a series of conceptual layers. It is important to note that the term "layer" as used in this sense throughout this section is in no way to be confused with the system engineering technique of "layering" that can be found in the design of higher level TCSEC systems. The innermost layer is the system hardware, followed by the kernel routines and system commands. User programs may then logically be viewed as being built upon the system commands and kernel interface, in the outermost layer.

Hardware Layer

System V/MLS takes advantage of the processing modes provided by the hardware layer (see page 3, "Hardware Architecture") to provide isolation between the system and user spaces. The two execution modes used by the system are:

- User mode: Processes in user mode can access their own instructions and data but not privileged instructions and data (or those of other processes). Execution of privileged instructions results in an error.

- Kernel mode: Processes are permitted the execution of all system instructions and can access kernel and user addresses.

Kernel Layer

The kernel layer provides the system call interface between user programs and the TCB. The kernel performs various primitive operations on behalf of user processes to support the user interface described below. Among the services provided by the kernel are:

- Controlling the execution of processes by allowing their creation, termination or suspension, and interprocess communication.

- Scheduling processes fairly for execution on the CPU.

- Allocating main memory for processes.

- Allocating secondary storage for long term data storage.

- Allowing processes controlled access to peripheral devices such as terminals, tape drives and disk drives.

The kernel provides its services transparently for the user. When a process executes a system call, the execution mode of the process changes from user mode to kernel mode, and the operating system executes and attempts to service the user request.

Scheduling:

Processes share the CPU in a time-shared manner, meaning each process is allowed up to one second of execution time at any one time. If a process happens to be in the middle of a system call at the one second interval, then that process is permitted to finish the system call before it is preempted. Processes are also preempted when they request a time consuming task, such as physical I/O, or are waiting on a synchronization event.

Once the current process has been preempted, the process with the highest priority is chosen from the ready queue to run next. Processes are given a priority based upon a compute time/elapsed time ratio and a fixed priority class. For example, system processes are given a higher priority class than user processes. Processes at the same priority are effectively executed in a round-robin fashion.

Memory Allocation:

The kernel and all user processes operate in virtual address space. The hardware divides the virtual address space into physical pieces called pages. Each of the four sections of virtual address space is composed of segments which are in turn composed of pages. Regions are the logical representation of pieces of virtual address space as viewed by the system software. Data tables are maintained by the kernel to provide virtual to physical address translation.

As stated previously, there are four sections of virtual memory, each one having a separate segment descriptor table (SDT). The SDTs are located in physical memory and contain the segment descriptors (SDs). Each segment has one page descriptor table (PDT) which contains its page descriptors (PDs). Segments are thus represented by both a SDT entry and an entire PDT.

System V/MLS manages memory by maintaining regions for each process. The regions allocated to a process define the memory space that the process can use during its lifetime. Every process has a pointer (through its process table entry, see page 30, "Process Table") to a data structure known as a process region (pregion) table. At system start up time, memory is allocated for the pregion tables, and a pregion table is associated with each process slot in the process table. Pregion entries contain information about the connection of a region to a process.

Pregion entries map to region table entries which contain a list of SDs. These SDs point to page tables which map to physical pages for the region. This list of SDs is called an rlist. The system region table (region table) contains entries for all active regions on the system. The region table entries contain all the information needed to attach a region to a process. The translation from virtual to physical address is performed within the hardware (see page 8, "Address Translation Mechanism") using this information. When a process is created, it has two SDTs associated with it (as there are two SDTs which map user address space), as part of the process structure. The SDT is a one to one mapping to an rlist of an attached region. When a process becomes active, the rlists for the attached regions are loaded into the appropriate SDT and the address of the SDT is loaded into an MMU register. This segment table information is then used in resolving virtual addresses.

A region is actually a virtually contiguous piece of memory that can be associated with some logical function. The typical types of regions associated with a process are text, data and stack. Two additional types of regions, however, may also be associated with a process; shared memory and shared text (e.g. libraries).

Shared memory regions are associated with a process through the utilization of the IPC shared memory mechanism known as "shared memory segments". The shared memory IPC mechanism provides system calls to facilitate the use of regions in a shared manner. The *shmget* system call creates a shared memory region by allocating aregion data structure and placing a pointer to the region table entry in a table known as the shared memory table. The shared memory region is attached to the virtual address space of the process through the *shmat* system call by allocating a pointer from the pregion table to the associated region table entry.

The sharing occurs when one or more processes attach to the same shared region. This is achieved when processes call *shmat* and provide it the same ID (which was returned by the *shmget* call when the region was created). Before a process can successfully attach to a shared region, however, it must have appropriate discretionary access control permissions for the region. Permission bits for a shared region of this type are kept in its shared memory table entry. Further information on DAC on shared memory objects can be found on page 46, "DAC on System V IPC Objects". Once a shared memory region is allocated to a process, it becomes part of the virtual address space of the process and is maintained by the system in the same manner as are other types of regions.

Shared text regions are also maintained in the same manner as are other regions, but allow for sharing through a completely different method. The header of an executable load module indicates if its text is to be shared. If so, the kernel looks for the original text region in the active region list and if found, attaches it to the process. If a shared text region does not yet exist, a new region (created RE) is attached to the process.

Memory allocation illustrating the use of shared memory and shared text is shown in Figure 4. Note that Process A and Process B are sharing a text region, and Process B and Process C are sharing a shared memory region.

To enhance the effectiveness of the use of shared text, a mechanism known as the "sticky-bit" is provided. The sticky-bit is one of the file mode bits associated with every file. The system administrator can set this bit for an executable file through the *chmod* system call. When a process executes a file that has its sticky-bit set, the text of the file remains in memory even if its region reference count drops to 0. This allows frequently used text regions (i.e., shared text) to remain in memory and thus spare the kernel the overhead of repeatedly having to bring shared text regions into memory.

Given that the demand for memory is often greater than the amount of physical memory available, the system supports page replacement. The algorithm, known as "least recently used second chance replacement", provides for a fair replacement strategy. To implement this algorithm, two bits are needed: 'reference' and 'need reference'. The reference bit indicates that the page has been

referenced by a process. The need reference bit indicates that the page hasn't been referenced, but will probably be referenced soon. When a page is aged the first time, the reference bit is cleared and the need reference bit is set. On the second aging pass, if the reference bit is clear and the need reference bit is set, then the need reference bit is cleared. If both bits are clear, the page is available to be replaced.

Shared Memory Scheme
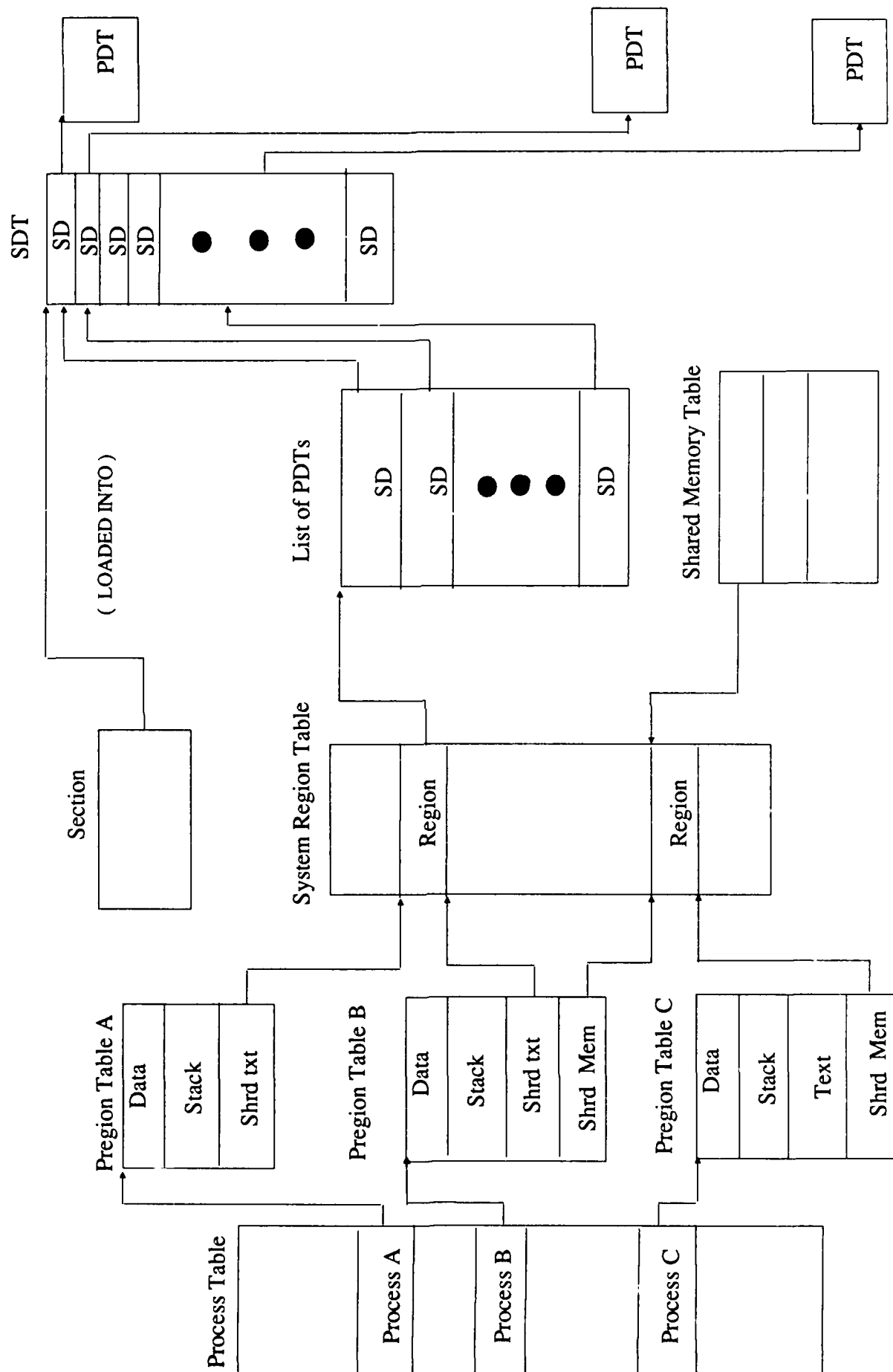
Figure 4. Shared Memory Scheme

October 18, 1989

Command Layer

The command layer provides a convenient interface that users can utilize to request system services. The TCB interface at the command layer is composed of those commands provided with SystemV/MLS, as described in Appendix C.

The "shell", /bin/sh, is the System V/MLS security enhanced Bourne shell command interpreter. The shell is actually both a command interpreter and a programming language. In either regard, it provides an interface to the TCB through which users may execute system and user provided programs which utilize lower level system services. The shell allows for control-flow primitives, parameter passing, variables and string substitution as well as allowing users to customize their own working environment.

The shell is trusted not to modify the information provided by the user and to ensure that it will invoke the actual program specified. System V/MLS incorporates two major changes in the shell in order to increase its level of trust. Upon invocation of a child shell process, the shell will reset the effective UID/GID of that process to its real UID/GID (for discussion of the various process IDs, see page 31, "Process Data Structures"). Additionally, the shell will enforce that any shell scripts run by root must be trusted shell scripts. This is done by checking that the label on the shell script is level 0, SYSTEM, (see page 56, "System Software Integrity") before allowing its execution by root.

For system administrators, the command layer of the TCB provides many programs that can be used to configure, maintain, and control the activities of the system. Functionality provided for the administrator includes adding users, changing file ownership information, and reviewing audit logs.

Those files that the system administrator must rely on to perform his/her duties are considered part of the TCB. For these files, one or more of the following are true:

- the file runs in a privileged execution mode; this occurs when a file is SUID or SGID to an administrative ID, or must be run by an administrator (see page 67, "Trusted Processes").

- the file may be read only by an administrator.

- the file may be written only by an administrator.

Filesystem

Filesystem Overview

The System V/MLS filesystem is a secondary storage allocation and management system for regular, directory and pipe files (see page 38, "Objects"). The kernel allocates secondary storage for user files, reclaims unused storage, and protects user files from illegal access. The filesystem has a tree structured hierarchy. At the base of the tree is the root directory (referred to as "/"). Every

non-leaf node of the tree is a directory of files, and files at the leaf nodes are either directories, regular files, or special files. Files are referenced by a path name,which describes the location of the file within the tree hierarchy.

Special files (also named pipes and directories) are created via the *mknod* system call, which is similar to *creat* in that an inode is allocated for the file. For special files, *mknod* writes a major and minor device number into the inode. The user interface to special files is through the filesystem. The special file occupies a position in the directory hierarchy of the filesystem. Additionally, the normal filesystem system calls (e.g., *open, close, read, write*) have an appropriate meaning for special files.

Files on System V/MLS do not assume any unique structure based upon their content. All files are stored in a similar fashion, and any meaning associated with the stored information is determined by the program accessing the file.

Internal Representation

Inodes are internal, TCB supported storage objects which are used to define and maintain filesystem based objects. The inode contains information related to filesystem objects such as ownership, GID, owner/group/other access permissions, object size, access times, and physical disk addresses for data. Every filesystem object has one inode, but may have several names, all mapping to the same inode.

Inodes are referenced via an inode pointer, stored in a directory. The directory contains pairs of filesystem object names and inode pointers. This allows multiple name/inode pointer pairs to refer to the same inode, allowing multiple names for inodes (and thus files).

System Initialization

The act of loading the kernel system image into memory and starting its execution is known as a system boot. System boot occurs whenever the system is started from a power-up, following system crashes and intentional system shutdowns.

System boot occurs in several phases. In the first phase, the computer hardware loads and executes the first block of data from the bootstrap disk. This data block contains a short bootstrap loader program which finds and loads the file named */unix* in the root directory. The file */unix* contains the machine instructions for the operating system kernel, and its execution comprises phase two of the system boot procedure.

In phase two, the kernel initializes the essential hardware elements of the system, such as the system clock and memory management unit. The kernel also defines system data structures which will be used to support and maintain processes. The kernel then begins to create process 0, the *sched* process. *sched* is created by defining process 0 within the system process maintenance tables.

The system then copies process 0 to create process 1. Process 1 is expanded in size and the machine instructions to invoke */etc/init* are placed within its code region. Process 1 is placed in the CPU ready queue, and invoked by the system scheduler.

Upon invocation by the scheduler, process 1 is considered the *init* process. The *init* process is responsible for setting up then process structure of the System V/MLS system. *init* creates a new process called *getty* for each login device available on the system. *getty* waits for a user to attempt to login at the terminal port associated with the program. During the time when *init* spawns such processes allowing users to login to the system, a transfer is made to multi-user mode. At this time the filesystem is examined to verify its correctness (by */etc/fsck*) and the system audit mechanism is invoked (*/mls/bin/satstart*).

When a user starts to login to the system, *getty* adjusts the line protocol and overlays itself with a new program, *login*. *login* checks and validates the password provided by the user. If the password is valid, *login* then invokes the user's first process. The first process is usually the "shell". For further information regarding the login process, see the discussion on page 62, "User Identification and Authentication".

TCB Protected Resources

Subjects

In System V/MLS, the subjects are processes which execute on behalf of the users. Processes, which may be thought of as programs in execution, are composed of the following logical sections of virtual address space: text, data, stack, and any shared memory regions. The kernel has its own text, data and stack regions. The system region table contains regions for every active process. Users cannot access the kernel regions, but can request the kernel to work on their behalf by using system calls.

Process Data Structures

The data structures associated with processes are the process table, the user area, and the process region table. These tables define the context of a process.

Process Table

The process table, which always remains memory-resident, defines process information to the TCB. Non-privileged users may indirectly modify their own process table entry through the use of system calls.

Important Process Table Fields are:

- process state -
  identifies the status of the process (e.g. ready, waiting, running, sleeping, blocked)

- process priority -

> The scheduler uses this information to determine which process will be selected to run.
> The priority is adjusted when a clock interrupt is generated.

- real user ID (RUID) -

> Records the login ID number of the user responsible for the process. It can be changed by *su*. The real and effective user IDs of a process are inherited from its parent.

- saved user ID -

> The effective user ID number of the process at the time of program invocation (exec). It can be changed by *su* and other setuid to root programs.

- process ID (PID) -

> The TCB assigns a number which at any given time uniquely identifies that process. The TCB assigns process IDs sequentially.

- parent process ID (PPID) -

> The process ID of the parent process.

- process group ID (PGID) -

> A process group is a set of processes sharing the same control terminal. The PGID is the process ID of the process from which all the other process group members are descendents. The kernel uses the PGID to identify a set of processes that should receive a common signal for certain events.

- a signal field (p_hold) -

> Each bit position represents the status (i.e. whether the process is or is not accepting that signal) of a signal for that process.

- a signal field (p_sig) -

> Each bit position represents whether a signal has been posted for that process.

- a pointer to this process's parent process table entry.

- a pointer to the children of this process -

> If a child dies and has children of its own, then the children are inherited by *init*, via the exit system call.

- a pointer to the process region table - (see Figure 5)

- a pointer to the process's user area

User Area

Every process in the process table is allocated a user area which, when paged into memory, is located in kernel address space. User areas contain information that the kernel uses when a process is executing. Only the kernel can directly access the user area of the executing process.

Important fields in the User Area are:

- pointer to the process table - (see Figure 5)

- real and effective user IDs - previously defined

- real group ID -
      Identifies the group associated with the process.

- effective group ID -
      Identifies the current group ID associated with the operating process. This may or may not be the same as the real group ID and is changed via the setgid mechanism (see page 47, "Setuid/Setgid Mechanisms").

Process Region Table

Each process region table (pregion table) is a table used during the mapping of a process's virtual address to physical address. For the implementation details, see page 24, "Memory Allocation". The kernel accesses the pregion table to identify information about the type and virtual address of a region. Each process has its own pregion table.

The Process Region Table Entry Fields are:

- a pointer to the entry in the system region table (the region's descriptor)

- the starting virtual address of the region

- a type field (e.g., unused, text, data, stack, shared memory)

- read-only flag

Entries in the system region table contain the following fields:

- type of the region (e.g., unused, private (not sharable), shared text, shared memory)

- various status flags (e.g., loaded, locked, locked with process waiting, private)

- size of region in pages

- an r-list, which is a pointer to a list of pointers to PDTs and Disk Block Descriptors (see below)

- number of page tables allocated to r-list

- if a region is on the free list, pointers to the regions before and after on the free list

- pointer to inode where blocks are

For each PDT, a Disk Block Descriptor table is allocated (contiguously). For each PDT entry there is a corresponding Disk Block Descriptor table entry which contains information describing a copy of the page on disk, if one exists.

PROCESS TABLE

PROCESS
REGION TABLE

USER AREA
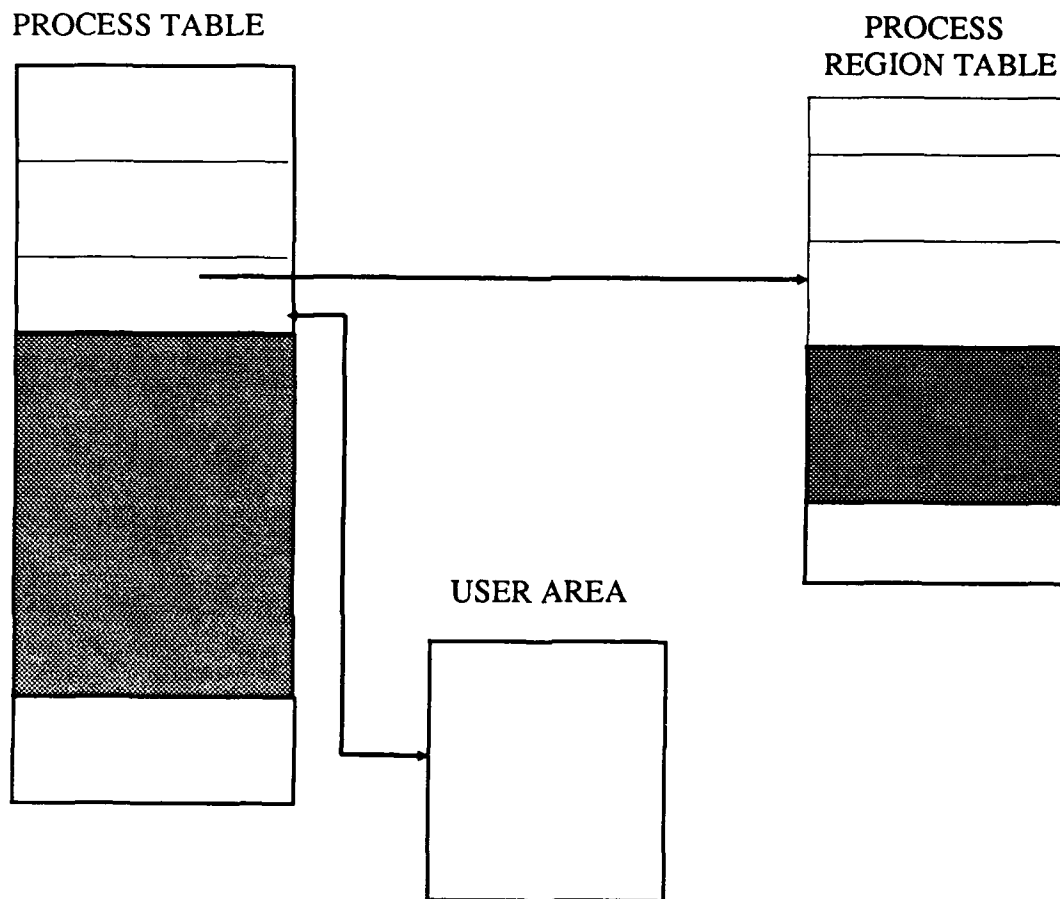
PROCESS DATA STRUCTURES

Figure 5. Process Data Structure

Process Creation and Execution

The only method of process creation available to the user is to invoke the fork system call. The process that makes a fork call is referred to as the parent process, and the newly created process is referred to as the child process. Every process has only one parent, but can have several children. Immediately after execution of the fork call, the only differences between the parent process and child process are the fact that the child process ID and the parent process ID's are distinct, the PPIDs differ, and some accounting flags are re-initialized.

A successful fork system call causes the kernel to perform the following sequence of operations:

- allocates an entry in the process table for the child

- assigns a unique process ID to the child

- copies data (eg. parent process real and effective user IDs, parent process IDs) from the parent process table entry to the child's process table entry

- increments the file and inode table counters

- makes a logical copy of the parent's text, data, stack and user area

- returns the child's process ID to the parent

- returns 0 to child process

After the process has been created by the *fork* call, the *exec* system call can be run to overlay the memory space of the newly created process with a copy of an executable file. The kernel checks the execute permissions for the executable file, and the size of the file against the limits of the invoking process. Next the kernel determines the layout of the executable file and overlays the text and data regions. Upon invoking *exec*, signals set to be caught by the invoking process are set to terminate the transformed process. Finally, the process is placed on the run queue and awaits execution. The *exec* system call will fail if any of the following are true:

- the new process file is not an ordinary file

- the new process is a shared text file that is currently open for write by some process

- not enough memory

- a signal was caught during the *exec* system call

- attempting to load a program whose size exceeds the system limit

- attempting to load a SGID file when no privilege for the combination of security label and discretionary group exists within the system

Signaling

Processes may send other processes signals via the *kill* system call, or the kernel may send processes signals by directly writing the signal into the p_sig field of the process table entry. The following chart identifies the signals supported on System V/MLS:

TYPE	VALUE	MEANING
SIGHUP	01	hangup
SIGINT	02	interrupt
SIGQUIT	03	quit
SIGILL	04	illegal instruction
SIGTRAP	05	trace trap
SIGIO	06	currently used as an abort signal
SIGEMT	07	Alignment Error
SIGFPT	08	floating point exception
SIGKILL	09	kill
SIGBUS	10	bus error
SIGSEGV	11	segmentation violation
SIGSYS	12	bad argument to system call
SIGPIPE	13	write on pipe with no one to read
SIGALRM	14	alarm clock
SIGTERM	15	software termination
SIGUSR1	16	user defined signal 1
SIGUSR2	17	user defined signal 2
SIGCLD	18	death of child
SIGPWR	19	power failure
	20	user defined signal 3
	21	user defined signal 4
SIGPOLL	22	pollable event occurred

When a process sends a signal via the *kill* system call, the real or effective UID of the sending process must be the same as the SAVED or effective UID of the receiving process; the only exception is the case when the effective UID of the sending process is super user (root). The superuser can send a signal to any process.

A signal will not be sent if one or more of the following are true:

- the signal number is not valid

- the signal is a SIGKILL and the receiving process ID is 1

- the real or effective UID of the sending process is not superuser, or its real or effective UID does not match the saved or effective UID of the receiving process

- no process can be found corresponding to the specified process ID

- the system's mandatory access control policy is violated and the effective UID of the sending process is not superuser

The kernel sends a signal to a process by setting a bit in the signal field (p_sig) of the process table entry which corresponds to the type of signal sent. If the process is sleeping at an interruptible priority and receives a signal, the kernel awakens it.

Signals are not queued, so if a process receives the same signal more than once before processing the previous occurrence, then those additional signals are ignored. If a process receives different signals at the same time, then the one with the lower signal number is processed first.

The TCB checks for signals before a process returns from kernel mode to user mode and when it enters or leaves the sleep state. The process that receives the signals may react to them in one of the following manners:

- by default the process calls *exit* and terminates

- the process may ignore the signal (except for SIGKILL)

- the process may execute a particular user function

A process can only respond to a signal while in user mode; therefore signals do not have an immediate effect on processes running in kernel mode. If a process is running in user mode and receives a signal, the kernel will respond to the interrupt and then return to the user process.

Process Termination

The *exit* system call terminates the executing process. The kernel determines if the calling process is a process group leader. If the process is a process group leader, all members of the group are sent a SIGHUP signal, and their process group number is changed to zero.

exit then causes the kernel to do the following:

- disable the process's ability to handle signals

- close all open files associated with the process

- free memory associated with the process by deallocating the appropriate regions

- change the process state to "zombie". A zombie process is one that still has an entry in the process table but doesn't have a user area associated with it. *init* removes a zombie process from the process table when its parent exits.

- save the accumulated kernel and user mode execution times of the process in the process table

- change the parent of any remaining child processes to process 1(the *init* process)

- sends a "death of child" signal to the parent process so process 1 can remove it from the process table.

- resume scheduler which chooses next process to run and performs a context switch

Objects

System V/MLS supports the following objects:

> regular files
> special files
> directories
> named pipes
> unnamed pipes
> shared memory segments
> message queues
> semaphores
> processes

This report also discusses 630 MTG buffers which are system objects (as are the process table, u-area, etc). 630 MTG buffers are emphasized due to their unique nature.

Of these objects, all are represented as part of the file system except for message queues, semaphores, shared memory segments, processes, and the 630 MTG buffers. All of these objects are named objects subject to the system's discretionary access policy except for unnamed pipes and 630 MTG buffers. The file system object access control information can be found in the inode for that object. The non-file system objects each have storage data structures which contain their access control information.

There are three basic access types allowed in System V/MLS:

> READ -
> > Any operation that results in a flow of information from an object to a subject.

WRITE -
> Any operation that results in a flow of information from a subject to an object or that causes a change of state within an object.

EXECUTE -
> Execute access on a file allows the loading and running of the contents of that file (object). Execute access on a directory allows a subject to search the directory for a filename.

The following section provides a description of each of the object types. These descriptions include usage, design, and implemention. There is some discussion of access policies and features, although details can be found in the discretionary and mandatory access sections of this report (see page 44, "Discretionary Access Control" and page 48, "Mandatory Access Control").

Regular Files

Regular files are the primary information containers for the system. Any data can be placed into a file. Allowable access types for regular files are read, write, and execute.

Directories

A directory is simply a file containing the names of those files which reside in it and their inode numbers. It is possible for a directory to contain an upgraded file or directory. It should be noted that files must be created in a directory at the level of the directory. This is a consequence of the System V/MLS "write equal only" policy. Therefore, the only way for a directory to contain an upgraded file, is for the file's classification to be upgraded using the *chpriv* command. The upgraded file name or directory name must be at the same label as the containing directory; thus making it visible at the label of the containing directory. The attributes, other than file name and inode number, are at the upgraded label and are therefore only visible at that upgraded, or higher label. Some directories can be marked as SECURED, and are treated differently by the TCB (see page 58, "SECURED Directories").

Under normal circumstances the TCB ensures that a directory's label dominates the label of its parent directory. Thus, as one traverses a path from the root to a file system object, the labels are monotonically non-decreasing. It is possible, however, for a trusted user to downgrade a file or directory which could leave that part of the file system non-monotonically non-decreasing. This happens when a file has been reclassified (via *chpriv*) such that the containing directory's label is greater than the file's new label (see page 57, "Reclassifying Information"). Although the file has been reclassified, it is still protected at the (higher) label of the containing directory. This is because a process must be operating at the label of the containing directory in order to traverse the path to the reclassified file and read it. However this process would not be able to write the file. To restore the normal file system hierarchy, the file would have to be moved via the *mvpriv* trusted process. *mvpriv* ensures that the invoker has discretionary search and write access to the containing directory and the target directory and that the invoker is operating at the label of the containing directory. Finally *mvpriv* ensures that the target directory's label is equal to the file's new label. Allowable access types for directories are read, write, and execute.

Special Files

Special files are file system objects which are used to represent devices and can be manipulated by user processes. For special files that represent single-level devices, the label of the special file as recorded by the file system is used. Storage medium devices are multi-level devices; the 630 MTG terminal is a multi-level device represented by multiple single-level pseudo devices. Access to multi-level device files is restricted to trusted processes that enforce the labeling requirements.

User terminals, except for the 630 MTG, are single level devices that can operate at only one label at a time. This range has a maximum security label and a minimum security label defined in a device clearances database (the */mls/cleardev* file). The maximum security label must dominate the minimum security label. No device is ever allowed to operate outside the range of security labels specified by its maximum and minimum security labels. Allowable access types on special files are read and write; execute access does not have any effect on special files.

Named Pipes

Named pipes are file system objects used as communication buffers between two processes. Named pipes provide an interprocess communication facility that manages data in a first-in, first-out manner. A process granted read access to a named pipe may extract the oldest information in the named pipe. Extracting the information deletes it from the named pipe. A process granted write access to a named pipe can append information to the named pipe. Allowable access types for named pipes are read and write; execute access does not have any effect on named pipes.

Unnamed pipes

Unnamed pipes provide an interprocess communication facility like named pipes. Unnamed pipes do not have names in the file system; however, they are represented in the file system with inodes. Unnamed pipes can be used as communication buffers between a child process and its parent process and between sibling processes. Since this communication can only occur if the pipe's file descriptor is passed on to the child from the parent as part of the process context duplicated when the child is forked, no other access control is enforced except in the case of executing a *newpriv* command. For an explanation of the *newpriv* command, see page 48, "Labels on System V/MLS". Allowable access types for unnamed pipes are read and write; execute access does not have any effect on unnamed pipes.

System V IPC Objects

There are three IPC mechanisms in System V/MLS known as System VIPC: messages, semaphores and shared memory segments. Although each is intended for a specific use, they all share common implementation properties. These properties are as follows:

> -A kernel resident table exists, one per mechanism, which contains entries describing all instances of the mechanism. For messages there is a system message table whose entries describe all current IPC messages, for semaphores there is a system semaphore table whose entries describe all instances of semaphores, and for shared memory segments

there is a system shared memory table whose entries describe all instances of shared memory segments.

-Each entry in one of the three system tables contains the following information:

-numeric key (user-chosen name)

-creator UID/GID (IDs from the creating process)

-owner UID/GID (IDs originally the same as those of the creator, but may be changed by the creator or owner)

-set of permission bits for user, group, others (see page 44, "Discretionary Access Control")

-status information (e.g., last process to update the entry, time of last access, number of processes attached)

-Each type of mechanism is associated with a corresponding "get" system call to create a new entry or retrieve an existing one (i.e., *msgget, semget, shmget*). A process supplies a user-chosen key to the call. The kernel searches the appropriate system table to see if an entry exists for the given key. The table is searched on the key field which is contained in each entry of the table. If no entry exists the kernel allocates a new structure, initializes it, and returns an identifier (ID) to the user. If an entry exists for the given key, the kernel checks permissions for the entry and if access is allowed for the requesting process, the identifier for the entry is returned. A creating process can be assured of obtaining an unused entry by specifying the key "IPC_PRIVATE" to the call.

-For each mechanism, the identifier returned from a "get" system call is based on the index into the table for the data structure (index = identifier modulo (number of entries in table)). When a process removes an entry, the kernel increments the identifier associated with it by the number of entries in the table. Processes that attempt to access the entry by its old identifier fail on their access.

-Each mechanism is associated with a corresponding set of "operation" system calls: *msgsnd, msgrcv,* for messages, *semop* for semaphores, *shmat, shmdt,* for shared memory segments. For all of these calls, a process must specify an appropriate identifier to the call (i.e., the ID returned from the "get" system call). The kernel then checks that the invoking process has appropriate access to the corresponding entry, (see page 46, "DAC on System V IPC Objects"). Note that this prevents a process from successfully gaining access to an entry by guessing at the entry ID.

-Each mechanism is associated with a corresponding "control" system call: *msgctl, semctl, shmctl*. These calls allow a process to query status information about an entry, set status information, or remove an entry from the system. A process must have read access to the entry to query status information. In order to set status information or

remove an entry, however, the process UID must match the creator UID or the owner UID. Creator UID and GID fields can never be changed, so the creating user always retains "control" access to the entry. Since the owner can change permission bits, rw access to the entry can be taken away from the creator, however, because the creator will still retain "control" access, he can always give himself back rw access.

The following three sections describe each of the IPC objects.

Semaphores

Semaphores are objects that are used to implement a process synchronization mechanism. System V semaphores are a generalization of the P and V operations described by Dijkstra[1] in that several operations can be done simultaneously, and increment and decrement operations can be by values greater than one. System V semaphores can therefore take on any numeric value and access to a resource (i.e., locking/unlocking) or to share a small value between processes. Allowable access types for semaphores are read and write; execute access does not have any effect on semaphores.

Message Queues

Message queues are containers for messages which are primarily used to hold requests to server processes. A process granted read access to a message queue may extract the oldest message from that message queue. Extracting the message deletes it. A process granted write acce to a message queue may append messages to the message queue. Allowable access types for message queues are read and write; execute access does not have any effect on message queues.

Shared Memory Segments

Shared memory segments are used to allow multiple processes to access the same information without the overhead of multiple copies. A description of how shared memory is implemented can be found on page 24, "Memory Allocation". Shared memory provides the fastest means of exchanging data between two processes. Allowable access types for shared memory segments are read and write; execute access does not have any effect on shared memory segments.

---

1   Dijkstras, E. W., "Cooperating Sequential Processes", in *Programming Languages*, ed. F., Genuys, Acdemic Press, New York, NY, 1968

Processes

Although processes are the subjects in the System V/MLS system, when they are viewed as the recipient of a signal they must be treated as objects. Processes are represented by entries in the process table.

When a program overlays a process via *exec*, the saved UID is set equal to the effective UID; the real UID remains as it was. This way, a process can move back and forth between the saved UID and real UID. However, if the effective UID of a process is 0 (i.e., superuser), invoking the *setuid* system call will set all three UIDs to the new value. Therefore a process cannot reclaim root permissions once it has given them up.

In order for a process to be allowed to send a signal to another process, the real or effective UID of the sending process must match the saved or effective UID of the receiving process, unless the effective UID of the sending process is superuser.   Also, the classification level of the sending process must match the classification level of the receiving process, unless the effective UID of the sending process is superuser. For more details on signals, see page 36, "Signaling".

630 MTG Window Buffers

The window buffers on the 630 MTG are associated with the process that is currently running on the terminal. These storage objects are discussed further in the section on page 73, "The 630 MTG Terminal Implementation".

TCB Protection Mechanisms

The System V/MLS TCB protects itself and users' data through the use of hardware and software protection mechanisms. The following sections discuss these protection mechanisms.

Hardware Protection Mechanisms

System V/MLS utilizes the system hardware architecture to provide for a trusted computing environment. Hardware elements that are essential in providing this environment include privileged execution modes and the virtual system environment provided to users. These two concepts, along with the special controls afforded to the 630 MTG terminal provide a protected user environment.

Hardware Protection

The System V/MLS hardware is based upon the WE 32100 Central Processing Unit, in conjunction with the WE 32101 Memory Management Unit. This combination provides an environment for users which enforces restrictions on the ability of users to access data. The data access restrictions are provided by three mechanisms: access determined by processor execution mode, controlled access within a given execution mode, and kernel privileged instructions.

The first control mechanism involves the association of access permissions with physical memory segments. For every physical segment, four access fields are specified, one for kernel, executive, supervisor, and user execution modes. These access fields are checked when access is requested to a segment. If the processor is in kernel mode when it requests access, the access decision is based upon the kernel permission field. When executing in user mode, the user permission field is used. The specifiable permissions to a physical page are; read/write/execute (RWE), read/execute (RE), execute only (EO), and no access (NA).

The second control mechanism involves the protection of data between users at the same execution mode. The system provides a virtual-to-physical translation mechanism, which isolates the data accessible to users. The translation tables needed for this mechanism are controlled by the TCB, and protected from unauthorized modification. This protection involves placing the tables into memory that is only accessible while the processor is in kernel mode, and restricting the ability to enter kernel mode.

This ensures that only the TCB will be capable of modifying the tables which define the virtual system environment. See page 24, "Memory Allocation", for a detailed discussion of the translation mechanism.

The third control mechanism consists of the set of kernel privileged instructions. These kernel privileged instructions, which may manipulate machine resources (e.g., physical memory, hardware interrupts), are restricted to processes operating in kernel mode. Since only the TCB operates in kernel mode, non-privileged users are prevented from using these instructions. For more information on kernel privileged instructions, see page 5, "Instruction Set".

Software Protection Mechanisms

As discussed previously (see page 22, "TCB Boundary"), the software portion of the TCB is composed of those routines running in kernel space and trusted processes. The TCB is capable of protecting the TCB routines and data files from unauthorized modification through the routine enforcement of the mandatory and discretionary access control policy.

Discretionary Access Control

Discretionary Access Control (DAC) allows owners of objects to grant or deny access to the named objects which they own based upon user-defined information sharing requirements. System V/MLS named objects are identified on page 38, "Objects". DAC in SystemV/MLS is provided through protection bits. Protection bits are associated with all System V/MLS named objects except processes. Access mediation procedures described below refer to discretionary access control only. Of course mandatory access control decisions always override discretionary access control decisions.

DAC on FileSystem Objects

The protection bits on filesystem objects are used to set access for an owner, a group, or all others. The creator of an object is the owner and at object creation, each object is marked with the owner's UID and GID. After object creation, however, the creator may transfer ownership to another user or group by using the *chown, chgrp* or *chpriv* commands, or the *chown* system call. The previous owner then loses all ownership access rights. A file's security label cannot be changed if the file is being held open by another process and the user's RUID is not root.

Once a subject has been granted access to an object, the subject retains the access until the subject destroys or releases the object. This is true if the object's owner changes the permissions such that the subject should no longer be capable of accessing the object, or even if the owner attempts to delete the object, any subject which has the object open will retain access.The only exception to this occurs with the character special file */dev/tty* (i.e. a user terminal). */dev/tty* has access checks performed with each read or write call; thus, an access permission change to the actual terminal associated with */dev/tty* will be reflected with the next access attempt.

Each object is also associated with a specific "privilege", which is System V/MLS terminology for a tuple consisting of an instantiation of a discretionary access control group at a given mandatory access control label. The "discretionary group" associated with the object corresponds to the traditional UNIX group mechanism. All members of the discretionary group have discretionary access permission to the object based upon the setting of the group field of the protection bits. Any named object inherits its privilege, and thus its discretionary group, from the process which created it; however, the discretionary group of a named object may be changed via the *chgrp* or *chpriv* commands (these commands must be executed by the owner of the object). There is no limit to the number of groups to which a user may belong; however, a user may only operate with one group's identifier in effect. Thus, a user may not have discretionary access to a file in group A while the user is operating as a user in group B. To obtain access to the group A file, the user may change the operating discretionary group of his or her process, and thus obtain discretionary access rights to the object, by using the *newgrp* or *newpriv* commands. In SystemV/MLS a group can be created by any user; however, groups with special access rights are created and owned by the system administrator. For a complete description of these special groups see page 54, "Special User Authorizations on System V/MLS".

Access granted to "other" permits all other users of the system discretionary access to the object.

Each file is represented in the system by an inode, which contains the owner and group ID of the file. The owner's UID and the group's GID are recorded in the inode at file creation. Also in the inode are eleven security relevant bits associated with the file; nine for the three sets of access control, and one each for the SUID and SGID bits. For a description of SUID and SGID see page 47, "Setuid/Setgid Mechanisms".

Discretionary access to a file-system object is checked in the following manner:

Before making any other check, the kernel ascertains whether the effective UID of the process requesting the access is zero (the superuser). If so, then access is granted immediately.

1. The effective user ID (EUID) of the process is checked against the owner ID of the file. If the EUID matches the owner ID of the file, access permission is checked for the owner. If the requested permission bit for the owner has been set, access is granted. Otherwise access is denied. If the EUID did not match, the check continues.

2. If access has not been determined, the effective discretionary group ID of the process is checked against the discretionary group ID of the object. If the Effective Group ID (EGID) matches the discretionary group ID of the object, access permission is checked for the group. If the requested permission bit for the group has been set, access is granted. Otherwise access is denied. If the EGID did not match, the check continues.

3. If access has not already been determined, the "other" bits are checked. If the requested permission for "other" has been set, access is granted. Otherwise access is denied.

A user may alter the discretionary access control attributes of a file by using the *chmod* command or system call. This command allows the owner of a file to change the protection bits on that file. The *umask* command specifies the default protection bit settings when a file is created. Any bits set to "1" in the *umask* will be cleared in that file's protection bits upon file creation (i.e. the protection bits on the newly created file will be the negation of the *umask* setting). The default umask for the system is --xrwxrwx (177). This results in initial access of rw-------(600), meaning that only the owner has access to the object and all other users have no access.

DAC on System V IPC Objects

Protection bits for user, group, and other are also associated with an IPC object, whether it be a message, semaphore, or shared memory segment. These permission bits are stored in the IPC object's associated system table entry as described on page 40, "System V IPC Objects". The SUID and SGID bits are not meaningful for IPC objects, as IPC objects are not executable. An IPC object has both a creator and an owner associated with it, and at object creation, the creator and the owner of the IPC object are the same. The creator (or the owner) may change the owner UID, owner GID, and the permission bits associated with the object through the IPC "control" system call (i.e., *msgctl*, *semctl*, or *shmctl*). The creator UID and GID can never be changed and therefore the creator always retains "control" access to the object.

Since each IPC object has associated with it both a creator UID and GID, and an owner UID and GID, there is an additional check made when determining discretionary access. Access to an IPC object is checked in the following manner:

> If the effective UID, EUID, of the process is root, access is granted.

> 1. The EUID of the process is checked against the creator UID and the owner UID of the object. If either matches, and the requested permission bit for the creator or owner has been set, access is granted. Otherwise access is denied. If the EUID did not match, the check continues.

2. The EGID of the process is checked against the creator GID and the owner GID of the object. If either matches, and the requested permission for the group has been set, access is granted. Otherwise access is denied. If the EGID did not match, the check continues.

3. If access has not yet been determined, the "other" bits are checked. If the requested permission for other has been set, access is granted. Otherwise access is denied.

As mentioned previously, the creator or owner of an IPC object can change the owner GID. The owner GID, however, can only be changed to a privilege at the same sensitivity level as the original owner GID. If a process attempts to change the owner GID to that of a privilege at a different level, the resulting GID will be the discretionary group of the requested GID and the current level of the IPC object. A check is made to ensure that such a resulting privilege exists on the system. MAC checks for IPC objects are only performed against the owner GID.

Setuid/Setgid Mechanisms

The high order two protection bits of a file are the set-user-ID (SUID) and the set-group-ID (SGID) bits. These bits have no meaning unless the file is an executable program. When the SUID bit has been turned on (via the *chmod* command or system call) and the program is later executed, the effective UID is copied to the saved UID as in a normal *exec*, and the UID of the executed program becomes the effective UID of the resulting process. When the SGID bit has been turned on and the program is later executed, the resulting process is given a GID which reflects the sensitivity level of the invoking process and the discretionary group of the program, provided that the resulting privilege is defined on the system. The process now has all the discretionary access rights of the owner of the program (and/or the owner's group) but the actions of the process are controlled by the program. The process can change its effective UID to either the "real" UID (the UID of the user who invoked the process) or the "saved" UID (the UID of the owner of the setuid program), resulting in the process having the sum of both users' discretionary access rights. If the effective UID is 0 (superuser), changing the effective UID is irreversible since the EUID, RUID, and saved UID all get changed to the UID of the program.

System V/MLS provides protection against some misuses of the SUID mechanism. If the ownership of a file is changed, the SUID/SGID bits are cleared. Also, if the file's group is changed, the SUID/SGID bits are cleared. If a file is modified by any user other than the owner, the SUID/SGID bits are cleared. Upon execution of a SGID file, the exec system call sets the effective GID to a GID that preserves the security label associated with the subject. SUID attacks against superuser are more difficult because no process can execute with superuser privilege unless the executed code has been labeled at SYSTEM level. Untrusted users are never cleared to SYSTEM level and hence cannot create a file executable by superuser. The System V/MLS */bin/sh* command interpreter does not allow SUID/SGID privileges to be inherited by processes spawned from SUID/SGID processes via shell commands.

It is still possible, however, for the DAC policy to be violated if users fail to write well-behaved SUID/SGID programs. SystemV/MLS addresses this problem by providing a configurable option to deny the setting of the SUID and SGID bits on files by anyone other than superuser. This allows

the administrator to inspect any candidate SUID/SGID programs proposed by users or hidden in applications before they are installed.

Mandatory Access Control

Mandatory security is enforced by System V/MLS over all subjects and objects. Subjects and objects are labeled as described on page 48, "Labels on System V/MLS", and these labels are used to enforce the mandatory security policy. Labels are assigned and maintained by the TCB, and may only be modified through trusted software.

The mandatory security policy enforced by System V/MLS relies upon two basic relationships between labels. These relationships are:

Dominance:

When label X dominates label Y, the hierarchical portion of label X is greater than or equal to the hierarchical portion of label Y, and label X contains at least all of the non-hierarchical categories that are contained in label Y. This check is performed within the kernel by the routine *mls_dom*.

Equivalence:

When label X is equivalent to label Y, the hierarchical portion of label X is identical to the hierarchical portion of label Y, and the set of non-hierarchical categories contained in label X is identical to the set of non-hierarchical categories contained in label Y. This check is performed within the kernel by the routine *mls_equ*.

System V/MLS supports three basic access modes to objects: read, write, and execute. The mandatory security policy supported by System V/MLS controls the basic access modes such that data is not compromised to unauthorized users in accordance with the following controls:

> To grant a subject read access, the label of the subject must dominate the label of the object.

> To grant a subject write access, the label of the subject must be identical to the label of the object.

> To grant a subject execute access, the label of the subject must dominate the label of the file object.

Labels on System V/MLS

System V/MLS uses the UNIX group ID (GID) to implement its labeling scheme. GIDs are associated with each subject through the process table and with each object as part of the inode or IPC data structure. When an object is created, its label is that of the creating process. A child process inherits the label of its parent process. At login, the process receives a default label according to

what is specified in the /mls/passwd file. For a complete description of how this works, see page 62, "User Identification and Authentication".

Every label has two parts; a hierarchical level and a set of non-hierarchical categories. The hierarchical level is represented by a number from 0 to 255. All levels are defined by the system administrator except for level 0. Level 0 is the lowest level on the system and is a special level reserved for system use. It is given the name "SYSTEM". A label may contain from 0 to 1024 categories. Relationships between labels are described elsewhere in this report (see page 94, "Mandatory Access Control").

Privileges

System V/MLS also incorporates a unique concept referred to as a"privilege". A privilege is the name given to the combination of a group and a level and is used as an easy way to move between level/group combinations. A privilege can be thought of as an instance of a group at a given level. It is possible to have separate privileges which are instantiations of a group at different levels, and each group, when instantiated at a given level, is represented as a distinct privilege. Take, for example, a discretionary group "projA" that may operate at several mandatory levels. There would be one privilege made for each level the group projA may operate with. The creation of privileges is constrained by the clearance of the user creating the privilege as well as by the set of levels currently defined on the system. Privileges are then used for both mandatory access decisions and discretionary group access decisions.

Each privilege has a set of members that may operate with that privilege. The creator of a privilege is the owner of that privilege. Only the owner may add users to the privilege; that can be done with the *addgrp* or *addpriv* command. *addgrp* will add a member to all privileges defined with that group provided the user to be added has the required clearance. *addpriv* will add a member to a particular privilege if the member is authorized for the label associated with the privilege (see page 71, "addgrp and *addpriv* trusted processes" for a more complete description of these commands).

When a subject creates an object, the current operating privilege of the subject is assigned to the object. Users may change the privilege associated with their files using the *chpriv* command. Changing the label associated with a file to a level that is not dominated by the original is a downgrade. Reclassification policy determines who, exactly, has the capability to downgrade aspecific file under given circumstances. The five possible reclassification policies are described in section 7.5 of the *Trusted Facility Manual for System V/MLS* and range from users operating as root to members of a special discretionary group named *secadm* (see page 57, "Reclassifying Information") to all users of the system.

When invoking SGID files, the process inherits its security label from the user executing the program and its discretionary group from the file. In other words, the invocation of the SGID program can not change the current operating level of the process. A check is then made to make sure that the resulting privilege is defined in the system.

The GID associated with System V/MLS objects is 16 bits long and is used to indirectly reference a privilege. Privileges are stored in the */mls/labels* file as machine dependent data structures that are immediately usable by the kernel without format conversions. Machine dependent representations are converted into machine independent canonical form whenever they are used outside the kernel. In the canonical form of a security label, level and categories are represented by numbers. These numbers are expanded as per the */mls/levels* and */mls/categories* files before being displayed in human readable form. For all terminals the */mls/cleardev* file allows the storage of additional information such as device maximum and minimum labels. These device maximum and minimum labels can then be used to restrict the allowable levels of information to be stored on or retrieved from the device.

The integer value assigned to a newly created privilege is determined by incrementing the current maximum privilege value by one. When the system capacity (60,000) has been used the system administrator is notified and determines the appropriate action to follow. Possible choices include the failure to add the new privilege, reuse a previously defined (but no longer in use) privilege, or 'retire' a currently active privilege and then utilize the retired privilege number. As recommended in the TFM, the reuse of a previously defined GID should occur no sooner than one year after the privilege was retired. This waiting period minimizes the potential for the GID to still exist on the system, or for the GID to exist on a system backup that might potentially be reloaded onto the system. If, in fact, a file with a GID of a removed privilege is loaded into the system and the same GID has been used for a new privilege, the file will be associated with the new privilege. This is the reason that it is important to have a long waiting period before a privilege GID is reused, and why the system administrator must be cautious when restoring files from backup tapes.

The */mls/labels* file is most often accessed with the GID as a search key. The kernel uses the subject's and object's GIDs to index into the */mls/labels* file and find the associated security labels when checking MAC access and to find the associated discretionary group ID (DGID) when checking DAC access. Similarly, when information is displayed in a human-readable form (e.g. printed output), the TCB translates the internal privilege information into a mandatory security label, which is output with the data.

The */mls/labels* file is a regular file containing data structures that map privileges to labels. Each privilege in the file contains a discretionary group plus the privilege's security label. The */mls/labels* file is accessible by the kernel or a trusted process which is SUID to root. New privileges are always added to the end of the */mls/labels* file and a check is made to ensure there is not an already existing privilege with the same GID.

The following is a description of the format and fields of the */mls/labels* and */mls/group* files. The description of the */mls/labels* file is a logical description only; internally it is stored on disk in a binary format.

/mls/labels :   <GID> : <DGID> : <LABEL> : <LABEL NAME> : <RESERVED>

          GID: Privilege ID

          DGID: Discretionary Group ID

Security Label: a hierarchical level and list of non-hierarchical categories

Security Label Name: machine generated unique name for label used to name the subdirectory of a SECURED directory associated with this label

Reserved: reserved for other attributes; network labels, ACLS

/mls/group: <NAME> : <INFO> : <GID> : <MEMBERS>

NAME: This is the name by which the privilege is known. This is the name used with *newpriv* or *newgrp* commands to designate a privilege or group

INFO: Contains various special key words used as flags or indicators

GID: Privilege ID.

MEMBERS: A comma separated list of login names of users who are authorized to use this privilege/group. (The first name in this list of privileged members is the owner of the privilege)

The GID and DGID are unsigned short integers in the range 0 through 60000. It is important to note that the GID and DGID identifiers share the same name/number space represented by entries in the */mls/group* file. When the *mkgrp* command is executed, a GID is assigned for each DGID by adding a */mls/labels* file entry with GID = DGID and the label set to SYSTEM. When a new privilege is created with the *mkpriv* command, entries are made in both the */mls/labels* file and the */mls/group* file. The new privilege receives a new, unique GID, and its DGID is the same as that of the underlying group.

The */mls/labels* file contains entries for all privileges in use on the system. It is a binary data file organized so entries can be quickly located by kernel routines using the GID as an index. The */mls/labels* file contains a hash table of fixed size. Slots are pointed to by a hash of the privilege's GID. Collisions in the hash table are resolved by chaining (i.e., the entry occupying the slot in the hash table points to the next entry outside the hash table whose GID hashes to the same hash index). Also, all */mls/labels* entries with the same discretionary access control group (DGID) are chained together. For a pictorial description of how the entries of the */mls/labels* file are linked together see Figure 6.

The privilege data portion of each */mls/labels* entry is a structure with the following fields: GID, DGID, hidden subdirectory name for this privilege, hierarchical level of the privilege, the number of significant category words, and 32 category words. Privileges will usually require many fewer than 32 32-bit words to represent their category set. Therefore, indicating the number of category words that are significant greatly reduces the processing time when working with categories. For a pictorial description of the fields of the privilege data portion of the */mls/labels* entries, see Figure 7.

Labels File Structure

d_dgrp

d_hash

Privilege Data

50 Entry
Hash Table
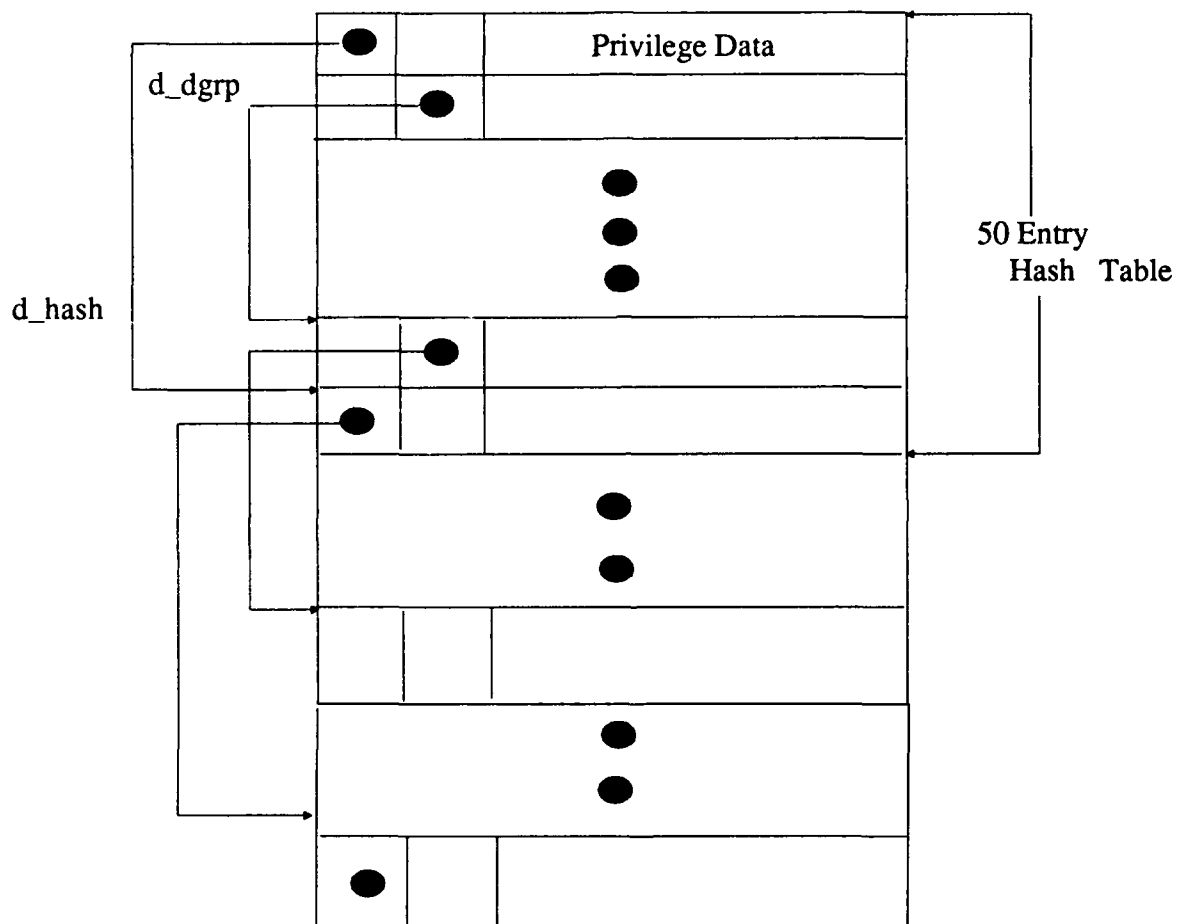
d_hash - points to next entry which
hashes to same slot

d_dgrp - points to the next entry which
has the same dgid

Figure 6. Labels File Structure

Privilege Data Structure

gid
dgid
Name
Level
Significant Category Words
Category Word 1
●   ●   ●
Category Word 32

Figure 7. Privilege Data Structure

Changing Subject Sensitivity Label Interactively

System V/MLS incorporates the ability for any user to alter the mandatory access control level and discretionary group associated with their session. This mechanism is invoked via the *newpriv* command, and may only be used to change to a MAC level which dominates the previous level. A user executes a *newpriv* command, specifying the desired privilege as an argument. The user must be a member of the selected privilege. The System V/MLS TCB then compares this privilege with the session's clearances stored in the */mls/sessions* database. Should any of these checks fail, the *newpriv* command fails and returns an error message and the user remains at the current privilege. If these checks succeed, then *newpriv* creates a new child process and invokes a new shell for the user at the new level. Additionally, upon execution of the *newpriv* command, all file descriptors of the parent process are checked to ensure that no violation of the system security policy can occur. If the new level dominates the current level, all the file descriptors are closed. The user may then operate at the new level for as long as is desired, and terminate that session by exiting from the shell. This will release the new shell only, and the user will revert to the shell in which he or she was operating before issuing the *newpriv* command. This ensures that no information may be passed in violation of the System V/MLS security policy.

Special User Authorizations on System V/MLS

System V/MLS makes use of a number of different discretionary access control groups, as well as specially defined mandatory access control labels, in order to maintain control over different elements of the TCB. Each of these, as well as the role it plays in the overall security of the system, is discussed below.

Operational Roles

In System V/MLS there are three roles associated with performing administrative duties on the system: Operator, System Administrator, and Security Administrator. A user associated with one or more of these roles is known as a system officer. The duties involved in these roles are performed with superuser (i.e., root) permission. System V/MLS relies on additional security measures to ensure that only authorized personnel are permitted to operate as system officers. It should be noted that these roles can only be enforced procedurally. The system does not enforce any kind of separation or least privilege amongst roles.

The operator's duties deal primarily with the mechanics of running a computer system. They include the following:

-system start-up/shutdown

-mounting, unmounting, and storage of labeled data

-backing up and restoring files

-distribution of labeled hardcopy

-ensuring object reuse requirements on removable media

The system administrator's duties deal primarily with configuring the system, and detecting and correcting abnormal conditions. They include the following:

-setting the system time/date

-managing user accounts

-installing/removing application software

-maintaining correct permissions

-granting special authorizations

The security administrator's duties deal with maintaining the security of the system. They include the following:

-administering clearance information

-printing, creating and editing the */mls/levels* and */mls/categories* files

-maintaining device clearances

-setting up secured directories

-reclassifying information

-importing and exporting information

-changing default protection

-configuring audit trail channels

-reviewing audit trail data

-assuring the integrity of the TCB

-installing System V/MLS

-uninstalling System V/MLS

In order to provide a more secure environment for the system officers, System V/MLS incorporates the following changes to the standard superuser environment:

-No one may login with the user-ID of root. System officers must first login as themselves and then *su* to root. In this way, all auditable actions performed by root can be traced back to the individual user.

-The ability to *su* to root may only be executed on devices that have a clearance at the SYSTEM level. This allows the capability to restrict system administrative actions to terminal devices that can be physically protected.

-System V/MLS is delivered with a specified default search path for program execution by system officers.

-System V/MLS contains a security enhanced version of the Bourne shell for use by the system officers.

-System officers, while operating as root, can not execute any program unless that program is labeled at the SYSTEM label (i.e.only trusted programs).

System Software Integrity

In order to help preserve the integrity of the system software, System V/MLS incorporates a simple integrity mechanism: use of the mandatory access control policy to assure that untrusted processes cannot write to TCB code and data. All system software (the kernel, trusted processes, and any other software which the site chooses to protect) is given the mandatory access control label "SYSTEM". SYSTEM is defined to be MAC hierarchical level zero and is one hierarchical level below the lowest mandatory level which is accessible by nonprivileged users. When a new user is added to the system, the user is given a minimum and maximum hierarchical level at which he or she can log on. In order to modify a SYSTEM level object, the user must either be running a trusted process which has root authorizations, or be logged in at the SYSTEM level. It is intended that only a very few terminals and a few trusted users on any System V/MLS system will be given a minimum clearance level of SYSTEM. This prevents modification of TCB files by a mandatory mechanism. Most files which contain executable TCB code are owned by the discretionary group *bin*. Most files which contain non-executable TCB files are owned by the discretionary group *sys*. Membership in these groups is restricted to trusted users.

System V/MLS defines a system high mandatory access control label in addition to the system low label, SYSTEM, described above. This label, called SYSHI, is defined as the highest hierarchical mandatory access control level, with the complete set of non-hierarchical categories defined for the site. SYSHI is initialized when the system switches from single to multi-user mode, and remains constant while the system is in multi-user mode. It is for this reason that new categories and levels should only be added to the system when it is in single-user mode.

Two special groups exist directly in support of system services: *lp* and *mail*. These groups are used for discretionary access control purposes, in order to allow several trusted processes which may execute with that group ID to share resources while making those resources inaccessible to untrusted users. The specific applications of these groups may be found on page 71, "/usr/lib/lpadmin", and page 69, "/bin/mail".

Reclassifying Information

System V/MLS allows users to change the privilege associated with files and named pipes, provided that the change is to a privilege which is currently defined on the system and for which the user is cleared. During the reclassification process a file can be upgraded, downgraded, or the label may stay the same, in which case the new privilege has a different discretionary group, but the same hierarchical level and set of non-hierarchical categories. When a file is upgraded, the new label dominates the old label, and when a file is downgraded, the new label does not dominate the old label. System V/MLS allows designated users, i. e. users who are allowed to downgrade, to reclassify to privileges with non-compatible labels.

System V/MLS supports five different reclassification policies. They are listed below in order from most restrictive to least restrictive:

1. System administrators are the only users allowed to reclassify objects either upward or downward.

2. System administrators and members of the *secadm* group are the only users allowed to reclassify objects either upward or downward.

3. System administrators are the only users allowed to reclassify objects either upward or downward, and all users can reclassify objects upward.

4. System administrators and members of the *secadm* group are the only users allowed to reclassify objects either upward or downward, and all users can reclassify objects upward.

5. System administrators and all users are allowed to reclassify objects either upward or downward.

System V/MLS is delivered to the customer with the third policy in effect. It is up to the site manager or accrediting authority to determine whether a more or less restrictive policy is appropriate for their site/application, taking into account the risks and benefits of the alternate policy.

System administrators, when operating with root capability, may reclassify any file on the system from any privilege to any other privilege provided the new privilege is defined on the system. All other users, members of the *secadm* group included, are subject to the following restrictions when they reclassify information:

1. They must own the object being reclassified.

2. The object being reclassified must be dominated by the label of the currently operating process.

3. They can not affect the classification of objects that are above their maximum clearance.

4. They can only do object reclassification on objects that are within their operating classification range.

The reclassification of objects is accomplished by using the *chpriv* command. One can specify the privilege name or the label and discretionary group which make up the privilege. For the reclassification to be successful, the restrictions described above must be satisfied. If ordinary users are not given the reclassification capability on a given system and a *secadm* group exists on the system, the user requesting reclassification must invoke the *chown* command to give ownership of the object to a member of the *secadm* group to do the reclassification. Once the reclassification is accomplished, the *secadm* member must return ownership to the original user. *chpriv* requires confirmation of the operation if any user downgrades a file, or if the user, while operating at a higher level than the file, changes either the label or the discretionary group of the file. It should be noted that a file's security label can not be changed if the file is being held open by another process and the user's real UID is not root. Also, only root may change the label of a character special device.

After a file has been reclassified to a lower classification, users at that lower classification will still not be able to access the file if it resides in a directory at a higher level. Therefore, to complete the reclassification process, the file will have to be moved to an existing directory that has a classification the same as the newly reclassified file. This can be done using the *mvpriv* command. The usual *mv* command does not allow this operation because of the System V/MLS MAC policy of "write equal only". For the *mvpriv* command to succeed: 1) the user must have discretionary search and write access to the directory containing the file and the target directory, 2) the user must be operating at the level of the containing directory, and 3) the file's classification must be identical to that of the target directory. *mvpriv* requires confirmation for each file moved.

After a file has been upgraded to a higher classification, it still resides in the lower level directory. Although the user, while operating at the file's new classification, would have access to it in the lower level directory, it is good practice to keep files of one classification together in a directory of the same classification. Also, while the file is in the lower level directory, it can be deleted by the owner while he or she is operating at the directory's lower level but not when operating at the file's higher level. This situation exists because of the System V/MLS "write equal only" security policy and the fact that creating and deleting files involves writes to the containing directory. In order to move the file to a directory at the same level as the reclassified file, the *mvpriv* command is used In order for the *mvpriv* command to be successful, the three conditions described above must hold true.

SECURED Directories

System V/MLS provides a mechanism to alleviate the problem of multi-level directory structures. The problem is this: certain directories (notably */dev* and */tmp*) must be accessed by all users of the system, whose authorizations run from system low to system high. This access by all users can result in violations of the system's security policy. The developers of System V/MLS refined a

mechanism which effectively solves this problem. This mechanism designates a group ID at the time System V/MLS is installed (by default, GID number 99) and designates that group as SECURED. From then on, any directory which has that GID is treated specially by the TCB. The mechanics are as follows:

Within a SECURED directory are some number of subdirectories. When a user logs in at a given security level, *login* will create subdirectories at that level in the SECURED directories */dev,/tmp*, and */usr/tmp* if they do not currently exist. The same process occurs whenever a user issues a *newpriv* command to move to a new operating level. These subdirectories are created in anticipation that they will be needed, although they may not actually be used. Mail subdirectories are created by explicitactions of the system administrator using the *MailSetup* command. Therefore, the system administrator determines at which security levels mail can be received on the system. These subdirectories contain the files to which non-privileged users have access. When a non-privileged user references a file which apparently resides either in a SECURED directory or in a directory beneath a SECURED directory, the user is never aware that the SECURED directory mechanism is controlling the traversal of the path and ensuring that he or she is given access only to the directory at the correct MAC label. To any non-privileged user, a SECURED directory is entirely transparent. *namei* is the System V routine that has been modified to recognize SECURED directories and skip over them if the effective user ID of the current process is not root or the current GID is not "SECURED". Since all references to files either go through *namei*, or are made after access checks have been made through *namei*, there is no way for the ordinary user to circumvent this mechanism.

SECURED directories may only be created by, and are only visible to, the superuser and users operating in the group "SECURED". As an example of how the mechanism operates, consider the directory */tmp*. A user operating at Level 1 attempts to create a file called foo in */tmp*, which has been marked as a SECURED directory. The TCB recognizes */tmp* as a SECURED directory, and creates the user's file in subdirectory *L1_tmp*, instead. Similarly, a user operating at Level 5 who attempts to create a file called foo in */tmp* may have it placed in L5_tmp. The actual paths to the files would be */tmp/L1_tmp/foo* and */tmp/L5_tmp/foo*. Each of these files would be invisible to the other user because of the difference in their MAC levels, and neither would note any difference in the operation of the filesystem.

System V/MLS is shipped with four SECURED directories. They are */dev, /tmp, /usr/tmp,* and */usr/mail*. The application of each of these SECURED directories is explained below.

/dev is the directory containing all of the device special files. Because the UNIX operating system represents physical I/O devices as just another instantiation of files, this is a convenient method of organizing all devices for easy handling and access control. The System V/MLS approach to handling devices is to populate the */dev* directory itself with device special files, and create links from the appropriate SECURED subdirectory to the device special file as needed. The contents of */dev* are therefore SECURED subdirectories for each mandatory access control label in use and device special files for each device on the system. Because they exist in the actual */dev* directory, which is marked as SECURED, they are unreachable by processes not executing with a real or effective UID of root (or operating with the SECURED group).

/tmp and */usr/tmp* are merely temporary storage directories, to which all users of the system are expected to have access. System V/MLS assures that no information will be disclosed across mandatory access control labels by its use of SECURED directories in these locations.

/usr/mail is an employment of the SECURED directory mechanism to provide an easy way to implement a multi-label secure mail system. The system administrator may define the labels at which he or she wishes to support mail. This is explained in greater detail later (see page 69, "/bin/mail").

Subject/Object Access Decision Process

When a subject attempts to "open" an object for later read/write/execute access, it must pass both mandatory and discretionary access checks. The access check occurs as follows:

The user program executes an *open* system call. The *open* system call invokes the *namei* kernel routine, which determines the location of the requested object. *open* then calls the *s5access* kernel routine for each directory in the pathname as well as the requested object. If search access is denied to any directory in the path, processing terminates and an error is returned. *s5access* invokes *mls_access* to determine if the subject is allowed mandatory access to the requested object. *mls_access* utilizes the routines *mls_dom* and *mls_equ* to determine if mandatory access should be granted. If *mls_access* grants mandatory access to the object, *s5access* continues processing to determine if discretionary access is permitted. Upon completion of the mandatory and discretionary access checks, *s5access* returns to *open*, which generates an audit record of the event (if applicable), and then returns to the user, indicating whether permission was granted or denied.

This basic procedure is followed for most accesses to objects with the following exceptions:

> - ipc access is granted via the routine *ipcaccess,* which invokes *mls_dom* and *mls_equ*. Control information for ipc objects is changed via the routines *ipc_set* and *ipc_rmid.*

> - changing the inode of a file (e.g. changing owner or file status information) can occur via either the *mls_chown* or the *mls_chmod* routine. Both routines call *mls_equ* to determine if the security label of the subject is equal to that of the object being changed.

> - a subject (process) is considered an object when it receives a signal from another subject. Signals are sent via the *kill* system call, which utilizes *mls_kill* to enforce restrictions upon which subjects may signal other subjects, and in turn utilizes *mls_equ* to determine that an equal label relationship between the subjects exists.

> - for all terminal devices, access controls are enforced when the file is opened and for each read/write call to the device. The open is checked as previously described, while the read/write access checks occur within *mls_ttyrdwrerr*, using *mls_dom* and *mls_equ* as appropriate for the access desired.

Auditing

Auditing in System V/MLS is initiated during the transition from single-user to multi-user mode. The command *satstart* creates and initializes the audit trail by invoking *sfsmap* which reads the raw device upon which each file system exists. The system administrator must designate the desired target file(s) for the Security Audit Trail (SAT) to be written to. The maximum size of a SAT file is configurable. The default maximum size is 5000 logical blocks, where a block can either be 512, 1024, or 2048 bytes depending upon the device type.

The audit trail file itself is owned by root, has group root and has file permissions set to rw------- (0600), meaning that only the audit trail daemon and root have read/write access to the file. In addition, the audit trail is protected at SYSHI. If new levels or categories have been added to the system then *satstart* will calculate the new SYSHI and label the audit trail appropriately, when the system is brought into multi-user mode.

There are two ways in System V/MLS for an audit record to be written; either internally through kernel probe points or directly through user-generated commands. Kernel probe points function as follows: for each probe point, there is an associated SAT function which collects all of the necessary information about that particular event into a trace record. A trace record is a binary record containing a header and data block describing the event to be recorded.

The trace records are buffered in the *sattrace* pseudo-device. The *satsave* daemon (running as root) reads the trace device and then writes the trace records to the audit trail file. The *sattrace* device uses a circular buffer of fixed size and uses the kernel routine *copyout* to move characters to the *satsave* daemon. If the audit buffer is full, the process which caused the record to be generated sleeps until there is room in the buffer. After the next read of the trace device, all sleeping processes are awakened and any blocked audit records are written to the buffer.

If no more audit records can be written because the audit trail has become full, the audit trail daemon will bring the system down to single-user mode automatically rather than allow records to be lost. Audit trail records which remain in the buffer when the audit trail daemon is stopped or killed are held in the buffer while the system is in single-user mode. When the system is brought into multi-user mode again, the data is written to the next audit trail file.

If the audit trail daemon cannot write the audit trail because of hardware or software failures, the buffer will eventually become filled and all user processes will block upon executing auditable events. When this happens, the only way to restart the system is to bring it down to firmware mode via the reset button or the power switch. If the reset button is used, the core image of the system can be saved and the buffer can then be analyzed off-line to see what auditable events occurred just before the system crashed.

Another method for generating audit records is by user commands through trusted processes writing directly to the *sattrace* psuedodevice. There are 8 minor devices defined with the *sattrace* device major number. The *sattrace* device with number 0 is read by the *satsave* daemon. Minor devices 1 through 7 provide a user level interface to the audit trail. This User Level Interface (ULI) is used to record auditable events that can only be inferred from kernel level trace records. For example,

the addition of a new user to the password file can be inferred from the exec of an editor followed by a successful open of the password file. Unfortunately, there would be no explicit record of what the administrator did to the password file. However, a probe point in *mkuser* can record this information. For a list of the ULI audit trail probe points see page 97, "Audit". These probe points use the ULI to insert information into the audit trail.

The *sattrace* device has 32 channels, one per specific type of audit trail record. Each of the 32 channels has 8 subchannels which are used to specify details particular to each channel or major auditable event. For example, file access grants are divided into those that involve read and write access; the read vs. write distinction is recorded by writing read grant records to a different subchannel than that used for write grant records. Channels are enabled and disabled with an *ioctl* call to the *sattrace* device. These *ioctl* calls are restricted to the superuser and only set the appropriate channels when the system is brought from single-user to multi-user mode. A *sattrace* device can be opened for r, w, or rw but can not be opened by more than one process at a time.

Anyone granted access to the operator's console when the system is in single-user or firmware mode has the capability to control and modify the TCB. Since the SAT daemon only runs when the system is in multi-user mode, actions taken at this time can not be audited.

Trusted Path

System V/MLS is capable of supporting a trusted communications path between the user and the TCB hardware and software. The trusted path mechanism varies among the different system configurations, but it is invariably reliant upon the trusted process */bin/getty*. *getty* has been rewritten so that when it detects the Data Terminal Ready (DTR) signal after DTR has been inactive, it searches for all processes which have that terminal port open and kills them. *getty* then overlays itself with login, and the user may proceed to log in to the host. To ensure that a trusted path to the host has been established, the user must cause the terminal to allow DTR to drop; this is most effectively accomplished by cycling the terminal's power.

User Identification and Authentication

System V/MLS requires all users, including privileged users, to identify and authenticate themselves before they are allowed to access system resources. Users identify themselves by entering a login ID and authenticate themselves by entering a password which consists of six to eight alphanumeric characters. Identification and authentication information is maintained in files within the */mls* directory. Only system administrators may gain access to this directory; it is protected from write access by the system mandatory access control policy (stored at SYSTEM label), and is protected from read access by the system discretionary access control policy.

System V/MLS removes the sensitive information (e.g. password) from the publicly readable files */etc/passwd* and */etc/group,* to the protected files */mls/passwd* and */mls/group.* These protected files are ASCII files which are referred to as "shadow files".

/mls/passwd contains the following information for each user:

    -login name
    -encrypted password
    -user ID
    -group ID
    -initial working directory
    -initial program to invoke upon login

/mls/group contains the following information for each group:

    -group name
    -state indicator (if marked to be removed) and secadm flag
    -group ID
    -list of all members of the group

At login, a user is assigned a login privilege. The user may specify a privilege as an argument to *login;* if the user and the terminal are both permitted to operate with that privilege, it will become the user's login privilege. If no privilege is specified at login the user will be assigned his minimum allowable privilege, provided that that privilege dominates the terminal's device minimum, and is dominated by the terminal's device maximum. If the user's requested or default privilege is outside the range allowed for the terminal, or if the privilege does not exist, *login* will fail. The minimum and maximum clearances for each terminal device are kept in the */mls/cleardev* file. The minimum and maximum clearances for users are kept in the */mls/clearances* file.

After login, users may change to any defined privilege, given that:

- The label associated with the privilege is within the user's clearance range.

- The label associated with the privilege dominates the user's current level.

- The label is within the range established for the user's login device.

- The user is a member of the privilege that he or she is attempting to change to.

When a user logs in, System V/MLS creates an entry in the */mls/sessions* database. This database is actually a directory, which has an entry (a file) for every terminal device that a user is currently logged into. These files, in turn, contain information about the user logged into the terminal, and the maximum and minimum clearances of that user for that terminal session. This information is used by certain user-level TCB commands (e.g . *newpriv*) in order to ensure that a user may only operate within the constraints defined at login time. The sessions database is also used to maintain a record of terminal devices which may be held open by a user process after a login session has completed. This record is then used by *getty* to find and kill all such processes. This allows *getty* to ensure a trusted path for user logins.

There are two ways for a user to obtain superuser privileges: in the first method, administrators must first login as SYSTEM level users. This level is less than the nonprivileged users' minimum. After an administrator enters the *su* command, he must enter the superuser password in order to acquire administrative capabilities. The second method involves obtaining physical access to the console terminal device when the system is being initialized, and accessing the system in single-user mode. This method may not require the use of a password, depending upon the exact configuration of the system; however, it is expected that the system console and the CPU hardware are restricted to only the most trusted users of the system.

Adding Users

Only a system administrator may set up user accounts on the system. The system administrator uses the *adduser* command to add users to the system. The following information is requested by *adduser*:

- login ID - a string of alphanumeric characters which may be specified by the system administrator. The system generates unique user IDs (numbers), each corresponding to a login ID. Login IDs are recorded in the audit trail for purposes of accountability.

- default privilege

- minimum and maximum clearances

- home directory

- initial program invoked upon login

- user's real name

After the system administrator provides this information, the automatic password generator provides a choice of several passwords (up to five), until the administrator selects one. By default, passwords generated by System V/MLS are between six and eight characters in length, and include two numeric characters. The administrator has the capability to override the password generator and can enter a password of his choice. After the administrator selects a password, *adduser* updates the appropriate files (i.e. */etc/passwd* and */mls/passwd*). Next, the administrator must communicate this password to the user in a secure manner. When the user logs in for the first time the automatic password generator provides a new password for him. Guidance on these procedures are provided in the *System V/MLS Trusted Facility Manual* and the *System V/MLS User's Guide and Reference Manual*. Users may change their passwords in the same fashion as the system administrator; however, they are not allowed to override the password generator.

When the administrator is defining a new user account, he has the option of assigning a minimum and maximum required time period for the user to change his password. The minimum ensures that the user cannot change his password before this time expires, and the maximum forces a user to change his password after this maximum time period expires. This process, known as password

aging, is beneficial in reducing the possibility that a password can be determined and used indefinitely by an intruder.

Deleting Users

In order to remove a user from the system, the system administrator must either remove or change the ownership of all files belonging to that user. To locate these files the administrator uses the *find* command, which has options to remove or change the ownership of the files. *find* recursively descends the directory hierarchy for each pathname provided and searches for the specified files. Next the system administrator uses the *deluser* command to remove the user from the group, password, and clearances files and remove the user's login directory. The administrator is advised in the TFM to use the *rmuser* command which invalidates the user's password rather than completely removing the user from the system. Leaving the user in the passwdfile ensures the user's UID will not be reused. In this case of a removed user, the user's login ID (login name) is not removed from all groups to which he or she belonged (e.g., group file is not modified). Therefore system administrators are advised to not reuse login IDs.

Object Reuse

System V/MLS disallows scavenging of deleted information on the following storage objects: directories, regular files, special files, named pipes, unnamed pipes, memory, shared memory segments, message queues, semaphores, and the 630 MTG buffers. In addition, System V/MLS supports the object reuse requirements for mountable media (cartridge tapes and diskettes) through administrative procedures. These procedures are explained in section 5.6 in the *System V/MLS Trusted Facility Manual.*

File System Objects

The System V/MLS TCB enforces the object reuse requirements (as described below) on the following filesystem objects: directories, regular files, and special files.

Disk Block Allocation

System V/MLS allocates disk blocks in a manner which ensures that object reuse is not possible for each of the previously mentioned filesystem objects. The system organizes and maintains a linked list of available disk blocks. Each link in the list is a disk block that contains an array of free disk block numbers; the last entry in the array is a pointer to the next block in the list.

s5alloc causes the kernel to obtain an available block from the *superblock* list. If this block is the last available block in the *superblock* list, the kernel uses this block as a pointer to fill the *superblock* list with free disk block numbers.

Then the kernel allocates a free disk block and buffer for this block. The kernel routine *clrbuf* causes the kernel to zero this buffer. When the user saves the data, the kernel will copy the contents of the buffer to the associated disk block.

Pipes

Both named and unnamed pipes use kernel buffers to store data.The TCB clears the buffers of any previous data (see Disk Block Allocation). A pipe may only use a maximum of ten buffers at anyone time. Unnamed pipes retrieve their inode and blocks from the pipe device *pipefstyp*, and then are allocated separate read and write open file table entries. Named pipes retrieve their inode and blocks from the filesystem, and then are opened just like a normal file.

Memory-Based Objects

The object reuse requirement is applicable to memory-based objects such as real memory pages and shared memory segments. The object reuse mechanism for these objects is described below.

Memory

System V/MLS utilizes the demand paging mechanism described on page 24, "Memory Allocation", to enforce the object reuse requirement on memory. When a process requires additional memory,the *growreg* kernel routine determines the number of new pages needed, and initializes the new pages. This initialization involves marking the pages as invalid and setting the "demand zero flag". The demand zero flag indicates that the page should be cleared when the process first references the page(s). When the process references the page, an invalid page exception occurs (since the page had previously been marked as invalid), causing the memory manager to be invoked. The memory manager examines the demand zero flag, and if set, clears the page and marks the page as valid.

Message Queues and Semaphores

When the system is initialized, the TCB designates specific areas of memory for message queues and semaphore maps. These maps (or tables) are areas of memory that contain pointers to message queues and semaphores. The *main* kernel routine initializes both maps (message queues and semaphores) to all zeros. There after, message queues and semaphores are cleared upon deallocation via *msgct'* and *semctl*. When the *msgctl* routine is called with a value of IPC_RMID, the message queue identifier is removed from the system and the appropriate message queue map entry is initialized to zeros. A process which has been sleeping on an IPC message queue which has been deleted is awakened and returned an error. Thus the message queue has been cleared, deallocated, and is ready to be used again. System V/MLS handles object reuse for semaphores in a similar manner. The *semctl* kernel routine sets all semaphores associated with the given semaphore identifier to zero, if IPC_RMID is set.

Shared Memory Segments

Shared memory is accessed (read and written) exactly the same as regular memory, but controlled through IPC system calls. When a shared memory segment is first requested, *growreg* is called to zero out, then allocate the memory to the subject requesting the segment. Thus, steps to eliminate object reuse are performed for shared memory segments immediately before allocation.

630 MTG Buffers

The object reuse requirement is applicable to the 630 MTG buffers and is described in the 630 MTG Intelligent Terminal section of the report (see page 73, "The 630 MTG Terminal Implementation").

System Backup and Restore

File backup and restoration is considered a system administrator duty. Backup is done using either the *sysadm backup* or *cpio* and *find* commands. File restoration occurs with *sysadm restore* or *cpio*.

When a file backup occurs, the GID is written to the backup device along with the file data. The GID is an unsigned short integer, which indirectly references the security label using the */mls/labels* file. Backup tapes are physically labeled by the system administrator with sticky labels representing the sensitivity level of the tape.

File restoration occurs by manually checking all files to be restored against the */mls/group.retired* and */mls/passwd* files to ensure that the UID and GID associated with the file are still valid. Administratively, GIDs should not be reused during what would be considered to be the "normal life" of a backup tape. Files on tapes being restored which are older than this specified reuse interval must be considered to be unlabeled. Tapes containing files without valid labels must be handled correctly by the system administrator. Such files should be considered to be SYSHI, until reviewed by the system administrator. Files restored with invalid ownership information must be assigned to a new owner by the system administrator.

Trusted Processes

A process must be trusted if it is given a privilege which permits it to violate the system security policy. There are different ways in which a process may gain this privilege; these may be broken down into three categories:

> Those processes which are relied upon to actively enforce the system security policy; these programs have intrinsic privilege, regardless of who executes them. Programs which setuid to root, e.g. */bin/ps* or */bin/passwd* fall into this category.

> Those processes which do not have intrinsic capability, and which must be executed by the system administrator or a trusted process to take advantage of that user's privileged status. These are not permitted to be used by nonprivileged users, but exist for the system administrator to use in order to set up or administer the system; e.g. */usr/bin/mkuser* or */usr/bin/rmuser*.

> Those processes which the system administrator (or any nonprivileged user) may execute in the course of day-to-day operation, which must be trusted not to abuse privileges when they are executed by a user who possesses them; e.g. */bin/cat* or */bin/ls*.

October 18, 1989

There are numerous trusted processes in the System V/MLS system. Programs which fall into the first of these three categories (as well as */etc/getty, /bin/login,* and */usr/bin/layers* which are invoked by setuid to root programs and thus run as root) are discussed in some detail below. Programs of the second and third type have been evaluated by the team, and are listed in Appendix C (see page C-1, "Trusted Process List"). The processes described below are listed in roughly alphabetical order, except in those cases in which several similar trusted processes have been grouped together.

/bin/chpriv: This program is a System V/MLS extension of the operating system which may change the group of the object, or its security label, or both, depending upon the given argument string. If the label and group provided as arguments to *chpriv* do not map to a valid privilege, an error is returned. The modified file attributes exist in the object's inode. For further information on the reclassification policies see page 94, "Mandatory Access Control". This command must be setuid to root in order to permit modification of the inode and access to the TCB data files which provide group, label, and privilege information.

/bin/df, /etc/devnm: This is one program (*/etc/devnm* is a link) which provides information about disk devices. When the program is executed by the name */bin/df,* it returns information about space available on disk devices. It has options which allow it to search raw devices and print total blocks used as well as blocks available. When executed by the name */etc/devnm,* it provides a mapping between file system names (such as */usr*) and the actual physical devices upon which they are mounted. Neither program has any security-relevant function, but because of its ability to access raw devices, it must be setuid to root.

/bin/ipcs: This is a utility which provides a report on the status of interprocess communications throughout the system. Under System V/MLS, *ipcs* provides information on all the objects dominated by the user. *ipcs* is capable of reporting information about message queues, shared memory, and semaphores. In order to obtain this information, it looks directly into */dev/kmem,* and therefore must be setuid to root.

/bin/labels: This is a utility program which formats and prints any of the security labels defined to the system in a human-readable format. This command is used to determine the security label of any file to which an invoker has mandatory access, the invoker's current operating security label, the security label associated with any given privilege, or the discretionary group of an object. In addition, this command is also used to construct labels from given level and category names. The command parses its arguments to determine what information is to be returned; it then accesses the TCB datafiles */mls/labels, /mls/clearances, /mls/levels,* and */mls/categories* in order to gather the information it needs to return to the user. The command verifies that the user is authorized to see the information, and will print an error message if the user is unauthorized to see the data requested. Because of its need to access TCB data files, labels must be setuid to root.

/bin/login: This is the program which identifies and authenticates users on the system. *login* has several functions:

> - read the user's password and compare it against the encrypted password stored in */mls/passwd;*

- compare the user's requested or default login security label with entries in the */mls/group* and */mls/clearances* files in order to verify that the user is initiating a session within his or her authorized range of security labels.

- check the requested or default login security label against the file */mls/cleardev*, which contains device maximum and minimum security labels.

- determine terminal type from the entry in */mls/cleardev.*

- determine (again from information in */mls/cleardev*) what if any special port handling programs may be required for this device (e.g., *630init*). If such a handler is required, then *login* forks a copy of the handler and releases the terminal to its control. If no handler is required, *login* forks a copy of the user's shell, as specified in */mls/passwd*, or */bin/sh* if none is specifed, and releases the terminal to the user.

/bin/mail, /usr/bin/mailx, and */usr/bin/mailcheck*: All of these programs exist in support of the multi-level secure mail system which operates under System V/MLS. *mail* and *mailx* are similar in that they provide a user interface through which mail can be sent to or received from another user. Each of them sets its group ID to *mail* in order to access mail files which are kept in the */usr/mail* directory. The primary differences between the two are features: *mail* is the "standard" Unix System V mailer. It lacks many of the more sophisticated features of *mailx*. *mailx* allows far greater flexibility, and incorporates a number of extensions which facilitate its use as a mailer for use in sending mail across networks (although networks are not included in the evaluated configuration). Traditional UNIX mail systems are quite simple; a directory (normally */usr/mail*) is set up and its permission bits set so that only users operating with the group ID of mail are allowed access. Each user's mail is kept in a separate file in that directory. The individual mail files are owned by individual users. */usr/mail* is a SECURED directory; because of this, it has been possible to implement a multi-level mail facility with virtually no modification to the underlying mail mechanism. Since */bin/mail* was not modified and given root privilege, read-down of mail is not possible. The system administrator must define a mail subdirectory for each label at which the system is to support mail, but other than that, the mail system functions at multiple labels quite transparently.

In practice, the multi-level secure mail system occasionally inconveniences users, who may not realize that they were sent mail at a label within their clearance range, but above the clearance of their current terminal session. To alleviate this potential problem, the system developers provided a mailbox-checking facility. *mailcheck* is a System V/MLS extension which allows users to determine whether there is mail in mailboxes at any or all of their other labels. *mailcheck* notifies the user of mail. If the user has mail at levels dominated by his or her current operating level, *mailcheck* prints the security label of each level in human-readable form, although does not print the contents of the messages(s). If the user has mail at a level not dominated by the user's current level (but within the user's clearance range), the string "(and other levels)" appears in the output. This message does not convey any information about the actual message(s) or about the level of the message(s), other than at least one other message exists within the user's clearance range. It is a somewhat incomplete implementation of a secure mail facility in that it permits a downward flow of information. This is acceptable in a B1 implementation. *mailcheck* does not report mail detected

in a mailbox above the user's maximum clearance. As an option and a convenience, *mailcheck* will forward mail from lower labels up to the current operating label in order for a user to read mail of a lower level. In order to access all users' mail files, the *mailcheck* program must be setuid to root.

/bin/newgrp: This program allows a user to change his or her discretionary access group after logging in, thus eliminating the need to log off and relogin at the new group. It checks the */mls/group* file to determine the user's authorized groups; i.e., that the requested change is to a valid group and there is a privilege in that group (of which the user is a member) corresponding to the user's current operating label. A new shell is then created with the user's real and effective group ID changed to the target group; the new values are used in mediating access control decisions. If the *newgrp* command is invoked as */bin/newgrp*, then the user's original shell remains in the background, and is reactivated upon termination of the new shell. This is done by the *fork* and *exec* system calls. However, if *newgrp*, the shell built-in, is invoked, then the current shell is overlaid with a new shell. The shell built-in only uses the *exec* system call. In order to modify the user's group ID values (which are stored in memory-based tables), the program must be setuid to root.

/bin/newpriv: This program allows a user to change his or her operating privilege. It is discussed in detail elsewhere in this document (see page 48, "Labels on System V/MLS").

/bin/passwd: This is the password setting and changing program in System V/MLS. It includes System V/MLS extensions, among which are the ability to define a minimum password length, the ability to require numeric characters in addition to alphabetics, and the ability to generate pseudo-random passwords; all of these features are enabled in the standard configuration of SystemV/MLS. *passwd* generates pronounceable random passwords which may be tailored to some extent by the user, based upon the value stored in the shell variable PASSWDOPTS, which determines ordering of syllables between alphabetics and numerics. The system administrator has the ability to set any user's password, and may explicitly override the random password selection mechanism. Since *passwd* modifies the file */mls/passwd*, it must be setuid to root.

/bin/ps: The *ps* command displays information about the status of processes on the system. When executed by the system administrator, *ps* is capable of displaying the status of all processes on the system; when executed by any other user, it will only display information about processes owned by that user and dominated by the label of the process which invoked the *ps* command. *ps* must run setuid to root in order to access the device */dev/mem*, from which it extracts process information.

/bin/su: This is the command which allows a process to assume the real and effective UID of the superuser (if the superuser password is known), or of any other user of the system, provided one knows the password for the target login ID and is operating at the SYSTEM label, or is currently operating as superuser. *su* is the only way of becoming root (superuser) on System V/MLS, andsu to root is restricted to terminals which have a device minimum label of SYSTEM. Users must be operating at the SYSTEM label in order to *su* to root or any other user. Since no untrusted user is expected to be able to login at the SYSTEM label (System V/MLS uses this mechanism in order to help ensure the incorruptibility of system code), very few user terminals should have this

capability. Because *su* must be able to create a process which has a UID of root, it is a setuid to root program.

/etc/getty: This program exists for every terminal line until auser attempts to login. Each *getty* resets its process group using the *setpgrp* system call, calls open for a particular terminal line, kills any background processes that have the terminal port open, and sleeps until the system senses a connection. Upon receiving a *login*, *getty* overlays itself with *login*.

/usr/bin/addgrp, /usr/bin/addpriv: These programs are SystemV/MLS utilities which allow the owner of a group or privilege to add members to them. These commands control much of the discretionary access control mechanism which exists in System V/MLS. In order for a user to grant discretionary access to a file, the user can create a group or privilege (using the *mkgrp* or *mkpriv* command described below), and set the group or privilege of the file to the new one by using the *chgrp* or *chpriv* commands. The user may then change the permission bit settings on the file (via the *chmod* command), granting the desired access to group members. The user can then add members to the group or privilege with the *addgrp* or *addpriv* commands. Operation of the commands is as follows: they accept as arguments the name of the group or privilege (if given a label and group, *addpriv* constructs the privilege name from that information), and the name of a user to be added to that group or privilege. The superuser may add members to any group or privilege, regardless of its ownership. Because these commands manipulate the shadow group file, */mls/group*, they must be setuid to root programs.

/usr/bin/at, /usr/bin/batch, /usr/bin/crontab: These programs are used to schedule execution of programs at a later time or date, or defer execution of programs until the system load decreases. Specifically, *at* and *crontab* allow a user to dictate a time and date for execution of a command sequence. *batch* submits a job for processing as soon as system load permits its execution. Jobs sent via *batch* go into a different CPU queue, and have a lower execution priority than ordinary interactive processes. All of these programs may be restricted by the use of files which will specifically allow or deny access to users identified by the system administrator. These programs require the ability to setuid to any user ID which may submit a job for processing, so all must run setuid to root. In System V/MLS, the ability to execute programs through these channels is restricted to a handful of administrative IDs.

/usr/bin/clearances: This is a System V/MLS utility which will inform a user of his or her maximum and minimum authorized security labels on the system. If executed by the superuser, clearances is capable of reporting all of the clearances available on the system, or the clearances for any user, or which users have access to a particular clearance. Because this program accesses the shadow group file and the */mls/clearances* file, it must be setuid to root.

/usr/lib/lpadmin: The *lpadmin* command is used to administer printers within the LP subsystem. Its three usages are to:

- set the system default destinations which *lp* checks when no destination has been explicitly specified.

- remove a printer from the LP subsystem by deleting the request directory, the qstatus entry, and the entries associated with the destination.

- change attributes of a printer (e.g. change pstat entries to reflect that the printer is hardwired, establish a new interface for a printer, select a model interface program for a printer, associate a new device with a printer).

lpadmin must be setuid to lp in order to write into */usr/spool/lp*.

cancel must be setuid to root in order to violate mandatory policy by writing the commands down into the FIFO which direct *lpsched* to cancel a previous print request.

/usr/bin/lsgrp, /usr/bin/lspriv: These commands list membership of all groups or privileges to which the user belongs (whether the user created the privilege or group or was added to the privilege or group). */usr/bin/lsgrp* is a link to the program contained in */usr/bin/lspriv*. Because the program must access the */mls/group* file, it must be setuid to root.

/usr/bin/lpstat: This program provides a mechanism through which all users may inquire about the status of the printers connected to the system. Various parameters determine what status information should be printed; specifically, one may determine whether a given printer is currently accepting requests, is enabled, has output requests queued, etc. Because *lpstat* retrieves its information from tables in memory to which access is restricted, it must be setuid to root.

/usr/bin/mkgrp, /usr/bin/mkpriv: These are programs which allow a user or system administrator to define new discretionary groups or privileges on the system. If either command is executed by a non-privileged user in order to create a new group, that user owns the group, and may add members to and delete members from it (if the system administrator creates a group or privilege, the first member he adds is considered to be the owner of the group or privilege). The group owner may also create privileges based on the group via the *mkpriv* command. A typical usage sequence would be as follows: user A wishes to establish a privilege called FUNNY, with members Larry, Moe, and Curly, operating at the label SECRET. First, the user would establish a discretionary group called FUNNYGRP, using the *mkgrp* command. Then, using the *mkpriv* command and specifying label SECRET and group FUNNYGRP, user A can create the privilege FUNNY (and explicitly add Larry, Moe and Curly to its membership as it is created). The privilege FUNNY will then exist at the desired label and with the desired members. Both of these commands (*mkgrp* and *mkpriv*) manipulate the shadow group and privilege files, and must therefore run setuid to root.

/usr/bin/mvpriv: This is a utility which allows a user to move a file from one directory into another directory (with a different label than the first directory) which has the same label as the file. This is necessary in some circumstances due to the ordering of the filesystem. When moving a file between directories of equal labels, the standard *mv* command can be used. However, System V/MLS MAC policy prohibits this command from moving files between directories of differing labels; for this, *mvpriv* must be used. The label of the file being moved must be the same as the label of the target directory. Also, the name of the file can not be changed (as it can with *mv*) when moved via *mvpriv*. *mvpriv* is setuid to root since it needs to violate MAC policy by writing to at least one directory with a label different from the label of the invoking process.

/usr/bin/delgrp, /usr/bin/delpriv: These two utilities delete members from a group or privilege. Only the owner of a group (or superuser) can delete members from that group. When this occurs, the members are also automatically deleted from all privileges that include the specified group. Similarly, only the owner of a privilege (or superuser) can delete members from that privilege. The owner can delete members of a specific privilege or members of all privileges that include a specified group or a specified label (with the owner's current operating group).

/usr/bin/rmgrp, /usr/bin/rmpriv: These two utilities delete groups or privileges from the system data files. Upon deletion from the system, the group and privilege numbers are set aside, for possible reuse at a later time. The procedure for reuse of privileges is detailed in the System V/MLS Trusted Facility Manual, which advises against doing this; it also lists procedures which may be followed to ensure that any reuse of group or privilege IDs will not contravene the system security policy. Both of these programs manipulate TCB data files, and therefore must be setuid to root.

/usr/lib/mailx/rmmail: This is a program which allows a user to remove an empty mail file from a mail subdirectory. In order to do this, it must be setgid to mail.

/usr/lib/mv_dir: *mv_dir* is called by *mv* if the object to be moved is really a directory. *mv_dir* only allows a user to rename a directory within its parent directory. Directories can not be moved from one directory to another via *mv_dir*. *mv_dir* needs to be setuid to root in order to do a move of a directory.

/usr/spool/lp/interface/5310 and */usr/lib/pgmark*:   The *5310* interface script is called by *lpsched* to derive the proper label (operating label of user) for the header and trailer banner pages of the hardcopy. The *5310* script calls the labels command to determine the correct label for the request. Page sensitivity labels can be added by *pgmark*. The sensitivity labels replace the top and bottom two lines with the appropriate label (operating label of user). If the label is too long for the top and bottom label areas, a ULI record reporting the "partial" disabling of labels at the top/bottom of pages is cut and the following string is substituted for the label: "** security label for privilege name too long to print **". The long label WILL be printed in the banner page, although it is likely that multiple pages will be needed to hold each banner page. Printer requests should be passed through a filter such as *pr* which inserts blank lines at the top and bottom so no data is lost from the hardcopy. Both are invoked with UID of *lp* and GID of bin in order to access files private to *lp*.

The 630 MTG Terminal Implementation

Overview

The 630 MTG (Multi-Tasking Graphics) terminal supports two to eight logical connections to the host multiplexed over a single tty connection; one data channel for each host window, plus a control channel (channel 0) reserved for the host process *layers* for direct communications with the 630 MTG terminal. Several 630 MTG terminals can be hooked to a host simultaneously. When a window is created, a data channel is associated with it. The xt driver (device driver for the virtual terminals) multiplexes I/O via the xt devices onto the real tty associated with the *layers* process managing the host end of the windowing protocol. The host resident application programs (e.g., shell) are

unaware that they are executing in a layers environment; each xt device appears to a host resident untrusted application to be a regular tty device.

The 630 MTG terminal can be used by a user or by a system administrator, but is not suitable for use as the system console. This is because it is possible to lose messages since processes which write error messages to the system console, like *login*, do not follow the xt protocol.

On the host side of the physical connection, there is an xt driver which multiplexes each data channel's I/O onto the real tty driver associated with the process, layers, managing that data channel. On the 630 MTG side of the physical connection, a firmware demultiplexer/controller, *demux*, receives the communication from the host. *demux* handles control information (such as download initiation signal) as well as passing data to the appropriate window.

The 630 MTG terminal has routines in its firmware which are called upon when a user requests certain terminal functions such as create a window and cut data from a window. A "630 process" has no relation to a UNIX process running on the host. There is only one process on the 630 MTG. That process in turn supports "threads". Threads may be throught of as streams of execution contained entirely within the 630 process. The 630 process creates and deletes, schedules and maintains data structures for all threads on the terminal. Each window has one and only one thread called *wproc*. Each thread has its own process structure (which contains a field for that thread's security label), and shares text and globals (neither containing user data) with every other thread. Threads do not have their own address space. Global variables shared among threads exist within the address space of the 630 process. Threads do however have their own local variables (such as program counters) and store them in their own private stacks. A thread's private stack is pointed to by the process structure. Process structures store the state they are in (e.g., RUN) and exist on a linked list of process structures. When a thread is executing, data stored in other threads is not available to the executing thread; although memory which that data might point to remains. This memory is not available to users due to the restrictions on downloading capability (see page 80, "Downloadable Software"). The 630 process is a trusted process and is part of the TCB (see page 22, "TCB Boundary").

When the 630 MTG is powered on or reset, the "cntlproc" thread is started. This is the underlying thread which reads all mouse input and controls the creation and deletion of window threads. A thread is created for each window created on the terminal. These threads are called "wproc" threads. This creation involves allocating a process structure and private stack for that thread, and initializing the keyboard queue (input from keyboard) and receive queue (input from host). When a window is deleted (via mouse option), the thread for that window is deleted. This involves clearing queues, freeing the stack and unlinking the process structure from the circular queue, thus freeing it. This circular queue is used for scheduling threads for execution. Each window is securely labeled. This is described on page 77, "Window Labels".

Logging Into The Host

A user invokes the trusted path on the 630 MTG terminal by powering the terminal down and back up, by pressing the Ctrl and Break keys simultaneously, or by pressing the shift, Ctrl, and Esc keys simultaneously. When this is done, a new *getty* is created and is attached to that terminal's port.

Then *getty* sanitizes the tty line and any xt devices previously associated with the terminal, and ensures that all processes attached to that terminal port are killed via the *kill* system call with the argument "-9", which specifies a nonmaskable termination signal. Then *getty* reads the login name and overlays itself with the *login* program. *login* gets and verifies the password, sets up the environment and checks the device clearances database (the */mls/cleardev* file) for security ranges, terminal types and the name of the handler for this terminal type. If login doesn't find a handler specified it overlays itself with the user's shell. *login* believes that there is a 630 MTG on the tty line if it sees a handler specified in the device clearances entry. For example, the following two entries in the */mls/cleardev* file show two devices:

> 515:4,1,2:1:L:630,/usr/bin/630init:
> 516:3:0:L:4425:

In this example, the device with a device number of 515 has a maximum clearance of level 4 with categories 1 and 2; has a minimum clearance of level 1; is a login device (i.e., a device on which a *getty* can be spawned) hardwired to a 630 MTG terminal; and executes /usr/bin/630init in place of the usual login shell. The next device, also a login device, has maximum clearance of level 3 and minimum clearance of level 0 (SYSTEM); is hardwired to a 4425 terminal; and runs the usual login shell. For a 630 MTG, *login* should find */usr/bin/630init* specified as the handler.

The *630init* Process

630init is a transient process which is overlayed by layers (as explained below). This *program first* verifies that the terminal is indeed a 630 MTG with the correct firmware version (version 8;8;6). To confirm this, *630init* downloads the *chk630* program, which verifies the state of the terminal, and computes a vector table checksum and a ROM checksum (checking for an un-corrupted terminal ROM). *chk630* reports the terminal state and computed checksums to 630init and returns the address of the end of ROM as an extra check to make sure that the firmware modifications are downloaded to the correct place. Finally *chk630* determines that only one host connection is enabled, no printer is enabled and that the cartridge port is not in use.

If these checks succeed, the *fw.mods* executable file is downloaded (by *630init*) and run on the 630 MTG terminal. *fw.mods* writes the security enhanced routines into RAM and sprinkles modifications throughout the RAM vector table so that the terminal will support the System V/MLS security policy; in doing so, it creates a secure smart terminal environment. The two main parts of this firmware modification involve:

1. Copy the firmware vector table from ROM into RAM. All functions called by the firmware will be branched to via this RAM table. *fw.mods* modifies addresses in the vector table to branch to multi-level handling routines for the 630 MTG terminal. This ensures use of the trusted 630 functions.

2. Changes the pointers for the PF keys which are located in 630 MTG Non-Volatile RAM (NVRAM) storage, causing them to be made inaccessible; these NVRAM-located items are effectively nulled each time the terminal's trusted path is invoked. Pointers to setup information, also kept in NVRAM, are not changed.[1]

Finally, *630init* overlays itself with *layers*.

layers

layers sets up the xt0 control connection and the xt1 user window, and labels the xt1 window with the security label of that device's minimum label, or of the user's requested label (default=minimum label), whichever is higher. The label on this window (along with the user's current operating label) may be changed (through execution of trusted code) by the *newpriv* command (see page 48, "Labels on System V/MLS").

layers causes the 630 MTG to operate in layers mode and arranges to pass SIGHUP to all its descendants. It makes the real tty device private to the TCB. *layers* reads its commands from channel 0 and creates/deletes windows until receiving the exit command. *layers* creates new processes, overlays them with a shell and terminates those processes running in windows to be deleted. Since *layers* is invoked directly from *login*, channel 0 can be trusted as a path for communication between the terminal and the host. Channel 0 in each set of xt devices will be private to the TCB. Users cannot invoke *layers* directly and *layers* is not a setuid to root program.

From this point forward, the user operates under *layers*, and may create and delete up to six more host windows, simply by selecting that option with the mouse. Each newly-created window is labeled with the user's login security label and runs the user's default login shell as specified in the */mls/passwd* file.

---

[1]   This is aceptable because terminal characteristics such as screen color and keyboard repeat rate are not security relevant. Terminal setup options can only be modified by predefined mouse operations.

wproc

wproc is the terminal emulator for windows in the layers environment. Once a window is created, host UNIX processes can be run in that window (e.g., shell). This section discusses how it is started and what content is stored in its associated structures.

The *cntlproc* thread is the underlying thread on the 630 MTG; it is always running (begins execution when machine is powered up). *cntlproc* initializes the 'Show Label' menu entry and then loops forever waiting for mouse clicks and setting flags.

When a new window is requested, *cntlproc* creates the *wproc* thread which then begins execution. Creation of this (or any) thread involves a number of activities; allocating a process structure, allocating a stack for *wproc*, setting the thread's state to RUN, initializing keyboard (buffers data from keyboard) and receive (buffers data from host) queues (used in the xt protocol), initializing registers to 0 and specifying the wproc program to run in the thread. Each *wproc* thread is associated with a window via specification in the process structure.

When *wproc* begins execution, it initializes its window structure and draws the label bar on the window. Then it enters an infinite loop checking flags set by *cntlproc*. These various flags result in the processing of mouse actions, label changes, window moves and reshapes, window activation/deactivation, cut and paste, and keyboard input. *wproc* constantly polls for these events, each event having its own service routine.

The window structure contains fields for cursor position, window size and boundaries, length of label string, etc. The wproc_line structure is the structure allocated for each line of text on the window, and contains the text, its size and pointers to the line which precedes it and the line which follows it. Other structures *wproc* uses are structures for specifying window coordinates and rectangles. These variables are manipulated on the *wproc* stack.

The *wproc* thread is labeled (much like a MLS host process has a label) in the process structure. Whenever a label is sent from the host to the 630 MTG terminal (e.g., at login when the first window is labeled, at each *newpriv* after that), the label is stored in the 630 MTG memory. Each *wproc* process structure contains pointers to the location in 630 MTG memory where the actual label in canonical and human-readable form is stored.

Window Labels

Each window has a trusted, human-readable label. This label is set to be that of the host process operating "in" the window. The top line of the window is known as the label bar, which is where the human-readable label is displayed. Writing by user processes in the label bar is disabled by the firmware. This is accomplished in *wproc*, which is hard coded to ignore the escape sequence the host sends to write into the label bar. Also, since the window's label is stored in the process structure and not in the window buffer itself, a user cannot move the mouse into the label bar area and click into it.

To untrusted software running on the host system, each window (xt1-xtn) appears to be a distinct terminal. Up to seven windows may be active at any given time (deleted windows may be recreated), at any classification level for which the user and the terminal device are authorized. It is possible for the complete security label of a window to be larger than the space available in the header line. In that event, as much of the security label as possible is displayed (the hierarchical classification is always displayed), and an indicator provided that the label is longer than is shown.[1] Clicking a mouse button on the appropriate pop-up menu box will display the entire security label of such a window.

Window labels may change (via trusted code), but they do not "float." In the event that a user executes the *newpriv* or *exit* command to change authorizations, the label of the window in which the user is operating will reflect the change. No other window will be affected.

Host windows are labeled in response to an *ioctl* from the host. All label information is sent over the xt0 control connection exclusively. The MLSLBLCHG *ioctl* takes as arguments lengths and pointers to the canonical and human-readable forms of the label being sent. The program invoking the *ioctl* sends it to the virtual channel on which that program is running. The xt driver re-routes the information to the control channel (xt0) in a message that contains the virtual xt number and the actual label (in canonical and human-readable form). The xt driver will not allow the *ioctl* unless it is sent from a process running as root. *demux* receives this control information and passes the received packets to the "doctl" routine. doctl reconstructs the message from the packets, stores the label in both forms in 630 MTG memory, and sends pointers to these locations to the appropriate *wproc* thread, which saves the pointers in its process structure and displays the human-readable string in the label bar. If the new label does not dominate the old label, security relevant data is cleared; pointers to the old label are replaced with pointers to the new label and the data currently in the window (and its scrolling buffer) is cleared.

In the event of a very long label, if there is not enough memory in which to store that label then the 630 MTG will either refuse to create the requested window, or will delete the window.

Secure Labeling at Login/Window Creation

During the login sequence layers sets up channel 0, the control channel, and initializes the first window. Before starting the shell which is to run in that window, it calls the mls library routine *devassign* to set the security label of the xt device. When *devassign* sets the label associated with the xt device, it issues the MLSLBLCHG *ioctl* to inform the 630 MTG of the new label.

---

[1]   The administrator is instructed in the TFM to choose hierarchical level names to be no longer than 20 characters.

When a new window is requested via a mouse click, the underlying *cntlproc* thread reads the mouse click and sends a command to *layers* to create a window. *layers* then proceeds as before.

Secure Labeling at Newpriv/Exit

The *newpriv* and *exit* (or <CNTL-D>) commands call *devassign*. When *devassign* changes the label associated with the xt device, it issues the MLSLBLCHG *ioctl* to inform the 630 MTG of the new label. The terminal then proceeds as before.

Window Creation

The enforcement mechanism used to ensure the integrity of windows and their associated buffers is based primarily upon the memory management scheme implemented by the 630 MTG terminal. Memory within a 630 MTG terminal is allocated via two calls: *alloc* and *gcalloc*, as described on page 20, "630 MTG Memory Management". These calls are used by the terminal and can not be used by user programs, since downloading is restricted (see page 80,"Downloadable Software"). Each window buffer is composed of memory allocated by *gcalloc*. Buffers have a maximum size of 10K bytes; this is an arbitrary limit imposed by the 630 MTG firmware. When a user creates a buffer, the terminal ensures that there are 10K bytes of memory space available. If not, the *gcalloc* memory pool is compacted. If there is still not a large enough memory fragment, an "out of memory" error is returned and the window creation fails (but terminal operations may continue).

Once the terminal ascertains that sufficient memory exists for a new window to be created, layers sends the label of the connection over channel 0. *cntlproc* creates a process structure for a new *wproc* thread. The *wproc* program (executing in the *wproc* thread) allocates for itself a window structure (on its stack) and allocates one zero-length data line. This window structure contains local variables (e.g., window coordinates) and contains no user data. After setting up the window structure, *wproc* enters an infinite loop checking the label-change flag at the top of the loop. When the set flag is detected, *wproc* clears the existing label on the window, if any, and copies the new label into the window structure and displays it. The details of this procedure are described on page 77, "Window Labels". As the first line is added into the buffer, the length of the first data line increases. Each new line which appears on the terminal is allocated another line of memory from the terminal pool of free memory, and the lines are linked together to form a doubly-linked list of text throughout the terminal's memory. Since each line in each buffer may be traced back to its header, all data is still associated with its sensitivity label. When the window buffer fills, lines stored at the beginning of the buffer are deleted as new lines are created.

Local Windows

In the event that a user wishes to create and modify a buffer without use of the host machine, the 630 MTG terminal allows for "local windows" - windows which have no process on the host associated with them. Although there is no host process executing in a local window, each local window still has an active *wproc* thread executing and has a process structure associated with it.

Local windows may be created before a user logs on to the host; however, these will be erased after the trusted path to login has been invoked. Local windows may also be created during a terminal

session by copying the complete contents from a host-connected window (it is "peeled"). Such a window will have the security label of the window from which the information originated. Security labels on local windows cannot be changed. Local windows use mouse-based editing. Off-screen buffering is employed in both host and local windows.

Cut and Paste

The 630 MTG terminal associates a buffer with each window, which is effectively a system buffer which is labeled with the label of the associated window. The user can scroll through these buffers and "cut and paste" text from one window to another. This functionality operates in accordance with the System V/MLS security policy. The cut and paste operation in a window is not an atomic action. First the user highlights (via mouse manipulation) text to be moved, then presses a mouse button. Each *cut* or *copy* will transfer the text to a newly allocated (via *alloc*) Global Save Buffer (GSB), which is effectively a buffer without a window attached, bearing the label of the window from whence the cut or copied text originated. The label of the GSB is stored in the actual buffer, unlike window buffers which have their labels stored in process structures. In this operation, the *wproc* thread associated with the source window is reading text itself and writing it into the GSB. The user must then move to and activate the target window, move the mouse to the location where the text is to be inserted, and press a mouse button. Here the *wproc* thread associated with the target window is reading text from the GSB (possibly at a lower level) and writing it into the target window's buffer. The target window's buffer has the same security label as the *wproc* thread causing the action. The 630 MTG terminal will compare, via a simple compare operation on the canonical form of the labels involved, the sensitivity of the GSB with the sensitivity of the target window; if the target does not dominate or if the two are incomparable, the transfer is disallowed. If the transfer is allowed, the terminal copies the text from the GSB to the target window. This may take place between any two windows, including local windows (described above).

Programmable Function Keys

On the base 630 MTG terminal, the character strings associated with the user programmable function keys (PFkeys) are stored in NVRAM. In the security enhanced 630 MTG terminal, a new PFkey storage area is allocated in RAM at login. The result is that the contents of PFkeys will not be retained between login sessions.

PFkey programming is restricted to the PFkey edit window. This window is labeled with the label of the user's login privilege (the lowest label a user has for the current session). Sending escape sequences from the host (in an attempt to modify the PFkeys) is mediated according to the System V/MLS MAC policy.

Downloadable Software

The downloading of software can only be initiated by the trusted process *dmdld*. *dmdld* checks that the terminal is in layers mode and changes ownership of the tty to root read/write only. It notifies the terminal (actually the channel associated with the window in which the download was invoked) to expect a download,using the JBOOT *ioctl* (which is restricted to root use only). Finally it does the actual download (sends the program on the same channel) using the xt protocol. When the

download is complete, *dmdld* changes ownership of the tty back to user with previous access mode, and execution resumes with the downloaded program.

Downloading software is restricted as follows. *dmdld* (setuid to root) only downloads programs from the directory */usr/dmd/bin*, which is writable only by root. The TFM prohibits the system administrator from changing the contents of this directory. The system no longer conforms to the evaluated configuration if the system administrator adds anything into this directory. Since users can not write to this directory, they can not write their own downloadable programs.

Additional Subjects and Objects Introduced

The 630 MTG terminal supports one entity which can be recognized as a trusted subject. This entity is a multiple-thread process. It enforces the system security policy on the 630 MTG terminal.

There is only one type of storage object local to the 630 MTG terminal: the buffers in which the GSB, individual terminal sessions and local windows exist. These buffers are system objects manipulated by the 630 MTG and are storage objects, in that they contain data which is subject to the Mandatory Access Control and Object Reuse requirements. They are not sharable with any other user on the system, and are destroyed when the user ends the terminal session. These window buffers are not directly accessible by untrusted subjects.

Auditing on the 630 MTG

All security relevant events are auditable by the host. There are no separately auditable (security relevant) events which occur within the 630 MTG terminal itself. For example, window buffer creation is implicitly auditable in that the (auditable) *devassign* of the associated xt channel implies the creation of the window buffer. Actions audited by the host can be distinguished in the audit trail as having taken place in a specific window. This information is derived from the device and inode number which are stored in the audit trail. When the audit trail is examined in verbose mode (see page 97, "Auditing") the "window name" (i.e., xt3) is printed. Another example is that of cut/copy/paste/send. The auditing of these data manipulation terminal commands is analogous to the auditing of IPC objects. The creation of IPC objects and opens for read/write access are auditable, but not the actual reads/writes. Similarly when two (or more) processes are created in different windows on a 630 terminal, their creation and the opening of System V/MLS objects are auditable. Therefore any process associated with a 630 MTG terminal must be presumed (by the administrator) to be able to exchange data with any other process associated with the terminal, as long as the communication does not violate the system security policy.

Logging Out

The terminal can detect drop of DTR (Data Terminal Ready) signal at any time during terminal session.

In order to end a 630 MTG terminal session, the user may take one of three distinct actions: power cycle the terminal; hold the shift, control, and escape keys simultaneously (this performs a software power cycle); or select the EXIT option with the mouse. The first two actions will cause DTR to

drop; the host will detect this, and send the hangup signal to all processes on the host which were affiliated with that terminal. If a process chooses to catch or ignore the SIGHUP and has not disconnected itself from the terminal device, it will be killed by *getty* when the next user attempts to login on the terminal device. When power is cycled, RAM is cleared. The reset key combination causes the terminal to initiate the clear/selftest/reboot sequence. The use of the EXIT option will send an instruction to the *layers* program running on the host which instructs it to terminate the session. When *cntlproc* gets the mouse click for EXIT it sends a message to the layers program running on the host. *layers* then kills all shells running in all windows and sends a message back to the terminal. On receiving the return message from *layers*, doctl runs the unbootmux routine which writes zeros throughout all of RAM and then performs a self test. When *layers* dies, *login*, waiting for the death of child signal, dies too, and a new *getty* is invoked for that line. This will log the user off of the host computer. After terminating all its descendants, *layers* restores the real tty to a known state (used by *getty*).

Based upon this overview of the functionality of the 630 MTG terminal, it is evident that there are a number of unique security issues related to this configuration. More detail on the features and assurances supported by the 630 MTG terminal will be found in Section 3 of this report.

Configuration Management

The system, in order to retain the B1 TCSEC rating, is proceeding into the Ratings Maintenance Phase (RAMP). This involves identifying all hardare components and tracking all software changes. There is a configuration management plan in place to track changes made to documentation, source code, test documentation, and test code. The System V/MLS Rating Maintenance Plan (RM Plan) document explains the configuration management procedures.

The principal divisions of AT&T which will play a part in maintaining the system are the Technologies Federal System Division and AT&T Bell Laboratories. This section will explain the scheme in place for managing changes to the System V/MLS product, and will then go on to explain how changes to the underlying UNIX System V are tracked.

The process for managing changes made to the System V/MLS product is as follows. All changes are initiated through the creation of a Modification Request (MR). MRs are submitted to the Configuration Control Board (CCB) in order to assign the MR to the most appropriate individual(s). These individuals are then responsible for developing a solution for the request as well as providing support and design documentation. After this step the MR solution is submitted for approval by the CCB. If the solution is inadequate, the MR is rejected and the problems with the proposed solution are identified. The MR continues to be submitted with new proposals until the CCB approves the proposed solution. All MRs are subject to a test analysis, which determines whether an appropriate test exists, whether an existing test must be modified, whether a new test must be written, or if code inspection is needed.

Configuration tools used throughout the configuration management system aid in automating procedures. Currently the tools used are Source Code Control System (SCCS), SABLE, and System V/MLS Tools.

SCCS records all enhancements and changes made to source code and documentation, comments on each version, and maintains a history of the changes made.

SABLE assists in the management of product development by performing modification request tracking, report and query functions, and human factors engineering. Modification request tracking is the mechanism used to track the current status of any MR. MRs can be in one of a number of states (i.e., accepted, under-study, deferred, approved, assigned, closed). A history is maintained of all MRs through all their states. Report and query functions allow the user to pull data base records (specific or in report format). Human factors engineering allows for a "customization" of SABLE users. This customization involves the setting of various defaults (e.g., preferred editor, menu vs command line entry).

There is manual interaction in place between SABLE and SCCS. These procedures are described in the *RM Plan for System V/MLS* .

System V/MLS Tools are tools specifically developed for configuration management which handle the tree structure of the System V/MLS product. These tools are documented in the RM Plan.

UNIX System V is configuration managed under the SCCS system at Bell Laboratories, although the System V/MLS VSAs are not participants in this particular configuration management process. For every RAMP cycle that entails a new release of the base operating system (UNIX System V), an analysis will be done by the VSA for each changed source file. All sources for the releases of UNIX System V to be RAMPed are maintained on a dedicated filesystem by the System V/MLS VSA. The VSA will use these sources for analysis of all changes. The VSA's analysis will involve contact with those organizations responsible for the code, for various queries. The VSA sends the results of his or her analysis to the CCB. The CCB will be responsible for approving the updates.

If a feature is added to the base operating system that adversely affects the system security and cannot be fixed, then the component is removed when the System V/MLS product is installed at the customer site. If the feature can be fixed, then the component is added, if not already present, to the System V/MLS product's source tree and work proceeds as normal for an MR.

The underlying hardware is not under configuration control at the System V/MLS development organization, as is the System V/MLS product (with histories of changes online, etc). The RAMP document[1] requires that the hardware be configuration identified and analyzed. The tracking process works as follows: When a new hardware base becomes available, the VSA examines the design and testing documentation. In effect, he or she does a complete mini-evaluation on the hardware. The VSA documents this analysis, keeps it on record, and presents the analysis to the CCB.

---

1   *Rating Maintenance Phase Program Document*, NCSC, June 1989.

If changes made to the underlying hardware are deemed potentially harmful to security by the CCB, the CCB does not approve that hardware product as an acceptable base and System V/MLS is not ported to that base.

AT&T Technologies Federal System Division is responsible for maintaining the rating for System V/MLS. The configuration management scheme enforced for System V/MLS provides added assurance that any changes made to the TCB will not compromise the trust of the originally evaluated system.

EVALUATION AS A B1 SYSTEM

Discretionary Access Control

Requirement

> The TCB shall define and control access between named users and named objects (e.g., files and programs) in the ADP system. The enforcement mechanism (e.g., self/group/public controls, access control lists) shall allow users to specify and control sharing of those objects by named individuals, or defined groups of individuals, or by both, and shall provide controls to limit propagation of access rights. The discretionary access control mechanism shall, either by explicit user action or by default, provide that objects are protected from unauthorized access.These access controls shall be capable of including or excluding access to the granularity of a single user. Access permission to an object by users not already possessing access permission shall only be assigned by authorized users.

Applicable Features

DAC is implemented in System V/MLS via protection bits on all named objects enumerated on page 38, "Objects". Protection bits are sufficient to provide self/group/public controls on sharing of objects by named individuals and defined groups of individuals. System V/MLS also allows for user-definable groups, called "privileges", which aid in controlling access and realistically fulfill the requirement of including or excluding access to the granularity of a single user. This is accomplished by allowing a user to change the privilege of a file to a specific privilege which the user may define, and then allowing the user to set the access mode bits to allow (or deny) members of the privilege access to the file. System V/MLS provides a default protection on newly created objects of read, write, and execute access for the owner of that object and no access to all others. For a complete description of DAC see page 44, "Discretionary Access Control".

DAC and the 630 MTG terminal: In the context of the 630 MTG terminal, DAC is a not an issue. The terminal retains no information between the time one user logs off of the SystemV/MLS system and the next user logs on; therefore, the only information to which the terminal has access at any given time is information available to the user currently logged into the host.The terminal has no role in mediation of any discretionary access, and therefore need only ensure that the access control decision of the host is not circumvented. Since the terminal physically does not have access to information to which its user does not have discretionary access, this requirement is trivially satisfied.

Conclusion

System V/MLS satisfies the B1 Discretionary Access Control requirement.

Object Reuse

Requirement

> All authorizations to the information contained within a storage object shall be revoked prior to initial assignment, allocation, or reallocation to a subject from the TCB's pool of unused storage objects. No information, including encrypted representations of information, produced by a prior subject's actions is to be available to any subject that obtains access to an object that has been released back to the system.

Applicable Features

In System V/MLS, the system administrator is responsible for ensuring that the object reuse requirements on mountable media (i.e., tapes and diskettes) are followed. The administrator must reformat the media using the appropriate UNIX format commands explained in section 5.6 of the *System V/MLS Trusted Facility Manual*

The System V/MLS TCB ensures the object reuse requirements are checked for the following:
> directories
> regular files
> special files
> named pipes
> unnamed pipes
> memory shared
> memory segments
> message queues
> semaphores

File System Objects

The file system objects are directories, regular files, pipes and special files. System V/MLS allocates disk blocks in a manner which ensures that no object reuse is possible for each of the previously mentioned file system objects (for further information see page 65, "Object Reuse"). Both named and unnamed pipes use kernel buffers to store data. The allocation of kernel buffers disallows object reuse for pipes as well as file system objects (see page 65, "Disk Block Allocation").

Memory-Based Objects

System V/MLS utilizes a demand paging mechanism. When a process requires another page of memory, the *growreg* kernel routine determines the number of new pages to be used, and initializes them by marking them invalid. When the process references the newpage, the system clears the page. Shared memory segments are implemented using the demand paging mechanism. (For further details see page 65, "Object Reuse".)

Message Queues and Semaphores

When memory for semaphores and message queues is initially allocated, System V/MLS clears this memory to ensure that no previous data may be obtained. Hereafter, both message queues and semaphores are cleared upon deallocation in System V/MLS. Deallocation occurs when the IPC_RMID command is sent to them *sgctl* or *semctl* system call.

Object Reuse on the 630 MTG Terminal

In order to prevent the scavenging of information from the terminal buffers when the connection to the host computer is broken, the 630 MTG will zero all windows and buffers, including local windows and Programmed Function (PF) keys. For the purpose of this evaluation, the term "storage object" may be defined simply as either a host or local window buffer in the context of the 630 MTG terminal. Each of these buffers is appropriately labeled, and each contains information and is subject to the object reuse requirement. Eliminating object reuse on the 630 MTG terminal is effectively a two-step process; first, when a user deletes a window, the window is removed from the screen and the memory which had been assigned to that window is made inaccessible. This is done by enforcing a high-water-mark mechanism for each window's address space. Second, all user-modifiable memory is cleared at the end of a user's terminal session; the terminal detects the Data Terminal Ready signal (DTR) drop, and upon that signal will zero its memory.

During a login session on the 630 MTG, all memory allocated for buffers, stacks and other 630 MTG system objects, is zeroed out by the allocation routines *alloc* and *gcalloc*.

Conclusion

System V/MLS satisfies the B1 Object Reuse requirement.

Labels

Requirement

> Sensitivity labels associated with each subject and storage object under its control (e.g., process, file, segment, device) shall be maintained by the TCB. These labels shall be used as the basis for mandatory access control decisions. In order to import non-labeled data, the TCB shall request and receive from an authorized user the security level of the data, and all such actions shall be auditable by the TCB.

Applicable Features

System V/MLS uses the UNIX group ID to implement its labeling scheme as described in the system overview (see page 48, "Labels on System V/MLS"). Group IDs are present in all subject and object structures. Mandatory access controls are enforced based on these labels as previously described on

page 49, "Mandatory Access Control". Non-labeled data which must be imported on to the system is initially labeled at SYSHI until proper labeling can be done by a system administrator.

Labeling on the 630 MTG Terminal

Labeling and the various sub-requirements associated with assuring the existence of trusted labels for all storage objects are handled by the 630 MTG terminal. There are many operations in which the 630 MTG terminal is required to enforce the labeling requirements. These fall into two ca'.gories: terminal operations and data manipulation operations. In the case of terminal operations (e.g., open window, delete window), the 630 MTG terminal communicates with the host computer via the xt0 port. In the case of data manipulation operations (e.g., cut, paste), the terminal ensures the integrity of the labels on those window buffers at all times. The 630 MTG ensures the integrity of human-readable labels attached to each window buffer present on the terminal.

Conclusion

System V/MLS satisfies the B1 Labels requirement.

Label Integrity

Requirement

> Sensitivity labels shall accurately represent security levels ofthe specific subjects or objects with which they are associated.When exported by the TCB, sensitivity labels shall accurately and unambiguously represent the internal labels and shall be associated with the information being exported.

Applicable Features

Sensitivity labels which are assigned to all subjects and objects are described in the system overview (see page 48, "Labels on System V/MLS"). The GIDs, which are stored internally in the user area, point to the actual label which is stored in internal kernel tables. When data files are exported in System V/MLS they are accompanied by their GIDs. Procedurally, they must also be accompanied by the /mls/labels mapping of those GIDs into their associated labels. For a discussion of the integrity of labels on the 630 MTG terminal (see page 77, "Window Labels").

Conclusion

System V/MLS satisfies the B1 Label Integrity requirement.

Exportation of Labeled Information

Requirement

> The TCB shall designate each communication channel and I/O device as either single-level or multilevel. Any change in this designation shall be done manually and shall be auditable by theTCB. The TCB shall maintain and be able to audit any change in the current security level or levels associated with a communication channel or I/O device.

Applicable Features

Secondary storage devices (such as disk or cartridge tape) on System V/MLS are multilevel. All other devices except the 630 MTG terminal are single-level. The interface to devices is via an unambiguously labeled special file located in the */dev* SECURED directory. Any changes to the level of device files (and thus to their associated device) are auditable.

The 630 MTG terminal is treated as a multi-level device corresponding to multiple single-level pseudo devices. The TCB maintains the clearance range of each terminal port in the data file */mls/cleardev*; all changes to this file are auditable events. Additionally, events which change the security level associated with a 630 MTG terminal session may be audited.

Conclusion

System V/MLS satisfies the B1 Exportation of Labeled Information requirement.

Exportation to Multilevel Devices

Requirement

> When the TCB exports an object to a multilevel I/O device, the sensitivity label associated with that object shall also be exported and shall reside on the same physical medium as the exported information and shall be in the same form (i.e.,machine-readable or human-readable form). When the TCB exports or imports an object over a multilevel communication channel, the protocol used on that channel shall provide for the unambiguous pairing between the sensitivity labels and the associated information that is sent or received.

Applicable Features

Multilevel devices are labeled with a maximum and minimum label. Access to multilevel I/O device files will be restricted to processes that enforce the labeling requirements (e.g. printer daemons, archiving programs, 630 MTG pseudodevice driver).

The 630 MTG terminal conforms to this requirement by maintaining a virtual terminal connection which is dedicated to information used for window control and data labeling. Each window on the 630 MTG terminal is displayed with a tamperproof header which contains its mandatory access control attributes.

Data can only be imported/exported to a floppy disk by the system administrator through the *backup* or *cpio* commands. Data files are imported/exported with their GIDs. They must also be accompanied by the */mls/labels* mapping. This is done via an administrative procedure.

Conclusion

System V/MLS satisfies the B1 Exportation to Multilevel Devices requirement.

Exportation to Single-Level Devices

Requirement

> Single-level I/O devices and single-level communication channels are not required to maintain the sensitivity labels of the information they process. However, the TCB shall include a mechanism by which the TCB and an authorized user reliably communicate to designate the single security level of information imported or exported via single-level communication channels or I/O devices.

Applicable Features

Facilities within *chpriv*, *newpriv*, and device-dependent functions provide reliable means by which authorized users may alter the level associated with single-level devices. Devices, including tty devices, normally reside in the */dev* directory. Single level devices, while in use, reside in exactly one SECURED subdirectory of */dev*. A single level device is only visible to a user if the device resides in the SECURED subdirectory of */dev* which corresponds to the user's current operating level. A user operating as root or in the group SECURED sees the "real" */dev* directory as well as all subdirectories.

The device clearances file, */mls/cleardev* , specifies the maximum and minimum permitted labels for the device. (Actually */mls/cleardev* specifies the maximum and minimum labels for the port -- but in a hardwired configuration we can identify the port with the device.) At login the maximum and minimum for the session are computed from the maximum and minimum of the port and user; this maximum and minimum are stored in the sessions database and are used by *newpriv* and *chpriv* to determine if the requested device reclassification is permitted.

At login the user's tty device is assigned the user's login label and is placed in the appropriate SECURED subdirectory of */dev*. Any subsequent changes in the label assigned to the tty by the user must be initiated by a trusted process, *newpriv*, which re-labels the tty line to reflect the current label of the user's process, and moves the tty to the appropriate SECURED subdirectory. *newpriv* consults the sessions database to ensure that the requested new operating label is permitted to both

the user and the tty. Every read and write to a tty line is verified to conform to MAC policy. Reads and writes to */dev/tty* are checked for both MAC and DAC.

chpriv can also be used to change the security classification of a single-level device. Only a user operating with a real UID of root will be permitted to use *chpriv* to change the security classification of a character device.

lpsched is the process which actually transmits files to the printer. Before sending the file, *lpsched* executes *newpriv* to change the classification of the printer device to that of the file to be printed. The labels of the printer and the file are identical for the duration of the print job. At the conclusion of the print job the printer is reclassified to SYSTEM.

Conclusion

System V/MLS satisfies the B1 Exportation to Single-Level Devices requirement.

Labeling Human-Readable Output

Requirement

> The ADP system administrator shall be able to specify the printable label names associated with exported sensitivity labels. The TCB shall mark the beginning and end of all human-readable, paged, hardcopy output (e.g., line printer output) with human-readable sensitivity labels that properly[1] represent the sensitivity of the output. The TCB shall, by default, mark the top and bottom of each page of human-readable, paged, hardcopy output (e.g., line printer output) with human-readable sensitivity labels that properly represent the overall sensitivity of the output or that properly represent the sensitivity of the information on the page. The TCB shall, by default and in an appropriate manner, mark other forms of human-readable output (e.g., maps, graphics) with human-readable sensitivity labels that properly represent the sensitivity of the output. Any override of these marking defaults shall be auditable by the TCB.

---

[1] The hierarchical classification component in human-readable sensitivity labels shall be equal to the greatest hierarchical classification of any of the information in hte output that the labels refer to; the non-hierarchical category component shall include all of the ono-hierarchical categories of the information in the output the labels refer to, but no other non-hierarchical categories.

Applicable Features

System V/MLS is capable of providing human-readable output in three ways: line printer output via the *lp* command; displayed in a window on the 630 MTG terminal; and displayed on a "dumb" terminal, such as the AT&T Teletype 4425 or 605 terminals.

In the event that the output is provided by the use of the *lp* command, the lp subsystem ensures that banner pages are printed for each job which display the current label of the user. A job may contain several files at different levels from a single user. The label on the banner and trailer pages of the whole print job will be the current label of the user and will therefore always dominate the label of any file contained in the job. Additionally, top and bottom page labels are provided by default; however, these may be overridden by the user. In the event that these are overridden, the act of overriding them is auditable. The top and bottom labels are not printed if the label is longer than 80 characters. In this case the top and bottom labels are replaced by the character string "** security label for privilege <priv name> is too long to print **" where name is the name of the user's current operating privilege. This label replacement is auditable.

The 630 MTG terminal labels all windows with their sensitivity label. The label display is dependent upon the size of the window. The hierarchical portion of the label is always displayed, as is as much of the non-hierarchical portion of the label as possible. In the event that the non-hierarchical part of the label is too long to be displayed, the 630 MTG flags the condition in the user's display, and provides an option which allows a user to display the entire label.

By default, the user's current level and privilege are defined a seach user's prompt in */etc/profile*. However, since users may select their own prompts, they may choose not to display their current level and privilege at all times. Also, if the default PS1 prompt, as defined in */etc/profile* would be more than 240 characters long, it is replaced by "<label too long> privname $" where privname is the name of the user's current operating privilege. For these reasons, System V/MLS provides a mechanism to display the user's current operating privilege at any time.

Conclusion

System V/MLS satisfies the B1 Labeling Human-Readable Output requirement.

Subject Sensitivity Labels

Requirement

> The TCB shall immediately notify a terminal user of each change in the security level associated with that user during an interactive session. A terminal user shall be able to query the TCB as desired for a display of the subject's complete sensitivity label.

Applicable Features

Terminal users may change their current operating privilege (mandatory label, discretionary group pair) during an interactive session via the *newpriv* or *exit* (or CNTL-D) commands. These commands print to the screen the new label of the user. In addition, the user's default prompt string is used to store the user's current operating label; this prompt can be modified by the user to contain any string. When *newpriv* or *exit* is invoked from a host connected 630 MTG terminal, the new label replaces the old label in the label bar for that window. Terminal users can request that the TCB display their current sensitivity label with the '*labels -u*' command. These mechanisms are explained in detail on (see page 54, "Changing Subject Sensitivity Label Interactively") and page 77, "Window Labels".

Conclusion

System V/MLS satisfies [1] the B2 Subject Sensitivity Labels requirement.

Device Labels

Requirement

> The TCB shall support the assignment of minimum and maximum security levels to all attached physical devices. These security levels shall be used by the TCB to enforce constraints imposed by the physical environments in which the devices are located.

Applicable Features

By invoking the *mkdevclr* command, the system administrator can enter into the */mls/cleardev* file the maximum and minimum levels at which a device may operate. Thereafter, the device is restricted to operate within that range of levels. Maximum and minimum levels may be assigned to all devices in the evaluated configuration.

Conclusion

System V/MLS satisfies[2] the B2 Device Labels requirement.

---

1 Although System V/MLS satisfies this requirement at the B2 level, it does not satisfy the assurance requirements above its rated level.

2 Although System V/MLS satisfies this requirement at the B2 level, it does not satisfy the assurance requirements above its rated level.

Mandatory Access Control

Requirement

The TCB shall enforce a mandatory access control policy over all subjects and storage objects under its control (e.g., processes, files, segments, devices). These subjects and objects shall be assigned sensitivity labels that are a combination of hierarchical classification levels and non-hierarchical categories, and the labels shall be used as the basis for mandatory access control decisions. The TCB shall be able to support two or more such security levels. The following requirements shall hold for all accesses between subjects and objects controlled by the TCB: A subject can read an object only if the hierarchical classification in the subject's security level is greater than or equal to the hierarchical classification in the object's security level and the non-hierarchical categories in the subject's security level include all the non-hierarchical categories in the object's security level. A subject can write an object only if the hierarchical classification in the subject's security level is less than or equal to the hierarchical classification in the object's security level and all the non-hierarchical categories in the subject's security level are included in the non-hierarchical categories in the object's security level. Identification and authentication data shall be used by the TCB to authenticate the user's identity and to ensure that the security level and authorization of subjects external to the TCB that may be created to act on the behalf of the individual user are dominated by the clearance and authorization of that user.

Applicable Features

System V/MLS enforces a mandatory security policy over all subjects and storage objects. This security policy maintains protection of objects such that no unauthorized subject is permitted to read or write objects (in this context, unauthorized means that the subject is executing at an inappropriate security level).

The mandatory security policy enforced by System V/MLS relies upon two basic relationships between the labels associated with subjects and objects - dominance and equivalence:

Label X dominates (>=) label Y when the hierarchical portion of label X is greater than or equal to the hierarchical portion of label Y, and label X contains at least all of the non-hierarchical categories that are contained in label Y.

Label X is equivalent (==) to label Y when the hierarchical portion of label X is identical to the hierarchical portion of label Y, and the set of non-hierarchical categories contained in label X is identical to the set of non-hierarchical categories contained in label Y.

The mandatory controls placed upon the System V/MLS objects are as follows:

- Files, directories

Read Access:	Subject> = Object
Execute Access:	Subject >= Object
Write Access:	Subject == Object

- Named pipes, unnamed pipes

Read Access:	Subject> = Object
Write Access:	Subject == Object

- Processes signaling other processes

Read Access:	No t Applicable
Write Access:	Subject (sending proc) == Object (receiving proc)

- Message queues, semaphores, shared memory

Read Access:	Subject >= Object
Write Access:	Subject == Object

Identification and authentication data is determined at *login* and stored in */mls/sessions*. When a user invokes a program. including SUID and SGID programs, to act on his or her behalf, the *exec* system call ensures that the sensitivity level of the process dominates the sensitivity level of the program. In addition, the invoking process's *sensitivity* level is preserved.

MAC and the 630 MTG Terminal

The portion of the TCB which runs in the 630 MTG terminal enforces System V/MLS mandatory access policy on the 630 MTG. The implementation of this feature is discussed on page 80, "Cut and Paste".

Conclusion

System V/MLS satisfies the B1 Mandatory Access Control requirement.

Identification and Authentication

Requirement

The TCB shall require users to identify themselves to it before beginning to perform any other actions that the TCB is expected to mediate. Furthermore, the TCB shall maintain authentication data that includes information for verifying the identity of individual users (e.g., passwords) as well as information for determining the clearance and authorizations of individual users. This data shall be used by the TCB to authenticate the user's identity and to ensure that the security level and authorizations of subjects external to the TCB that may be created to act on behalf of the individual user are dominated by the clearance

and authorization of that user. The TCB shall protect authentication data so that it cannot be accessed by any unauthorized user. The TCB shall be able to enforce individual accountability by providing the capability to uniquely identify each individual ADP system user. The TCB shall also provide the capability of associating this identity with all auditable actions taken by that individual.

Applicable Features

System V/MLS requires all users, including system operators, to identify and authenticate themselves before they are allowed to access system resources. Users enter login IDs and passwords to identify and authenticate themselves to the system. When a system administrator adds a user account, the TCB ensures that unique login IDs are provided on an individual basis.

Only system administrators may gain access to the identification and authentication information because the TCB maintains this data within the protected *mls* directory. The following files contain identification and authentication data: */mls/passwd, /mls/group*. In addition to storing the passwords in a file inaccessible to nonprivileged users, System V/MLS encrypts the passwords.

To obtain superuser privileges, administrators must first login at designated terminals as non-privileged users. These terminal ports have a minimum device label which is SYSTEM level, which is less than the unprivileged users' minimum. In addition, administrators must enter the *su* command and provide the superuser password in order to acquire administrative capabilities.

Identification and Authentication and the 630 MTG Terminal

After the user is identified and authenticated by the host, the terminal port is identified in the */mls/cleardev* file as being connected to a 630 MTG terminal. Next, firmware modifications are downloaded, and *layers* is invoked. After the trusted channel 0 has been established, one or more additional channels are created and the user is allowed to operate on the terminal. Any windows which are created by the user are constrained by the system MAC policy explained on page 48, "Mandatory Access Control".

Conclusion

System V/MLS satisfies the B1 Identification and Authentication requirement.

Trusted Path

Requirement

> The TCB shall support a trusted communication path between itself and users for initial login and authentication. Communications via this path shall be initiated exclusively by a user.

Applicable Features

System V/MLS supports a trusted communications path for initial user login to the system for all terminal types in the evaluated configuration. In all cases, the user is advised to power-cycle the terminal to ensure that the trusted path is established for the login identification and authentication sequence. It should also be noted that System V/MLS allows users the option of changing their password via this trusted path mechanism, through the use of an argument to the *login* command. Users should be encouraged to change their passwords through this interface rather than by use of the *passwd* command.

Conclusion

System V/MLS satisfies[1] the B2 Trusted Path requirement.

Audit

Requirement

> The TCB shall be able to create, maintain, and protect from modification or unauthorized access or destruction an audit trail of accesses to the objects it protects. The audit data shall be protected by the TCB so that read access to it is limited to those who are authorized for audit data. *The TCB shall be able to record the following types of events:* use of identification and authentication mechanisms, introduction of objects into a user's address space (e.g., file open, program initiation), deletion of objects, actions taken by computer operators and system administrators and/or system security officers, and other security relevant events. The TCB shall also be able to audit any override of human-readable output markings. For each recorded event, the audit record shall identify: date and time of the event, user, type of event, and success or failure of the event. For identification/authentication events the origin of request (e.g., terminal ID) shall be included in the audit record. For events that introduce an object into a user's address space and for object deletion events the audit record shall include the name of the object and the object's security level. The ADP system administrator shall be able to selectively audit the actions of any one or more users based on individual identity and/or object security level.

---

1   Although System V/MLS satisfies this requirement at the B2 level, it does not satisfy the assurance requirements above its rated level.

Applicable Features

The audit trail is a file consisting of a header and audit records. The header provides general information: identity of the system this audit trail was generated on and the time/date the system was brought into multi-user mode. The header information also provides a user name map, a group name map, a label name map, a terminal name map, and a filesystem map. These maps provide a translation between internal and human-readable names. In System V/MLS the audit trail is protected at SYSHI.

Each audit record has 2 parts: a header followed by a data block specific for that auditable event. Audit records are sequenced so that missing records can be detected. The header consists of the channel number, the number of bytes in the record, the process ID of the process writing the record, and a time stamp. The structure of each audit record is dependent on the type of event being recorded.

Currently 23 channels are used for the following kernel probe points. Subchannels which record kernel functions are indicated by indenting them beneath their channel.

sat_accs:	open for read/write access to a filesystem object granted
sat_accf:	open for read/write access to a filesystem object denied
sat_chown:	modification of the owner of a file
sat_chgrp:	modification of the discretionary group of a file
sat_chmod:	modification of the mode bits of a filesystem object
sat_clk:	reset of system clock
sat_startup:	record current time
sat_reclas:	reclassification of file
sat_exece:	successful/failed execs
sat_exit:	exit status of process
sat_fork:	process fork
sat_ipcaccs:	successful open for read/write access of IPC object
sat_ipcaccf:	failed open attempts for read/write access to IPC objects
sat_ipccreat:	creation of an IPC object
sat_ipcchown:	change of the owner of an IPC object
sat_ipcchgrp:	change of the discretionary group of an IPC object
sat_ipcchmod:	change of the mode bits of an IPC object
sat_ipcreclas:	reclassification of IPC object
sat_ipcrm:	removal of an IPC object
sat_kill:	all signals sent by privileged processes
sat_link:	new links to existing files
sat_mknd:	creation of file
sat_mount:	mount of local filesystem
sat_unmount:	unmount of filesystem
sat_pipe:	creationof unnamed pipes
sat_serr:	system calls that fail
sat_setuid:	modification of effective UID of a process
sat_setgid:	modification of effective GID of a process

sat_uli:	data written to sattrace device
sat_ulrm:	removal of file or directory

There are 14 user level audit trail probes:

mkuser:	adding a new user
rmuser:	removal of a user
mklbl:	creation of a new label
mkgrp:	addition of a new group or new privilege
rmpriv:	removal of a group or privilege
maxclear:	changes in a user's clearance
addpriv:	addition of new users to a group
delpriv:	removal of users from a group
5310:	suppression of labels on 5310 printer output.
mount:	the inode-name map of the filesystem being mounted
passwd:	records the change of a user's password
mkdevclr:	addition of a new device clearance
rmdevcl:	removal of a device clearance
login:	only unsuccessful login attempts-- successful logins are audited as combination of several kernel probe points (e.g. sat_exece, sat_fork)

System V/MLS contains an audit trail fomatter, *satfmt*, which formats the records in the audit trail in one of 3 formats: raw, verbose and sensitive. Raw mode outputs the audit record information completely in numeric characters in decimal rather than hex. Verbose mode converts all numeric references to symbolic names wherever possible. Sensitive mode outputs only those events defined as security sensitive and which are of special interest to the security administrator (e.g., failed access attempts).

satfmt maintains an internal name map data structure for each object defined on the system, including complete name maps for all mounted filesystems. This allows for path resolution to be determined quickly, without having to reference the actual filesystem. In addition, when the object has multiple links all path names are listed in verbose output. Also, *satfmt* with the -N option is the post-selection tool for choosing actions of a particular individual. System Administrators may selectively audit based on the object security level by using *grep* on the audit trail file. Instructions for using *grep* in this manner exist in the TFM.

When the currently active audit trail file becomes 80% full, a warning message is sent to the system console. When the current audit file fills, the *satsave* daemon switches to the next audit file, wrapping around to the first file when the last file is full. If the next audit trail file is not empty, the system will go into single user mode.

Auditing User Actions On The 630 M T G Terminal

All host connected processes are subject to auditing on the host. Terminal operations (e.g., create a window) are implicitly audited. The terminal operations which manipulate data between processes (e.g., cut/paste) are not audited. Both of these justifications and descriptions can be found on page

October 18, 1989

81, "Auditing on the 630 MTG". Logins on a 630 MTG terminal device are auditable, as are all other logins.

Conclusion

System V/MLS satisfies the B1 Audit requirement.

System /.rchitecture

Requirement

> The TCB shall maintain a domain for its own execution that protects it from external interference or tampering (e.g., by modification of its code or data structures). Resources controlled by the TCB may be a defined subset of the subjects and objects in the ADP system. The TCB shall maintain process isolation through the provision of distinct address spaces under its control. The TCB shall isolate the resources to be protected so that they are subject to the access control and auditing requirements.

Applicable Features

The System V/MLS TCB, which includes the 630 MTG terminal as part of the evaluated configuration, meets the system architecture requirement by maintaining a domain for its own execution and maintaining process isolation, both in the host and in the terminal. The host system provides these features via both hardware memory management and kernel data structure organization. As discussed previously, the system hardware supports both a kernel and user domain, with access to these memory spaces enforced by the hardware. System data and code files are protected from modification by both mandatory and discretionary policy, ensuring that the system is not corrupted. The 630 MTG terminal does not provide hardware memory management; however, the same functionality is achieved by ensuring that all code which runs on the terminal is trusted. This ensures that the execution domain of the 630 MTG terminal is inviolate. Process isolation is a topic which is not applicable to the 630 MTG terminal, in that the terminal executes only one process at a time, and the processes which may be executed are all trusted.

All objects and resources which are supported by System V/MLS in its evaluated configuration are defined to the TCB and exist under its protection. All elements of the system are protected by the System V/MLS security policy.

Conclusion

System V/MLS satisfies the B1 System Architecture requirement.

System Integrity

Requirement

Hardware and/or software features shall be provided that can beused to periodically validate the correct operation of the on-site hardware and firmware elements of the TCB.

Applicable Features

A complete set of system integrity tests are shipped with the System V/MLS system. The test facility is referred to as the Diagnostic Monitor (DGMON), and is shipped as standard equipment with each 3B2 computer. The test facility is a comprehensive one, and as such requires that the system be brought down to firmware mode before the tests may be run.

The DGMON tests exercise the CPU (including the privileged instructions), the system board, the memory, the Vcache, and the peripheral device controllers. The tests and instructions for their operation are discussed in a manual which is available to system administrators. This manual, the *AT&T 3B2 Computer Off-Line Diagnostics Manual*, provides a good description of each test, what it covers, and how to use it.

Conclusion

System V/MLS satisfies the B1 System Integrity requirement.

Security Testing

Requirement

The security mechanisms of the ADP system shall be tested and found to work as claimed in the system documentation. A team of individuals who thoroughly understand the specific implementation of the TCB shall subject its design documentation, source code, and object code to thorough analysis and testing. Their objectives shall be: to uncover all design and implementation flaws that would permit a subject external to the TCB to read, change, or delete data normally denied under the mandatory or discretionary security policy enforced by the TCB; as well as to assure that no subject (without authorization to do so) is able to cause the TCB to enter a state such that it is unable to respond to communications initiated by other users. All discovered flaws shall be removed or neutralized and the TCB retested to demonstrate that they have been eliminated and that new flaws have not been introduced.

Applicable Features

Overview of Vendor Test Suite

The complete AT&T System V/MLS test suite consists of three distinct parts: the security test suite, the functional tests, and the 630 MTG terminal test suite. Although there are several instances where security tests and functional tests may overlap, AT&T felt it was important to separate the two and perform each independent of the other. The 630 MTG terminal tests are also separated into both security and functional tests and will be discussed later.

The System V/MLS security tests are aimed at ensuring that all B1 security requirements as stated in the *Department of Defense Trusted Computer System Evaluation Criteria* are met. These tests are broken into the following six sections of the requirements: discretionary access contol, object reuse, labeling, mandatory access control, identification and authentication, and audit.

The functional tests ensure that the System V/MLS software functions correctly as described in the system documentation. These tests are broken into the following sections:

1) Section (1) System V User and Administrative Command Tests

2) Section (1S) System V/MLS Security Enhanced or New User & Administrative Command Tests

3) Section (2) System V System Call Tests

4) Section (2S) System V/MLS Security Enhanced or New System Call Tests

Testing of the trusted processes is included in the functional testing.

The 630 MTG terminal has its own test plan and procedures to test its functional and security properties. The underlying philosophy of the 630 MTG tests parallels the philosophy used by the SystemV/MLS tests; namely, that functional tests are constructed for functions advertised in the User's Guide and security tests are constructed from TCSEC requirements. The System V/MLS tests are first run on the 630 MTG to ensure that there is nothing specific to the 630 MTG terminal that affects the system. The goal of the 630 MTG test plan is to assure that System V/MLS security policy is enforced when a user is using the terminal. The 630 MTG functional and security tests are performed as described below.

Functional testing for the 630 MTG consists of tests for new or modified commands that are executed on the host and tests for 630 MTG specific functionality available through mouse menu interaction. Terminal commands, or mouse selections, are each tested manually. These tests involve such functions as bringing window top and current, putting window into local edit mode, interwindow data moves, creating local windows, creating new windows, moving/reshaping windows, showing a window's label, editing the programmable function keys, scrolling and terminal reset.

Security testing for the 630 MTG consists of tests which correspond to those TCSEC requirements involving the 630 MTG. These tests first cause certain operations to occur and then observe the effects or lack of effects of these operations. No extra testing is performed for Audit or DAC since the 630 MTG introduces no extra auditable events or discretionary access controls.

Team Augmentation of Vendor Coverage

In addition to AT&T's security and functional tests, the team developed more than 20 of its own tests. These tests fell into several areas:

- Testing the limits of the system (e.g., seeing what happens to the audit log when process numbers (PIDs) wrap from their maximum value back to their minimum value, determining whether SystemV/MLS can handle the maximum number of categories (1024), tryingto overrun the Global Save Buffer of the 630 MTG)

- Object reuse (e.g., seeing whether the registers of the math accelerator unit (MAU) are cleared during a process context switch)

- Testing various System V/MLS features (e.g., that the system administrator can turn on or off the ability of ordinary users to write and execute setuid/setgid programs and shell scripts, that no user, system administrators included, can login as root)

Performing Team Testing

The evaluation team conducted the System V/MLS testing on AT&T3B2/500 and 3B2/600 computers owned by the vendor and located at the vendor's site in Whippany, NJ. The evaluation team executed all of AT&T's automated and manual tests. The security test suite consists of 4 sets of automated tests and 2 sets of manual tests and the functional tests consist of 219 automated and 52 manual tests. The tests were run using Release 1.1.1 of System V/MLS integrated with Release 3.1.1 of UNIX System V. The evaluated software configuration consists of Release 1.1.2 of System V/MLS, which is Release 1.1.1 with the inclusion of fixes for problems encountered during team testing, integrated with System V Release 3.1.1. The hardware configuration of computers on which the tests were run is as follows:

1. AT&T 3B2/500 which includes:

      a. WE 32100 microprocessor and 18 MHz clock - CM518A
      b. WE 32101 Memory Management Unit
      c. WE 32106 Math Accelerator Unit (MAU)
      d . Virtual Cache
      e. Two 4 Mbyte ECC RAM - CM523A
      f. 147 Mbyte harddisk (Control Data Corp.) - KS-23371,L17
      g. 720 Kbyte floppy diskdrive - KS-23114,L4
      h. SCSI Host Adaptor Card - CM195W
      i. SCSI 60 Mbyte (TM/60S) cartridge tape drive - KS-23417,L2

    j. Enhanced Ports (EPORTS) Board - CM195Y

    k. AT&T 4425 Terminal as the System Console

    l. AT&T 4425 Terminal hardwired to EPORTS board

    m. AT&T 630 MTG terminal hardwired to EPORTS board

2. AT&T 3B2/600 which includes:

    a. WE 32100 microprocessor and 18 MHz clock - CM518A

    b. WE 32101 Memory Management Unit

    c. WE 32106 Math Accelerator Unit (MAU)

    d. Virtual Cache

    e. Two 4 Mbyte ECC RAM - CM523A

    f. 147 Mbyte harddisk (Control Data Corp.)
        - KS-23371, L17

    g. 720 Kbyte floppy diskdrive - KS-23114, L4

    h. SCSI Host Adaptor Card - CM195W

    i. SCSI 60Mbyte (TM/60S) cartridge tape drive
        - KS-23417, L2

    j. EnhancedPorts(EPORTS) Board - CM195Y

    k. AT&T 605 Terminal as the System Console

    l. AT&T 4425 Terminal hardwired to EPORTS board

    m. AT&T630 MTG terminal hardwired to EPORTS board

    n. AT&T 5310 DotMatrix Printer hardwired to EPORTS board

Problems Uncovered During System V/MLS Testing

The following problems were uncovered in the software as a result of the team's testing effort:

    1. There was a bug in the *satmap* routine.

    2. There was an hdelogger race condition during system initialization.

    3. The system failed to zero out the MAU registers during a process context switch.

    4. System V/MLS could not handle the maximum number of categories (1024).

The vendor made software fixes for each of the problems identified above, and the new software resulting from SystemV/MLS Release 1.1.1 with these fixes is Release 1.1.2. This release was retested to verify that the problems were corrected and that no additional problems were introduced during the process of making the requisite fixes. The team used the following computer configuration for performing the retesting activity:

AT&T 3B2/600 which includes:

a. WE 32100 microprocessor and 18 MHz clock - CM518A
b. WE 32101 Memory Management Unit
c. WE 32106 Math Accelerator Unit (MAU)
d. Virtual Cache
e. Two 4 Mbyte ECC RAM - CM523A
f. Two 147 Mbyte hard disks (Control Data Corp.)
    - KS-23371, L17
g. 720 Kbyte floppy disk drive
    - KS-23114,L4
h. SCSI Host Adaptor Card -CM195W

i. SCSI 60 Mbyte (TM/60S) cartridge tape drive
    - KS-23417,L2
j. Two Enhanced Ports (EPORTS) Board - CM195Y
k. One HiPorts Board - CM195BA
l. AT&T 4425 Terminal hardwired to an interface board
m. AT&T 630 MTG terminal hardwired to an interface board
n. AT&T 5310 Dot Matrix Printer hardwired to an interface board

The team also found numerous small omissions and unclear passages in the Test Plan, the Trusted Facility Manual, and the Installation Guide. These documentation problems have been addressed by the vendor and corrected in the final released documentation. The team did not find any design flaws during the testing of System V/MLS.

Conclusion

System V/MLS satisfies the B1 Security Testing requirement.

Design Specification and Verification

Requirement

A formal or informal model of the security policy supported bythe TCB shall be maintained over the life cycle of the ADP system and demonstrated to be consistent with its axioms.

Applicable Features

System V/MLS implements a modified version of the Bell and LaPadula security model to enforce mandatory security. The UNIX System V/MLS implementation is more restrictive than the Bell and LaPadula model in that write access to an object is only permitted when the subject and object have identical security labels. The System V/MLS security policy is maintained throughout the system including the 630 MTG terminal.

Conclusion

System V/MLS satisfies the B1 Design Specification and Verification requirement.

Security Features User's Guide

Requirement

> A single summary, chapter, or manual in user documentation shall describe the
> protection mechanisms provided by the TCB, guidelines on their use, and how they
> interact with one another.

Applicable Features

The *System V/MLS User's Guide and Reference Manual*, *630 MTG User's Guide*, *3B2 UNIX System V Programmers's Reference Manual*, and the *UNIX System V User's Guide* collectively meet the Security Features User's Guide requirements. These manuals are intended for users and provide descriptions of the functions of System V/MLS.

The *System V/MLS User's Guide* and the *630 MTG User's Guide* provide descriptions of and guidelines for the use of the protection mechanisms provided by the TCB. They are structured as follows:

System V/MLS User's Guide and Reference Manual

This manual consists of the following three sections: System V/MLS Policy Definition, System V/MLS Tutorial, and Manual Pages for System V/MLS Commands.

Section I   Defines the System V/MLS security policy, explains security labels, discretionary access control, and auditing.

Section II   Describes the features and general use of System V/MLS (e.g., login procedures, creating directories, making privileges).

Section III   Contains the manual pages for System V/MLS (includes new and modified commands).

630 MTG User's Guide

This manual describes the modifications made to the 630 MTG in order to allow it to be used within a multi-level secure computer system. Trustworthy security labels on windows, terminal firmware verification on login, and terminal memory clearing on logout are added security relevant features for the 630 MTG. In addition, this manual explains the 630 MTG features which have been disabled to prevent using the terminal to circumvent security.

Conclusion

System V/MLS satisfies the B1 Security Features User's Guide requirement.

Trusted Facility Manual

Requirement

> A manual addressed to the ADP system administrator shall present cautions about functions and privileges that should be controlled when running a secure facility. The procedures for examining and maintaining the audit files as well as the detailed audit record structure for each type of audit event shall be given. The manual shall describe the operator and administrator functions related to security, to include changing the security characteristics of a user. It shall provide guidelines on the consistent and effective use of the protection features of the system, how they interact, how to securely generate a new TCB, and facility procedures, warnings, and privileges that need to be controlled in order to operate the facility in a secure manner.

Applicable Features

The *System V/MLS Trusted Facility Manual, User's Guide and Reference Manual, System V User's Guide, 3B2 System V Programmer's Reference Manual, 3B2 System V System Administration Utilities Guide, 630 MTG User's Guide, 630/MLS Trusted Facility Manual-Multi Level Secure Window Management*, and the *630 MTG User's Manual* are intended for use by system operators and administrators. These manuals are used to guide an administrator or operator in the correct way to operate System V/MLS in a secure manner, and collectively meet the Trusted Facility Manual requirements.

System V/MLS Trusted Facility Manual

This manual consists of sections which describe the procedures for operating the system in a secure manner.

Section I   Introduces the types of system officers and explain show to use this guide.

Section II Describes the threats to information security as well as the measures taken to protect against these threats. Both internal and external threats are explained.

Section III Provides a description of labeling, mandatory accesscontrol, discretionary access control, identification and authentication, trusted path, auditing, and object reuse.

Section IV Explains the following: additional accountability, default path, Bourne shell, physical security, maximum number of file descriptors, and mandatory access control.

Section V   Explains system start-up and shutdown; mounting, unmounting, and storing labeled data; backing up and restoring files; maintaining the audit trail; and distributing labeled hardcopy.

Section VI   Describes the following: setting the system time and date; managing user accounts; installing and removing software; maintaining correct permissions; and granting special authorizations.

Section VII   Explains security administrator roles: administrating clearance information; securing directories; reclassifying information; reviewing audit trail data; and installing and uninstallin System V/MLS.

Section VIII   Includes appendices of setuid and setgid files that exist after System V/MLS is installed and the TCB listing.

630/MLS Trusted Facility Manual

This manual describes logical and physical connections to the host, downloading software, 630 installation, DAC, MAC, and object reuse. In addition, this manual describes accountability, and trusted communications.

Conclusion

System V/MLS satisfies the B1 Trusted Facility Manual requirement.

Test Documentation

Requirement

> The system developer shall provide to the evaluators a document that describes the test plan, test procedures that show how the security mechanisms were tested, and results of the security mechanisms' functional testing.

Applicable Features

The System V/MLS Test Plan describes the testing methodology employed by the testers of the System V/MLS system. The SystemV/MLS test plan outlines their specific testing approach, which consists of features testing of all mechanisms which exist to satisfy Criteria requirements. The tests are largely automated, with the exception of tests for identification and authentication, and other areas in which manual intervention is required.

Conclusion

System V/MLS satisfies the B1 Test Documentation requirement.

Design Documentation

Requirement

Documentation shall be available that provides a description of the manufacturer's philosophy of protection and an explanation of how this philosophy is translated into the TCB. If the TCB is composed of distinct modules, the interfaces between these modules shall be described. An informal or formal description of the security policy model enforced by the TCB shall be available and an explanation provided to show that it is sufficient to enforce the security policy. The specific TCB protection mechanisms shall be identified and an explanation given to show that they satisfy the model.

Applicable Features

There is no one document which purports to act as complete design documentation for System V/MLS. Instead, there exist several distinct documents, each detailing the operation of some subset of the entire system. The documentation for the underlying UNIX system upon which System V/MLS is based consists largely of Maurice Bach's *The Design of the UNIX Operating System*, as well as the course notes from several AT&T UNIX system internals classes.

The system developers have also written a series of relatively detailed papers which specifically discuss the operation of, and philosophy underlying, System V/MLS. Finally, some inspection of the system source code has shown that it suffices as documentation for the less complex trusted processes. AT&T developers have produced more detailed documentation in those cases in which the functionality of the trusted process was not readily discernable from a simple inspection of the commented code and cross references.

Conclusion

System V/MLS satisfies the B1 Design Documentation requirement.

This page intentionally left blank.

EVALUATORS' COMMENTS

The evaluation team found AT&T System V/MLS to be a flexible general purpose operating system when run in its evaluated configuration. During the course of working with the system, the team developed opinions about some of the system's design characteristics and features. The following are several of the team's opinions about System V/MLS.

- System V/MLS is unique among systems evaluated by the National Computer Security Center to date in that it allows use of a windowing terminal (the 630 MTG), which is capable of processing information of differing sensitivity levels simultaneously. The team found this to be an extremely useful and convenient feature. In addition to the 630 MTG terminal's ability to protect information at differing sensitivity labels is the terminal's intrinsic "smart" terminal functionality (e.g. line re-entry,"cut and paste" capabilities, programmable function keys), all of which operate within the boundaries defined by the System V/MLS security policy.

- If the superuser creates an object, the file's mandatory access control label is NOT derived from the source of the data. Instead, the file is labeled at SYSTEM level, the lowest mandatory access control label supported by the system. The superuser must then explicitly reclassify the object to the appropriate sensitivity label. For example, if the superuser wishes to concatenate two or more audit trail files together in order to produce a third file containing all of the audit data from a given day, then the resulting file, a derivative of several files with the mandatory access control label SYSHI, will be labeled as SYSTEM level. Although System V/MLS provides a warning about this to the administrator when the *su* command is successfully invoked, the team feels that undue care must be taken by system administrators in order to maintain the confidentiality of information which they may access in the course of their duties. It should be noted though that discretionary access control is still in place. The default umask setting is for owner access only.

- The 3B2/500, due to the design of its system board and the way that board is fitted into its chassis, has two separate console ports. It is important that only the console port on the back of the machine be used.

- In the event that there is no entry for a terminal device in the */mls/cleardev* file, then the clearance range of the device is from SYSTEM level to SYSHI - that is, the entire range of labels supported on the machine. The team feels that this is inappropriate, and that only devices with valid */mls/cleardev* entries should be allowed any access to the system. System V/MLS does, however, send a message to the system console each time a user logs into the system from a device which has no entry in the */mls/cleardev* file. Additionally it should be noted that a user's operation on a terminal is still limited to that user's clearance range.

- The team feels that it is important to note that System V/MLS loses NO audit data when the system audit log overflows. In this case, the buffer containing the audit data is saved to a file before the system stops processing user programs. Some catastrophic failures,

such as a power failure or system panic, may cause up to one buffer of audit data to be lost, and buffer size may be selected such that this amount of data may be made to be the size of one audit record.

- The system administrator responsible for reviewing the audit data should note that the *satfmt* command option "-S", which selects "sensitive" mode output, will only inform the user about those events which the vendor of System V/MLS has defined as "sensitive." These events exclude several occurrences which a prudent system administrator may find to be security-relevant. The team recommends the use of the "-V" option, which selects "verbose" output. In verbose mode, all of the audit data is reported to the administrator, who can then review it for events which may be considered to be of particular relevance to their installation. Verbose mode is the default option for *satfmt*.

- System V/MLS does support a trusted communications path at login time (and, through an argument to the login command, it supports a trusted path for password change, as well). This trusted path mechanism is simple, but effective; it is simply good security practice for users to power off their terminals at the end of their login sessions, and this practice will automatically invoke the System V/MLS trusted path mechanism.

- To provide discretionary access to the level of a single user, System V/MLS uses the traditional UNIX mechanism of protection bits for the file's owner, group, and other. The inconvenience of this mechanism in excluding single users, needing to constantly modify groups as new users gain access to the system or users are deleted from the system or group, as well as the impossibility of giving different users different access rights to an object have been thoroughly discussed in the literature and are well known within the computer security community. However, the situation is mitigated somewhat in the case of System V/MLS because the ordinary user can create his or her own groups and privileges, thereby being able to tailor groups and group membership to specific situations, even to creating groups to the granularity of a single file, if necessary.

- When a user sends mail to another user who is not cleared to the label of the message, the mail still goes through. The sending user is not notified of the situation. The receiving user can not read that mail (and of course is not notified of the mail) unless his or her clearance is raised to the label of the message.

EVALUATED HARDWARE COMPONENTS

The hardware components include: The 3B2/500 and the 3B2/600computers, in all configurations, provided that the chipset usedin the machine is the WE 32100 series chipset, and not the WE32200 series. Machines equipped with the WE 32200 chipset werenot addressed in the course of this evaluation.

The hardware boards which make up the WE 32100 chip set are thefollowing:

<u>Comcode</u>

System Board		CM518A	103984225
which includes:			
Microprocessor	WE32100		
Memory Management Unit (MMU)	WE 32101		
Math Accelerator Unit (MAU)	WE 32106		
Virtual Cache Memory		CM522A	103984472

Memory Extension Boards:

2 Mbyte	CM523B	103984605
4 Mbyte	CM523A	103984597

The following interface boards may be used to connect peripheraldevices to the 3B2/500 and 3B2/600 computers of the evaluated configuration:

<u>Comcode</u>

Ports Card	CM195B	103828620
HiPorts Card	CM195BA	103985362
EPortsCard	CM195Y	104166533
SCSI Host Adaptor	CM195W	104166525

The following devices may be used in conjunction with a 3B2/500or 3b2/600 while operating in a B1-level evaluated configuration.

	Device Type	Capacity	Interface	Comcode
	Fixed Disk	94M byte	SCSI	405188616
	Fixed Disk	135M byte	SCSI	405188608
*	Fixed Disk	147M byte	SCSI	405209552
	Fixed Disk	300M byte	SCSI	405428129
*	Cartridge Tape	60M byte	SCSI	405267568
	Cartridge Tape	120M byte	SCSI	405408147
	9-Track Tape	40M byte	SCSI	405218611

	9-Track Tape	160M byte	SCSI	405206848
*	Floppy Disk	720K byte	System Board	403960875

Display	Mono	EPORTS/PORTS/ HIPORTS	Model 4425	Comcode N/A
Display	Mono	EPORTS/PORTS/ HIPORTS	Model 605 BCT 501007850	
Display	Mono (Graphics/ Windowing)	EPORTS/PORTS/ HIPORTS	Model 630 MTG	

Controller	501001671
Monitor	501001 ⁷
Mouse	524594157
Keyboard	500064865
512 RAM	501002166

Printer	Dot Matrix	EPORTS/PORTS Model 5310 501006084

* indicates that the device is supplied with a base 3B2 system

The configurations tested by the team were: a 3B2/500 with aVcache board, two 4425 terminals, one 630 MTG terminal, onefloppy diskette, one 147M byte hard disk drive, one 60M bytecartridge tape drive, and two EPorts boards; a 3B2/600 with one 4425 terminal, one 605 BCT terminal, one 630 MTG terminal, one floppy diskette, two 147M byte hard disk drives, one 60M bytecartridge tape drive, two EPorts boards, and one 5310 printer; and a 3B2/600 with one 4425 terminal, one 630 MTG terminal, one floppy diskette, one 60M byte cartridge tape drive, two 147M byte hard disk drives, one HiPorts board, two EPorts boards, and one5310 printer.

EVALUATED SOFTWARE COMPONENTS

Below is a complete listing of the installed components from the following packages:

- AT&T UNIX System V Release 3.1.1 Operating System Utilities

- AT&T System V/MLS Release 1.1.2

- AT&T 630/MLS Release 1.1.2

The number preceding each item indicates:

T - TCB item evaluated by NCSC.

(blank) - Non-TCB item not evaluated, can not be used by administrator.

Distributed Software Components

T	/bin/ar
T	/bin/basename
T	/bin/cat
T	/bin/chgrp
T	/bin/chmod
T	/bin/chown
T	/bin/chpriv
	/bin/cmp
T	/bin/cp
T	/bin/cpio
T	/bin/date
T	/bin/dd
T	/bin/df
T	/bin/diff
T	/bin/dirname
T	/bin/du
T	/bin/echo
T	/bin/ed
	/bin/env
T	/bin/epstty
T	/bin/expr
T	/bin/false
T	/bin/file
T	/bin/find
T	/bin/grep
T	/bin/ipcrm

T	/bin/ipcs
T	/bin/kill
T	/bin/labels
T	/bin/line
T	/bin/ln
T	/bin/login
T	/bin/ls
T	/bin/mail
	/bin/mesg
T	/bin/mkdir
T	/bin/mv
T	/bin/newgrp
T	/bin/newpriv
T	/bin/nice
T	/bin/nohup
T	/bin/od
T	/bin/passwd
T	/bin/pdp11
T	/bin/pr
T	/bin/ps
T	/bin/pwd
T	/bin/red
T	/bin/rm
T	/bin/rmail
T	/bin/rmdir
T	/bin/rsh
T	/bin/sed
T	/bin/setpgrp
T	/bin/sh
T	/bin/sleep
T	/bin/sort
T	/bin/stty
T	/bin/su
T	/bin/sum
T	/bin/sync
T	/bin/tail
T	/bin/tee
T	/bin/time
T	/bin/touch
T	/bin/true
T	/bin/tty
T	/bin/u370
T	/bin/u3b
T	/bin/u3b15
T	/bin/u3b2
T	/bin/u3b5

T	/bin/uname
T	/bin/vax
T	/bin/wc
T	/bin/who
T	/bin/write
T	/boot/CONLOG
T	/boot/DISP
T	/boot/EPORTS
T	/boot/GENTTY
T	/boot/HDELOG
T	/boot/IDISK
T	/boot/IPC
T	/boot/IUART
T	/boot/KERNEL
T	/boot/MAU
T	/boot/MEM
T	/boot/MLS
T	/boot/MSG
T	/boot/OSM
T	/boot/PDI_
T	/boot/PIR
T	/boot/PORTS
T	/boot/PRF
T	/boot/S5
T	/boot/SAT
T	/boot/SCSI
T	/boot/SD00
T	/boot/SEM
T	/boot/SHM
T	/boot/ST01
T	/boot/STUBS
T	/boot/SXT
T	/boot/VCACHE
T	/boot/XT
T	/dgmon
T	/dgn/.edt_swapp
T	/dgn/EPORTS
T	/dgn/MAU
T	/dgn/PORTS
T	/dgn/SBD
T	/dgn/SCSI
T	/dgn/VCACHE
T	/dgn/X.EPORTS
T	/dgn/X.MAU
T	/dgn/X.PORTS
T	/dgn/X.SBD

T	/dgn/X.SCSI
T	/dgn/X.VCACHE
T	/dgn/edt_data
T	/edt/SCSI/edt_data
T	/edt/SCSI/edtgen
T	/etc/TIMEZONE
	/etc/advtab
T	/etc/bcheckrc
T	/etc/brc
T	/etc/bupsched
T	/etc/bzapunix
T	/etc/checklist
T	/etc/chroot
T	/etc/ckauto
T	/etc/ckbupscd
T	/etc/clri
T	/etc/conslog
T	/etc/coredirs
T	/etc/crash
T	/etc/cron
T	/etc/dcopy
T	/etc/devnm
T	/etc/dfsck
T	/etc/disketteparm
T	/etc/disks
T	/etc/drvinstall
T	/etc/editsa
T	/etc/edittbl
T	/etc/errcrash
T	/etc/errdump
T	/etc/errint
T	/etc/express
T	/etc/ff
T	/etc/finc
T	/etc/fltboot
T	/etc/fmtflop
T	/etc/fmthard
T	/etc/format
T	/etc/frec
T	/etc/fsck
T	/etc/fsck1b
T	/etc/fsdb
T	/etc/fsdb1b
T	/etc/fsstat
T	/etc/fstab
T	/etc/fstyp

T	/etc/fstyp.d/s5fstyp
T	/etc/fuser
T	/etc/getmajor
T	/etc/getty
T	/etc/gettydefs
T	/etc/gettype
T	/etc/group
T	/etc/grpck
T	/etc/hdeadd
T	/etc/hdefix
T	/etc/hdelogger
T	/etc/helpadm
T	/etc/init
T	/etc/init.d/ANNOUNCE T
	/etc/init.d/MOUNTFSYS
T	/etc/init.d/PRESERVE
T	/etc/init.d/README
T	/etc/init.d/RMTMPFILES
T	/etc/init.d/autoconfig
T	/etc/init.d/cron
T	/etc/init.d/disks
T	/etc/init.d/firstcheck
T	/etc/init.d/perf
T	/etc/init.d/scsi
T	/etc/init.d/sysetup
T	/etc/inittab
T	/etc/ioctl.syscon
T	/etc/killall
T	/etc/labelit
T	/etc/ldsysdump
	/etc/led
T	/etc/link
T	/etc/log/filesave.log
T	/etc/magic
	/etc/master.d/README
T	/etc/master.d/conlog
T	/etc/master.d/disp
T	/etc/master.d/eports
T	/etc/master.d/gentty
T	/etc/master.d/hdelog
T	/etc/master.d/idisk
T	/etc/master.d/ipc
T	/etc/master.d/iuart
T	/etc/master.d/kernel
T	/etc/master.d/mau
T	/etc/master.d/mem

T	/etc/master.d/mls
T	/etc/master.d/msg
T	/etc/master.d/osm
T	/etc/master.d/pdi_
T	/etc/master.d/pir
T	/etc/master.d/ports
T	/etc/master.d/prf
T	/etc/master.d/s5
T	/etc/master.d/sat
T	/etc/master.d/scsi
T	/etc/master.d/sd00
T	/etc/master.d/sem
T	/etc/master.d/shm
T	/etc/master.d/st01
T	/etc/master.d/stubs
T	/etc/master.d/sxt
T	/etc/master.d/vcache
T	/etc/master.d/xt
T	/etc/mkboot
T	/etc/mkfs
T	/etc/mknod
T	/etc/mkunix
T	/etc/mnttab
T	/etc/motd
T	/etc/mount
T	/etc/mountall
T	/etc/mvdir
T	/etc/ncheck
T	/etc/newboot
	/etc/oadvtab
T	/etc/passwd
T	/etc/ports
T	/etc/prfdc
T	/etc/prfld
T	/etc/prfpr
T	/etc/prfsnap
T	/etc/prfstat
T	/etc/profile
T	/etc/prtconf
T	/etc/prtconf.d/scsi
T	/etc/prtvtoc
T	/etc/pump
T	/etc/pwck
T	/etc/rc.d/lp
T	/etc/rc.d/setup
T	/etc/rc0

T	/etc/rc0.d/K00ANNOUNCE
T	/etc/rc0.d/K06satstop
T	/etc/rc0.d/K70cron
T	/etc/rc2
T	/etc/rc2.d/K06satstop
T	/etc/rc2.d/S00firstcheck
T	/etc/rc2.d/S00scsi
T	/etc/rc2.d/S01MOUNTFSYS
T	/etc/rc2.d/S02PRESERVE
T	/etc/rc2.d/S05RMTMPFILES
T	/etc/rc2.d/S06satstart
T	/etc/rc2.d/S07mlsstart
T	/etc/rc2.d/S10disks
T	/etc/rc2.d/S15autoconfig
T	/etc/rc2.d/S20sysetup
T	/etc/rc2.d/S21perf
T	/etc/rc2.d/S75cron
T	/etc/rc2.d/S80errstart
T	/etc/rc3
T	/etc/rmha
T	/etc/save.d/except
T	/etc/savecpio
T	/etc/scsi/compress
T	/etc/scsi/compress.d/qtape
T	/etc/scsi/edittbl
T	/etc/scsi/mkdev
T	/etc/setclk
T	/etc/setmnt
T	/etc/shutdown
T	/etc/stdprofile
T	/etc/swap
T	/etc/sysdef
T	/etc/system
T	/etc/telinit
	/etc/termcap
T	/etc/uadmin
T	/etc/umount
T	/etc/umountall
T	/etc/unlink
T	/etc/utmp
T	/etc/volcopy
T	/etc/wall
T	/etc/whodo
T	/etc/wtmp
T	/filledt
T	/lib/lboot

T	/lib/mboot
T	/lib/olboot
T	/lib/pump/eports
T	/lib/pump/ports
T	/lib/pump/ports.hpp
T	/lib/pump/scsi
T	/mls/.mkgrp
T	/mls/.mkuser
T	/mls/bin/LpSetup
T	/mls/bin/MailSetup
T	/mls/bin/Permsetup
T	/mls/bin/SecadmSetup
T	/mls/bin/UucpSetup
T	/mls/bin/group.cleanup
T	/mls/bin/mklbl
T	/mls/bin/mlsstart
T	/mls/bin/prlbl /mls/bin/rsh
T	/mls/bin/satfmt
T	/mls/bin/satmap
T	/mls/bin/satsave
T	/mls/bin/satstart
T	/mls/bin/satstop
T	/mls/bin/sessions
T	/mls/bin/sfsmap
	/mls/bin/sh
T	/mls/bin/updatepwgr
T	/mls/categories
T	/mls/clearances
T	/mls/cleardev
T	/mls/group
T	/mls/h_labels
T	/mls/labels
T	/mls/levels
T	/mls/passwd
T	/shlib/libc_s
	/shlib/libnsl_s
T	/unix
T	/usr/adm/bin/mvlog
T	/usr/adm/errlog
	/usr/adm/graflog
T	/usr/adm/sulog
T	/usr/admin/.gettyvalues19
T	/usr/admin/.profile
T	/usr/admin/bupsched
T	/usr/admin/checkfsys
T	/usr/admin/checkfsys.d/diskette

T	/usr/admin/gettyvalues
T	/usr/admin/makefsys
T	/usr/admin/makefsys.d/diskette
T	/usr/admin/menu/DESC
T	/usr/admin/menu/diagnostics/DESC
T	/usr/admin/menu/diagnostics/diskrepair
T	/usr/admin/menu/diagnostics/diskreport
T	/usr/admin/menu/diskmgmt/DESC
T	/usr/admin/menu/diskmgmt/checkfsys
T	/usr/admin/menu/diskmgmt/checkfsys.d/diskette
T	/usr/admin/menu/diskmgmt/cpdisk
T	/usr/admin/menu/diskmgmt/cpdisk.d/diskette
T	/usr/admin/menu/diskmgmt/erase
T	/usr/admin/menu/diskmgmt/erase.d/diskette
T	/usr/admin/menu/diskmgmt/format
T	/usr/admin/menu/diskmgmt/format.d/diskette
T	/usr/admin/menu/diskmgmt/harddisk/DESC
T	/usr/admin/menu/diskmgmt/harddisk/display
T	/usr/admin/menu/diskmgmt/harddisk/display.d/disk
T	/usr/admin/menu/diskmgmt/harddisk/format
T	/usr/admin/menu/diskmgmt/harddisk/format.d/disk
T	/usr/admin/menu/diskmgmt/harddisk/makehdfsys
T	/usr/admin/menu/diskmgmt/harddisk/partitioning
T	/usr/admin/menu/diskmgmt/harddisk/partitioning.d/disk
T	/usr/admin/menu/diskmgmt/harddisk/rmdisk
T	/usr/admin/menu/diskmgmt/harddisk/rmdisk.d/disk
T	/usr/admin/menu/diskmgmt/makefsys
T	/usr/admin/menu/diskmgmt/makefsys.d/diskette
T	/usr/admin/menu/diskmgmt/mountfsys
T	/usr/admin/menu/diskmgmt/mountfsys.d/diskette
T	/usr/admin/menu/diskmgmt/umountfsys
T	/usr/admin/menu/diskmgmt/umountfsys.d/diskette
T	/usr/admin/menu/filemgmt/DESC
T	/usr/admin/menu/filemgmt/backup
T	/usr/admin/menu/filemgmt/backup.d/9track
T	/usr/admin/menu/filemgmt/backup.d/diskette
T	/usr/admin/menu/filemgmt/backup.d/qtape
T	/usr/admin/menu/filemgmt/bupsched/DESC
T	/usr/admin/menu/filemgmt/bupsched/schedcheck
T	/usr/admin/menu/filemgmt/bupsched/schedmsg
T	/usr/admin/menu/filemgmt/diskuse
T	/usr/admin/menu/filemgmt/fileage
T	/usr/admin/menu/filemgmt/filesize
T	/usr/admin/menu/filemgmt/hsbackup
T	/usr/admin/menu/filemgmt/hsbackup.d/9track
T	/usr/admin/menu/filemgmt/hsbackup.d/disk

T	/usr/admin/menu/filemgmt/hsbackup.d/qtape
T	/usr/admin/menu/filemgmt/hsrestore
T	/usr/admin/menu/filemgmt/hsrestore.d/9track
T	/usr/admin/menu/filemgmt/hsrestore.d/disk
T	/usr/admin/menu/filemgmt/hsrestore.d/qtape
T	/usr/admin/menu/filemgmt/restore
T	/usr/admin/menu/filemgmt/restore.d/9track
T	/usr/admin/menu/filemgmt/restore.d/diskette
T	/usr/admin/menu/filemgmt/restore.d/qtape
T	/usr/admin/menu/filemgmt/store
T	/usr/admin/menu/filemgmt/store.d/9track
T	/usr/admin/menu/filemgmt/store.d/diskette
T	/usr/admin/menu/filemgmt/store.d/qtape
T	/usr/admin/menu/machinemgmt/DESC
T	/usr/admin/menu/machinemgmt/autold
T	/usr/admin/menu/machinemgmt/firmware
T	/usr/admin/menu/machinemgmt/floppykey
T	/usr/admin/menu/machinemgmt/powerdown
T	/usr/admin/menu/machinemgmt/reboot
T	/usr/admin/menu/machinemgmt/whoson
T	/usr/admin/menu/packagemgmt/DESC
T	/usr/admin/menu/softwaremgmt/DESC
T	/usr/admin/menu/softwaremgmt/install
T	/usr/admin/menu/softwaremgmt/installpkg
T	/usr/admin/menu/softwaremgmt/installpkg.d/diskette
T	/usr/admin/menu/softwaremgmt/listpkg
T	/usr/admin/menu/softwaremgmt/removepkg
T	/usr/admin/menu/softwaremgmt/removepkg.d/diskette
T	/usr/admin/menu/softwaremgmt/runpkg
T	/usr/admin/menu/softwaremgmt/runpkg.d/diskette
T	/usr/admin/menu/softwaremgmt/tapepkg
T	/usr/admin/menu/softwaremgmt/uninstall
T	/usr/admin/menu/syssetup/DESC
T	/usr/admin/menu/syssetup/admpasswd
T	/usr/admin/menu/syssetup/datetime
T	/usr/admin/menu/syssetup/nodename
T	/usr/admin/menu/syssetup/setup
T	/usr/admin/menu/syssetup/syspasswd
T	/usr/admin/menu/tapemgmt/DESC
T	/usr/admin/menu/tapemgmt/compress
T	/usr/admin/menu/tapemgmt/compress.d/qtape
T	/usr/admin/menu/tapemgmt/rmtape
T	/usr/admin/menu/tapemgmt/rmtape.d/9track
T	/usr/admin/menu/tapemgmt/rmtape.d/qtape
T	/usr/admin/menu/ttymgmt/DESC
T	/usr/admin/menu/ttymgmt/lineset

T	/usr/admin/menu/ttymgmt/mklineset
T	/usr/admin/menu/ttymgmt/modtty
T	/usr/admin/menu/usermgmt/DESC
T	/usr/admin/menu/usermgmt/addgroup
T	/usr/admin/menu/usermgmt/adduser
T	/usr/admin/menu/usermgmt/delgroup
T	/usr/admin/menu/usermgmt/deluser
T	/usr/admin/menu/usermgmt/lsgroup
T	/usr/admin/menu/usermgmt/lsuser
T	/usr/admin/menu/usermgmt/modadduser
T	/usr/admin/menu/usermgmt/modgroup/DESC
T	/usr/admin/menu/usermgmt/modgroup/chgname
T	/usr/admin/menu/usermgmt/moduser/DESC
T	/usr/admin/menu/usermgmt/moduser/chgloginid
T	/usr/admin/menu/usermgmt/moduser/chgpasswd
T	/usr/admin/menu/usermgmt/moduser/chgshell
T	/usr/admin/mountfsys
T	/usr/admin/mountfsys.d/diskette
T	/usr/admin/powerdown
T	/usr/admin/profile.dot
T	/usr/admin/setup
T	/usr/admin/sysadm
T	/usr/admin/umountfsys
T	/usr/admin/umountfsys.d/diskette
T	/usr/admin/unixadmin
	/usr/bin/300
	/usr/bin/300s
	/usr/bin/4014
	/usr/bin/450
T	/usr/bin/630init
T	/usr/bin/PFload
T	/usr/bin/addgrp
T	/usr/bin/addpriv
T	/usr/bin/adduser
	/usr/bin/assist
T	/usr/bin/at
	/usr/bin/awk
	/usr/bin/banner
T	/usr/bin/batch
	/usr/bin/bc
	/usr/bin/bdiff
	/usr/bin/bfs
	/usr/bin/cal
	/usr/bin/calendar
T	/usr/bin/cancel
	/usr/bin/captoinfo

18 October 1989

T	/usr/bin/checkfsys
	/usr/bin/chrtbl
T	/usr/bin/clearances
	/usr/bin/col
	/usr/bin/comm
T	/usr/bin/crontab
	/usr/bin/csplit
	/usr/bin/ctags
T	/usr/bin/cut
	/usr/bin/dc
T	/usr/bin/delgrp
T	/usr/bin/delpriv
T	/usr/bin/deluser
	/usr/bin/deroff
	/usr/bin/diff3
	/usr/bin/dircmp
T	/usr/bin/disable
T	/usr/bin/dmdld
T	/usr/bin/dominates
T	/usr/bin/dsconfig
	/usr/bin/edit
	/usr/bin/egrep
T	/usr/bin/enable
	/usr/bin/ex
	/usr/bin/factor
	/usr/bin/fgrep
T	/usr/bin/getopt
	/usr/bin/glossary
	/usr/bin/graf/abs
	/usr/bin/graf/af
	/usr/bin/graf/bar
	/usr/bin/graf/bel
	/usr/bin/graf/bucket
	/usr/bin/graf/ceil
	/usr/bin/graf/cor
	/usr/bin/graf/cusum
	/usr/bin/graf/cvrtopt
	/usr/bin/graf/dtoc /
	/usr/bin/graf/erase
	/usr/bin/graf/exp
	/usr/bin/graf/floor
	/usr/bin/graf/gamma
	/usr/bin/graf/gas
	/usr/bin/graf/gd
	/usr/bin/graf/ged
	/usr/bin/graf/gtop

usr/bin/graf/hardcopy
/usr/bin/graf/hilo
/usr/bin/graf/hist
/usr/bin/graf/hpd
/usr/bin/graf/label
/usr/bin/graf/list
/usr/bin/graf/log
/usr/bin/graf/lreg
/usr/bin/graf/mean
/usr/bin/graf/mod
/usr/bin/graf/pair
/usr/bin/graf/pd
/usr/bin/graf/pie
/usr/bin/graf/plot
/usr/bin/graf/point
/usr/bin/graf/power
/usr/bin/graf/prime
/usr/bin/graf/prod
/usr/bin/graf/ptog
/usr/bin/graf/qsort
/usr/bin/graf/quit
/usr/bin/graf/rand
/usr/bin/graf/rank
/usr/bin/graf/remcom
/usr/bin/graf/root
/usr/bin/graf/round
/usr/bin/graf/siline
/usr/bin/graf/sin
/usr/bin/graf/subset
/usr/bin/graf/td
/usr/bin/graf/tekset
/usr/bin/graf/title
/usr/bin/graf/total
/usr/bin/graf/ttoc
/usr/bin/graf/var
/usr/bin/graf/vtoc
/usr/bin/graf/whatis
/usr/bin/graf/yoo
/usr/bin/graph
/usr/bin/graphics
/usr/bin/greek
/usr/bin/help
/usr/bin/hp
/usr/bin/hpio
T   /usr/bin/id
/usr/bin/infocmp

	/usr/bin/ismpx
	/usr/bin/join
	/usr/bin/jterm
	/usr/bin/jwin
T	/usr/bin/layers
	/usr/bin/locate
	/usr/bin/logname
T	/usr/bin/lp
T	/usr/bin/lpstat
T	/usr/bin/lsgrp
T	/usr/bin/lspriv
T	/usr/bin/mailcheck
T	/usr/bin/mailx
T	/usr/bin/makefsys
T	/usr/bin/maxclear
T	/usr/bin/mkdevclr
T	/usr/bin/mkgrp
T	/usr/bin/mkpriv
T	/usr/bin/mkuser
T	/usr/bin/mountfsys
T	/usr/bin/mvpriv
	/usr/bin/nawk
	/usr/bin/newform
	/usr/bin/news
	/usr/bin/nl
	/usr/bin/oawk
T	/usr/bin/pack
	/usr/bin/paste
T	/usr/bin/pcat
T	/usr/bin/pg
T	/usr/bin/powerdown
T	/usr/bin/rmdevclr
T	/usr/bin/rmgrp
T	/usr/bin/rmpriv
T	/usr/bin/rmuser
T	/usr/bin/sadp
T	/usr/bin/sag
T	/usr/bin/sar
T	/usr/bin/sdiff
T	/usr/bin/setup
	/usr/bin/shl
	/usr/bin/spell
	/usr/bin/spline
	/usr/bin/split
	/usr/bin/starter
T	/usr/bin/sysadm

T	/usr/bin/tabs
T	/usr/bin/tic
T	/usr/bin/timex
	/usr/bin/tplot
T	/usr/bin/tput
T	/usr/bin/tr
T	/usr/bin/umountfsys
	/usr/bin/uniq
	/usr/bin/units
T	/usr/bin/unpack
	/usr/bin/usage
	/usr/bin/vedit
	/usr/bin/vi
	/usr/bin/view
	/usr/bin/what
	/usr/bin/xargs
T	/usr/bin/xtd
T	/usr/bin/xts
T	/usr/bin/xtt
T	/usr/dmd/bin/chk630
T	/usr/dmd/bin/fw.mods
	/usr/include/curses.h
	/usr/include/mls/mlsfiles.h
	/usr/include/mls/security.h
	/usr/include/pn.h
	/usr/include/sys/acct.h
	/usr/include/sys/adv.h
	/usr/include/sys/bitmasks.h
	/usr/include/sys/boot.h
	/usr/include/sys/boothdr.h
	/usr/include/sys/bpb.h
	/usr/include/sys/bubd.h
	/usr/include/sys/buf.h
	/usr/include/sys/callo.h
	/usr/include/sys/cdump.h
	/usr/include/sys/cio_defs.h
	/usr/include/sys/cirmgr.h
	/usr/include/sys/clock.h
	/usr/include/sys/cmn_err.h
	/usr/include/sys/comm.h
	/usr/include/sys/conf.h
	/usr/include/sys/crtctl.h
	/usr/include/sys/csr.h
	/usr/include/sys/ctype.h
	/usr/include/sys/debug.h
	/usr/include/sys/devcode.h

/usr/include/sys/diagnostic.h
/usr/include/sys/dir.h
/usr/include/sys/dirent.h
/usr/include/sys/disasm.h
/usr/include/sys/diskette.h
/usr/include/sys/dma.h
/usr/include/sys/dsd.h
/usr/include/sys/du_dep.h
/usr/include/sys/eccmem.h
/usr/include/sys/edt.h
/usr/include/sys/elog.h
/usr/include/sys/ep_dep.h
/usr/include/sys/ep_lla.h
/usr/include/sys/eppc.h
/usr/include/sys/erec.h
/usr/include/sys/err.h
/usr/include/sys/errno.h
/usr/include/sys/extbus.h
/usr/include/sys/faultr.h
/usr/include/sys/fblk.h
/usr/include/sys/fcntl.h
/usr/include/sys/file.h
/usr/include/sys/filsys.h
/usr/include/sys/firmware.h
/usr/include/sys/flock.h
/usr/include/sys/fs/s5dir.h
/usr/include/sys/fs/s5fblk.h
/usr/include/sys/fs/s5filsys.h
/usr/include/sys/fs/s5inode.h
/usr/include/sys/fs/s5macros.h
/usr/include/sys/fs/s5param.h
/usr/include/sys/fsid.h
/usr/include/sys/fstyp.h
/usr/include/sys/gate.h
/usr/include/sys/gdpstr.h
/usr/include/sys/getpages.h
/usr/include/sys/hdeioctl.h
/usr/include/sys/hdelog.h
/usr/include/sys/hetero.h
/usr/include/sys/id.h
/usr/include/sys/idtab.h
/usr/include/sys/if.h
/usr/include/sys/immu.h
/usr/include/sys/inline.h
/usr/include/sys/ino.h
/usr/include/sys/inode.h

/usr/include/sys/io.h
/usr/include/sys/iobd.h
/usr/include/sys/iobuf.h
/usr/include/sys/ioctl.h
/usr/include/sys/ipc.h
/usr/include/sys/iu.h
/usr/include/sys/jioctl.h
/usr/include/sys/lboot.h
/usr/include/sys/libxedt.h
/usr/include/sys/lihdr.h
/usr/include/sys/lla.h
/usr/include/sys/lla_ppc.h
/usr/include/sys/lock.h
/usr/include/sys/log.h
/usr/include/sys/macro.h
/usr/include/sys/map.h
/usr/include/sys/mau.h
/usr/include/sys/message.h
/usr/include/sys/mount.h
/usr/include/sys/msg.h
/usr/include/sys/nami.h
/usr/include/sys/netid.h
/usr/include/sys/nserve.h
/usr/include/sys/nvram.h
/usr/include/sys/open.h
/usr/include/sys/page.h
/usr/include/sys/panregs.h
/usr/include/sys/param.h
/usr/include/sys/pcb.h
/usr/include/sys/pdi.h
/usr/include/sys/pfdat.h
/usr/include/sys/pir.h
/usr/include/sys/poll.h
/usr/include/sys/pp_dep.h
/usr/include/sys/ppc.h
/usr/include/sys/ppc_lla.h
/usr/include/sys/proc.h
/usr/include/sys/psw.h
/usr/include/sys/pump.h
/usr/include/sys/que.h
/usr/include/sys/queue.h
/usr/include/sys/rbuf.h
/usr/include/sys/rdebug.h
/usr/include/sys/recover.h
/usr/include/sys/reg.h
/usr/include/sys/region.h

18 October 1989

/usr/include/sys/rfsys.h
/usr/include/sys/romscb.h
/usr/include/sys/rst.h
/usr/include/sys/sbd.h
/usr/include/sys/sccsid.h
/usr/include/sys/scsi.h
/usr/include/sys/scsi_edt.h
/usr/include/sys/scsi_ha.h
/usr/include/sys/sdi.h
/usr/include/sys/seg.h
/usr/include/sys/sem.h
/usr/include/sys/sema.h
/usr/include/sys/sgs.h
/usr/include/sys/shm.h
/usr/include/sys/signal.h
/usr/include/sys/sit.h
/usr/include/sys/st00_ioctl.h
/usr/include/sys/st01_ioctl.h
/usr/include/sys/stat.h
/usr/include/sys/statfs.h
/usr/include/sys/stdio.h
/usr/include/sys/stermio.h
/usr/include/sys/stream.h
/usr/include/sys/strlog.h
/usr/include/sys/stropts.h
/usr/include/sys/strstat.h
/usr/include/sys/swap.h
/usr/include/sys/sxt.h
/usr/include/sys/sys3b.h
/usr/include/sys/sysgdb.h
/usr/include/sys/sysinfo.h
/usr/include/sys/sysmacros.h
/usr/include/sys/systm.h
/usr/include/sys/tcon.h
/usr/include/sys/termio.h
/usr/include/sys/tihdr.h
/usr/include/sys/times.h
/usr/include/sys/timod.h
/usr/include/sys/tiuser.h
/usr/include/sys/todc.h
/usr/include/sys/trace.h
/usr/include/sys/ttold.h
/usr/include/sys/tty.h
/usr/include/sys/tuneable.h
/usr/include/sys/types.h
/usr/include/sys/uadmin.h

	/usr/include/sys/user.h
	/usr/include/sys/ustat.h
	/usr/include/sys/utsname.h
	/usr/include/sys/var.h
	/usr/include/sys/vcache.h
	/usr/include/sys/vtoc.h
	/usr/include/term.h
	/usr/include/unctrl.h
	/usr/include/windows.h
T	/usr/lbin/admerr
T	/usr/lbin/agefile
T	/usr/lbin/askx
T	/usr/lbin/checklist
T	/usr/lbin/checkre
T	/usr/lbin/checkyn
T	/usr/lbin/chkyn
T	/usr/lbin/devinfo
T	/usr/lbin/disklabel
T	/usr/lbin/diskumount
T	/usr/lbin/drivename
T	/usr/lbin/fdate
T	/usr/lbin/filecheck
T	/usr/lbin/fsinfo
T	/usr/lbin/getedt
T	/usr/lbin/ignore
T	/usr/lbin/labelfsname
T	/usr/lbin/largest
T	/usr/lbin/lightenfs
T	/usr/lbin/mklost+found
T	/usr/lbin/mkmenus
T	/usr/lbin/mktable
T	/usr/lbin/ncpio
T	/usr/lbin/num
T	/usr/lbin/oldfile
T	/usr/lbin/optparttn
T	/usr/lbin/readpkg
T	/usr/lbin/restorefiles
T	/usr/lbin/rmjunk
T	/usr/lbin/rmkdir
T	/usr/lbin/rrmdir
T	/usr/lbin/samedev
T	/usr/lbin/selectdevice
T	/usr/lbin/selpattern
T	/usr/lbin/skip
T	/usr/lbin/spacewatch
T	/usr/lbin/spclsize

T   /usr/lbin/vmkfs
             /usr/lib/.COREterm/1/1620
             /usr/lib/.COREterm/1/1640
             /usr/lib/.COREterm/2/2621
             /usr/lib/.COREterm/2/2621-wl
             /usr/lib/.COREterm/2/2621A
             /usr/lib/.COREterm/2/2621a
             /usr/lib/.COREterm/2/2622
             /usr/lib/.COREterm/2/2622a
             /usr/lib/.COREterm/2/2623
             /usr/lib/.COREterm/2/2623a
             /usr/lib/.COREterm/2/2624
             /usr/lib/.COREterm/2/2624-4p
             /usr/lib/.COREterm/2/2624a
             /usr/lib/.COREterm/2/2624b
             /usr/lib/.COREterm/2/2626
             /usr/lib/.COREterm/2/2626A
             /usr/lib/.COREterm/2/2626P
             /usr/lib/.COREterm/2/2626a
             /usr/lib/.COREterm/2/2626p
             /usr/lib/.COREterm/2/2640
             /usr/lib/.COREterm/2/2640a
             /usr/lib/.COREterm/2/2640b
             /usr/lib/.COREterm/2/2644a
             /usr/lib/.COREterm/2/2645
             /usr/lib/.COREterm/2/2648
             /usr/lib/.COREterm/2/2648A
             /usr/lib/.COREterm/2/2648a
             /usr/lib/.COREterm/3/31
             /usr/lib/.COREterm/3/36
             /usr/lib/.COREterm/3/3a
             /usr/lib/.COREterm/4/4012
             /usr/lib/.COREterm/4/4013
             /usr/lib/.COREterm/4/4014
             /usr/lib/.COREterm/4/4015
             /usr/lib/.COREterm/4/4023
             /usr/lib/.COREterm/4/4024
             /usr/lib/.COREterm/4/4025
             /usr/lib/.COREterm/4/4025cu
             /usr/lib/.COREterm/4/4027
             /usr/lib/.COREterm/4/4027cu
             /usr/lib/.COREterm/4/4112
             /usr/lib/.COREterm/4/4112-nd
             /usr/lib/.COREterm/4/4113
             /usr/lib/.COREterm/4/4114
             /usr/lib/.COREterm/4/4115

```
       /usr/lib/.COREterm/4/42
       /usr/lib/.COREterm/4/4410
       /usr/lib/.COREterm/4/4415
       /usr/lib/.COREterm/4/4418
       /usr/lib/.COREterm/4/4424
       /usr/lib/.COREterm/4/4424-1
       /usr/lib/.COREterm/4/4424-2
       /usr/lib/.COREterm/4/4424-3
   T   /usr/lib/.COREterm/4/4425
       /usr/lib/.COREterm/4/4426
       /usr/lib/.COREterm/4/450
       /usr/lib/.COREterm/5/513bct
       /usr/lib/.COREterm/5/5410
       /usr/lib/.COREterm/5/5418
       /usr/lib/.COREterm/5/5420
       /usr/lib/.COREterm/5/5420_2
       /usr/lib/.COREterm/5/5425
       /usr/lib/.COREterm/5/5620
       /usr/lib/.COREterm/6/610bct
       /usr/lib/.COREterm/9/925
       /usr/lib/.COREterm/9/950
       /usr/lib/.COREterm/A/ATT4410
       /usr/lib/.COREterm/A/ATT4415
       /usr/lib/.COREterm/A/ATT4418
       /usr/lib/.COREterm/A/ATT4424
       /usr/lib/.COREterm/A/ATT4424-2
   T   /usr/lib/.COREterm/A/ATT4425
       /usr/lib/.COREterm/A/ATT4426
       /usr/lib/.COREterm/A/ATT513
       /usr/lib/.COREterm/A/ATT5410
       /usr/lib/.COREterm/A/ATT5418
       /usr/lib/.COREterm/A/ATT5420
       /usr/lib/.COREterm/A/ATT5420_2
       /usr/lib/.COREterm/A/ATT5425
       /usr/lib/.COREterm/A/ATT5620
       /usr/lib/.COREterm/A/ATT610BCT
       /usr/lib/.COREterm/A/ATTPT505
       /usr/lib/.COREterm/a/adm3
       /usr/lib/.COREterm/a/adm31
       /usr/lib/.COREterm/a/adm36
       /usr/lib/.COREterm/a/adm3a
       /usr/lib/.COREterm/a/adm42
       /usr/lib/.COREterm/a/att4410
       /usr/lib/.COREterm/a/att4415
       /usr/lib/.COREterm/a/att4418
       /usr/lib/.COREterm/a/att4424
```

T

```
/usr/lib/.COREterm/a/att4424-2
/usr/lib/.COREterm/a/att4425
/usr/lib/.COREterm/a/att4426
/usr/lib/.COREterm/a/att513
/usr/lib/.COREterm/a/att5410
/usr/lib/.COREterm/a/att5418
/usr/lib/.COREterm/a/att5420
/usr/lib/.COREterm/a/att5420_2
/usr/lib/.COREterm/a/att5425
/usr/lib/.COREterm/a/att5620
/usr/lib/.COREterm/a/att610bct
/usr/lib/.COREterm/a/attpt505
/usr/lib/.COREterm/b/bantam
/usr/lib/.COREterm/c/c100
/usr/lib/.COREterm/c/c100-4p
/usr/lib/.COREterm/c/c104
/usr/lib/.COREterm/c/c108
/usr/lib/.COREterm/c/c108-8p
/usr/lib/.COREterm/c/concept
/usr/lib/.COREterm/c/concept100
/usr/lib/.COREterm/c/concept108
/usr/lib/.COREterm/c/concept108-8p
/usr/lib/.COREterm/d/decwriter
/usr/lib/.COREterm/d/diablo
/usr/lib/.COREterm/d/dmd
/usr/lib/.COREterm/d/dumb
/usr/lib/.COREterm/d/dw
/usr/lib/.COREterm/d/dw1
/usr/lib/.COREterm/d/dw2
/usr/lib/.COREterm/d/dw3
/usr/lib/.COREterm/d/dw4
/usr/lib/.COREterm/h/hp2621
/usr/lib/.COREterm/h/hp2621-wl
/usr/lib/.COREterm/h/hp2621A
/usr/lib/.COREterm/h/hp2621a
/usr/lib/.COREterm/h/hp2622
/usr/lib/.COREterm/h/hp2622a
/usr/lib/.COREterm/h/hp2623
/usr/lib/.COREterm/h/hp2623a
/usr/lib/.COREterm/h/hp2624
/usr/lib/.COREterm/h/hp2624a
/usr/lib/.COREterm/h/hp2624b
/usr/lib/.COREterm/h/hp2624b-4p
/usr/lib/.COREterm/h/hp2626
/usr/lib/.COREterm/h/hp2626a
/usr/lib/.COREterm/h/hp2626p
```

/usr/lib/.COREterm/h/hp262x
/usr/lib/.COREterm/h/hp2640a
/usr/lib/.COREterm/h/hp2640b
/usr/lib/.COREterm/h/hp2644a
/usr/lib/.COREterm/h/hp2645
/usr/lib/.COREterm/h/hp2648
/usr/lib/.COREterm/h/hp2648a
/usr/lib/.COREterm/h/hp45
/usr/lib/.COREterm/i/intext
/usr/lib/.COREterm/i/intext2
/usr/lib/.COREterm/i/intextii
/usr/lib/.COREterm/l/la120
/usr/lib/.COREterm/l/lp
/usr/lib/.COREterm/l/lpr
/usr/lib/.COREterm/p/pe550
/usr/lib/.COREterm/p/pe6100
/usr/lib/.COREterm/p/print
/usr/lib/.COREterm/p/printer
/usr/lib/.COREterm/p/printing
/usr/lib/.COREterm/p/pt505
/usr/lib/.COREterm/r/regent100
/usr/lib/.COREterm/r/regent20
/usr/lib/.COREterm/r/regent200
/usr/lib/.COREterm/r/regent25
/usr/lib/.COREterm/r/regent40
/usr/lib/.COREterm/r/regent60
/usr/lib/.COREterm/t/tek
/usr/lib/.COREterm/t/tek4012
/usr/lib/.COREterm/t/tek4013
/usr/lib/.COREterm/t/tek4014
/usr/lib/.COREterm/t/tek4015
/usr/lib/.COREterm/t/tek4023
/usr/lib/.COREterm/t/tek4024
/usr/lib/.COREterm/t/tek4025
/usr/lib/.COREterm/t/tek4027
/usr/lib/.COREterm/t/tek4112
/usr/lib/.COREterm/t/teletec
/usr/lib/.COREterm/t/televideo950
/usr/lib/.COREterm/t/tex
/usr/lib/.COREterm/t/tty4424
/usr/lib/.COREterm/t/tty4424-2
/usr/lib/.COREterm/t/tty4426
/usr/lib/.COREterm/t/tty5410
/usr/lib/.COREterm/t/tty5420
/usr/lib/.COREterm/t/tty5425
/usr/lib/.COREterm/t/tty5620

18 October 1989

```
                    /usr/lib/.COREterm/t/ttydmd
                    /usr/lib/.COREterm/t/tvi925
                    /usr/lib/.COREterm/t/tvi950
                    /usr/lib/.COREterm/v/viewpoint
                    /usr/lib/.COREterm/v/vt100
                    /usr/lib/.COREterm/v/vt100-am
                    /usr/lib/.COREterm/v/vt52
                    /usr/lib/5310
T                   /usr/lib/accept
                    /usr/lib/assist/bin/mecho
                    /usr/lib/assist/bin/mforms
                    /usr/lib/assist/bin/mscript
                    /usr/lib/assist/bin/msearch
                    /usr/lib/assist/bin/msetup
                    /usr/lib/assist/bin/tsetup
                    /usr/lib/assist/lib/forms/Batch.fs
                    /usr/lib/assist/lib/forms/Cflist.fs
                    /usr/lib/assist/lib/forms/Compare.fs
                    /usr/lib/assist/lib/forms/Compile.fs
                    /usr/lib/assist/lib/forms/Create.fs
                    /usr/lib/assist/lib/forms/Debug.fs
                    /usr/lib/assist/lib/forms/Directory.fs
                    /usr/lib/assist/lib/forms/Edit.fs
                    /usr/lib/assist/lib/forms/Electr.fs
                    /usr/lib/assist/lib/forms/Find.fs
                    /usr/lib/assist/lib/forms/Format.fs
                    /usr/lib/assist/lib/forms/Look.fs
                    /usr/lib/assist/lib/forms/Maint.fs
                    /usr/lib/assist/lib/forms/Perm.fs
                    /usr/lib/assist/lib/forms/Preproc.fs
                    /usr/lib/assist/lib/forms/Prog.fs
                    /usr/lib/assist/lib/forms/Setting.fs
                    /usr/lib/assist/lib/forms/Sorting.fs
                    /usr/lib/assist/lib/forms/Stats.fs
                    /usr/lib/assist/lib/forms/System.fs
                    /usr/lib/assist/lib/forms/TOP.fs
                    /usr/lib/assist/lib/forms/ar.fs
                    /usr/lib/assist/lib/forms/ar.val
                    /usr/lib/assist/lib/forms/ar_d.fs
                    /usr/lib/assist/lib/forms/ar_m.fs
                    /usr/lib/assist/lib/forms/ar_p.fs
                    /usr/lib/assist/lib/forms/ar_q.fs
                    /usr/lib/assist/lib/forms/ar_r.fs
                    /usr/lib/assist/lib/forms/ar_t.fs
                    /usr/lib/assist/lib/forms/ar_x.fs
                    /usr/lib/assist/lib/forms/assistwalk.fs
```

/usr/lib/assist/lib/forms/awk.fs
/usr/lib/assist/lib/forms/bfs.fs
/usr/lib/assist/lib/forms/cal.fs
/usr/lib/assist/lib/forms/cat.fs
/usr/lib/assist/lib/forms/cc.fs
/usr/lib/assist/lib/forms/cccomp.fs
/usr/lib/assist/lib/forms/cctyp.fs
/usr/lib/assist/lib/forms/cd.fs
/usr/lib/assist/lib/forms/chmod.fs
/usr/lib/assist/lib/forms/chmoda.fs
/usr/lib/assist/lib/forms/chmods.fs
/usr/lib/assist/lib/forms/chown.fs
/usr/lib/assist/lib/forms/cmp.fs
/usr/lib/assist/lib/forms/comm.fs
/usr/lib/assist/lib/forms/cp.fs
/usr/lib/assist/lib/forms/cpio.fs
/usr/lib/assist/lib/forms/cpioi.fs
/usr/lib/assist/lib/forms/cpioo.fs
/usr/lib/assist/lib/forms/cpiop.fs
/usr/lib/assist/lib/forms/crypt.fs
/usr/lib/assist/lib/forms/csplit.fs
/usr/lib/assist/lib/forms/cu.fs
/usr/lib/assist/lib/forms/cut.fs
/usr/lib/assist/lib/forms/d_help_c
/usr/lib/assist/lib/forms/d_help_m
/usr/lib/assist/lib/forms/d_help_p
/usr/lib/assist/lib/forms/d_help_t
/usr/lib/assist/lib/forms/date.fs
/usr/lib/assist/lib/forms/dc.fs
/usr/lib/assist/lib/forms/df.fs
/usr/lib/assist/lib/forms/diff.fs
/usr/lib/assist/lib/forms/dircmp.fs
/usr/lib/assist/lib/forms/du.fs
/usr/lib/assist/lib/forms/echo.fs
/usr/lib/assist/lib/forms/ed.fs
/usr/lib/assist/lib/forms/egrep.fs
/usr/lib/assist/lib/forms/env.fs
/usr/lib/assist/lib/forms/fgrep.fs
/usr/lib/assist/lib/forms/file.fs
/usr/lib/assist/lib/forms/find.fs
/usr/lib/assist/lib/forms/g_help_c
/usr/lib/assist/lib/forms/g_help_m
/usr/lib/assist/lib/forms/g_help_p
/usr/lib/assist/lib/forms/g_help_t
/usr/lib/assist/lib/forms/grep.fs
/usr/lib/assist/lib/forms/id.fs

/usr/lib/assist/lib/forms/kill.fs
/usr/lib/assist/lib/forms/ld.fs
/usr/lib/assist/lib/forms/lex.fs
/usr/lib/assist/lib/forms/ln.fs
/usr/lib/assist/lib/forms/lp.fs
/usr/lib/assist/lib/forms/lpstat.fs
/usr/lib/assist/lib/forms/ls.fs
/usr/lib/assist/lib/forms/lscomp.fs
/usr/lib/assist/lib/forms/lstyp.fs
/usr/lib/assist/lib/forms/mail.fs
/usr/lib/assist/lib/forms/make.fs
/usr/lib/assist/lib/forms/mesg.fs
/usr/lib/assist/lib/forms/mkdir.fs
/usr/lib/assist/lib/forms/mm.fs
/usr/lib/assist/lib/forms/mmcomp.fs
/usr/lib/assist/lib/forms/mmtyp.fs
/usr/lib/assist/lib/forms/mv.fs
/usr/lib/assist/lib/forms/nohup.fs
/usr/lib/assist/lib/forms/nroff.fs
/usr/lib/assist/lib/forms/od.fs
/usr/lib/assist/lib/forms/passwd.fs
/usr/lib/assist/lib/forms/paste.fs
/usr/lib/assist/lib/forms/pg.fs
/usr/lib/assist/lib/forms/pr.fs
/usr/lib/assist/lib/forms/prcomp.fs
/usr/lib/assist/lib/forms/prtyp.fs
/usr/lib/assist/lib/forms/ps.fs
/usr/lib/assist/lib/forms/pu_menu.fs
/usr/lib/assist/lib/forms/pwd.fs
/usr/lib/assist/lib/forms/rm.fs
/usr/lib/assist/lib/forms/rmdir.fs
/usr/lib/assist/lib/forms/sdb.fs
/usr/lib/assist/lib/forms/sdbwalk.fs
/usr/lib/assist/lib/forms/sed.fs
/usr/lib/assist/lib/forms/sh.fs
/usr/lib/assist/lib/forms/shell.lpstat
/usr/lib/assist/lib/forms/size.fs
/usr/lib/assist/lib/forms/sort.fs
/usr/lib/assist/lib/forms/sortc.fs
/usr/lib/assist/lib/forms/sorttyp.fs
/usr/lib/assist/lib/forms/spell.fs
/usr/lib/assist/lib/forms/split.fs
/usr/lib/assist/lib/forms/strip.fs
/usr/lib/assist/lib/forms/stty.fs
/usr/lib/assist/lib/forms/sttyrep.fs
/usr/lib/assist/lib/forms/sttyset.fs

/usr/lib/assist/lib/forms/su.fs
/usr/lib/assist/lib/forms/tail.fs
/usr/lib/assist/lib/forms/taili.fs
/usr/lib/assist/lib/forms/tailn.fs
/usr/lib/assist/lib/forms/umask.fs
/usr/lib/assist/lib/forms/uname.fs
/usr/lib/assist/lib/forms/unixwalk.fs
/usr/lib/assist/lib/forms/uucp.fs
/usr/lib/assist/lib/forms/uucpl.fs
/usr/lib/assist/lib/forms/uucpr.fs
/usr/lib/assist/lib/forms/vi.3.0
/usr/lib/assist/lib/forms/vi.fs
/usr/lib/assist/lib/forms/viwalk.fs
/usr/lib/assist/lib/forms/wc.fs
/usr/lib/assist/lib/forms/who.fs
/usr/lib/assist/lib/forms/yacc.fs
/usr/lib/assist/lib/scripts/as.build
/usr/lib/assist/lib/scripts/sdb.build
/usr/lib/assist/lib/scripts/t.ascf.z
/usr/lib/assist/lib/scripts/t.ascontinue
/usr/lib/assist/lib/scripts/t.asending
/usr/lib/assist/lib/scripts/t.ashelp.z
/usr/lib/assist/lib/scripts/t.asintro
/usr/lib/assist/lib/scripts/t.asmenu.z
/usr/lib/assist/lib/scripts/t.aspopmenu.z
/usr/lib/assist/lib/scripts/t.assubs.z
/usr/lib/assist/lib/scripts/t.sdbbreak.z
/usr/lib/assist/lib/scripts/t.sdbcontinue
/usr/lib/assist/lib/scripts/t.sdbcore.z
/usr/lib/assist/lib/scripts/t.sdbending
/usr/lib/assist/lib/scripts/t.sdbexec.z
/usr/lib/assist/lib/scripts/t.sdbintro.z
/usr/lib/assist/lib/scripts/t.sdbmulti.z
/usr/lib/assist/lib/scripts/t.sdbsdbend
/usr/lib/assist/lib/scripts/t.sdbsearch.z
/usr/lib/assist/lib/scripts/t.sdbsubs.z
/usr/lib/assist/lib/scripts/t.uncontinue.z
/usr/lib/assist/lib/scripts/t.unending.z
/usr/lib/assist/lib/scripts/t.unfiles.z
/usr/lib/assist/lib/scripts/t.unintro.z
/usr/lib/assist/lib/scripts/t.unintroend.z
/usr/lib/assist/lib/scripts/t.unpipes.z
/usr/lib/assist/lib/scripts/t.unrediri.z
/usr/lib/assist/lib/scripts/t.unrediro.z
/usr/lib/assist/lib/scripts/t.unstdio.z
/usr/lib/assist/lib/scripts/t.unsubs.z

18 October 1989

```
         /usr/lib/assist/lib/scripts/t.viadd.z
         /usr/lib/assist/lib/scripts/t.vicontinue
         /usr/lib/assist/lib/scripts/t.vicursor.z
         /usr/lib/assist/lib/scripts/t.videlete.z
         /usr/lib/assist/lib/scripts/t.videmo.z
         /usr/lib/assist/lib/scripts/t.viending
         /usr/lib/assist/lib/scripts/t.viintro
         /usr/lib/assist/lib/scripts/t.viintroend
         /usr/lib/assist/lib/scripts/t.vimanip.z
         /usr/lib/assist/lib/scripts/t.viopts.z
         /usr/lib/assist/lib/scripts/t.visubs.z
         /usr/lib/assist/lib/scripts/t.viundo.z
         /usr/lib/assist/lib/scripts/un.build
         /usr/lib/assist/lib/scripts/vi.build
         /usr/lib/assist/lib/scripts/vicommands
         /usr/lib/assist/lib/search/searchlist
         /usr/lib/assist/lib/search/unix/matchpairs
         /usr/lib/assist/lib/search/unix/search-file
         /usr/lib/assist/lib/setup/assistrc
         /usr/lib/assist/lib/setup/messages
         /usr/lib/assist/lib/setup/t.altchar
         /usr/lib/assist/lib/setup/t.basic
         /usr/lib/assist/lib/setup/t.fkeys
         /usr/lib/assist/lib/setup/t.standout
         /usr/lib/calprog
T        /usr/lib/cron/.proto
T        /usr/lib/cron/at.allow
T        /usr/lib/cron/cron.allow
T        /usr/lib/cron/log
T        /usr/lib/cron/queuedefs
         /usr/lib/diff3prog
         /usr/lib/diffh
         /usr/lib/expreserve
         /usr/lib/exrecover
         /usr/lib/exstrings
         /usr/lib/getoptcvt
         /usr/lib/graf/ttoc.d/ed.notoc
         /usr/lib/graf/ttoc.d/ed.toc
         /usr/lib/graf/ttoc.d/ed.ttoc.t
         /usr/lib/graf/whatis/abs
         /usr/lib/graf/whatis/af
         /usr/lib/graf/whatis/bar
         /usr/lib/graf/whatis/bel
         /usr/lib/graf/whatis/bucket
         /usr/lib/graf/whatis/ceil
         /usr/lib/graf/whatis/cor
```

/usr/lib/graf/whatis/cusum
/usr/lib/graf/whatis/cvrtopt
/usr/lib/graf/whatis/dtoc
/usr/lib/graf/whatis/erase
/usr/lib/graf/whatis/exp
/usr/lib/graf/whatis/floor
/usr/lib/graf/whatis/gamma
/usr/lib/graf/whatis/gas
/usr/lib/graf/whatis/gd
/usr/lib/graf/whatis/ged
/usr/lib/graf/whatis/gtop
/usr/lib/graf/whatis/hardcopy
/usr/lib/graf/whatis/hilo
/usr/lib/graf/whatis/hist
/usr/lib/graf/whatis/hpd
/usr/lib/graf/whatis/label
/usr/lib/graf/whatis/list
/usr/lib/graf/whatis/log
/usr/lib/graf/whatis/lreg
/usr/lib/graf/whatis/mean
/usr/lib/graf/whatis/mod
/usr/lib/graf/whatis/pair
/usr/lib/graf/whatis/pd
/usr/lib/graf/whatis/pie
/usr/lib/graf/whatis/plot
/usr/lib/graf/whatis/point
/usr/lib/graf/whatis/power
/usr/lib/graf/whatis/prime
/usr/lib/graf/whatis/prod
/usr/lib/graf/whatis/ptog
/usr/lib/graf/whatis/qsort
/usr/lib/graf/whatis/quit
/usr/lib/graf/whatis/rand
/usr/lib/graf/whatis/rank
/usr/lib/graf/whatis/remcom
/usr/lib/graf/whatis/root
/usr/lib/graf/whatis/round
/usr/lib/graf/whatis/siline
/usr/lib/graf/whatis/sin
/usr/lib/graf/whatis/subset
/usr/lib/graf/whatis/td
/usr/lib/graf/whatis/tekset
/usr/lib/graf/whatis/title
/usr/lib/graf/whatis/total
/usr/lib/graf/whatis/ttoc
/usr/lib/graf/whatis/var

```
/usr/lib/graf/whatis/vtoc
/usr/lib/graf/whatis/whatis
/usr/lib/graf/whatis/yoo
/usr/lib/help/HELPLOG
/usr/lib/help/admgloss
/usr/lib/help/admstart
/usr/lib/help/checklen
/usr/lib/help/db/descriptions.a
/usr/lib/help/db/examples.a
/usr/lib/help/db/glossary.a
/usr/lib/help/db/options.a
/usr/lib/help/db/screens.a
/usr/lib/help/db/tables/display
/usr/lib/help/db/tables/keywords
/usr/lib/help/db/tables/responses
/usr/lib/help/defnlen
/usr/lib/help/delete
/usr/lib/help/editcmd
/usr/lib/help/extract
/usr/lib/help/fetch
/usr/lib/help/glossary
/usr/lib/help/helpclean
/usr/lib/help/interact
/usr/lib/help/keysrch
/usr/lib/help/list
/usr/lib/help/locking
/usr/lib/help/replace
/usr/lib/hp2631a
/usr/lib/layersys/lsys.8;7;3
/usr/lib/layersys/relogin
/usr/lib/layersys/set_enc.j
/usr/lib/layersys/wtinit
/usr/lib/lib.b
/usr/lib/lib300.a
/usr/lib/lib300s.a
/usr/lib/lib4014.a
/usr/lib/lib450.a
/usr/lib/libcurses.a
/usr/lib/libmls.a
/usr/lib/libplot.a
/usr/lib/libtermcap.a
/usr/lib/libtermlib.a
/usr/lib/libvt0.a
/usr/lib/libwindows.a
/usr/lib/llib-lcurses
/usr/lib/llib-lcurses.l
```

T	/usr/lib/lpadmin
T	/usr/lib/lpmove
T	/usr/lib/lpsched
T	/usr/lib/lpshut
T	/usr/lib/mailx/mailx.help
	/usr/lib/mailx/mailx.help.~
T	/usr/lib/mailx/rmmail
T	/usr/lib/mv_dir
T	/usr/lib/pgmark
	/usr/lib/pprx
	/usr/lib/prx
	/usr/lib/readme/assist
T	/usr/lib/reject
T	/usr/lib/sa/sa1
T	/usr/lib/sa/sa2
T	/usr/lib/sa/sadc
T	/usr/lib/scsi/format
T	/usr/lib/scsi/hdefix
T	/usr/lib/scsi/labelfsname
T	/usr/lib/scsi/qiccopy
T	/usr/lib/scsi/sd00.0
T	/usr/lib/scsi/selectscsi
T	/usr/lib/scsi/tapecntl
T	/usr/lib/scsi/tc.index
	/usr/lib/spell/compress
	/usr/lib/spell/hashcheck
	/usr/lib/spell/hashmake
	/usr/lib/spell/hlista
	/usr/lib/spell/hlistb
	/usr/lib/spell/hstop
	/usr/lib/spell/spellhist
	/usr/lib/spell/spellin
	/usr/lib/spell/spellprog
	/usr/lib/t300
	/usr/lib/t300s
	/usr/lib/t4014
	/usr/lib/t450
	/usr/lib/tabset/3101
	/usr/lib/tabset/beehive
	/usr/lib/tabset/std
	/usr/lib/tabset/teleray
	/usr/lib/tabset/vt100
	/usr/lib/tabset/xerox1720
	/usr/lib/terminfo/1/1620
	/usr/lib/terminfo/1/1640
	/usr/lib/terminfo/2/2621

18 October 1989

/usr/lib/terminfo/2/2621-wl
/usr/lib/terminfo/2/2621A
/usr/lib/terminfo/2/2621a
/usr/lib/terminfo/2/2622
/usr/lib/terminfo/2/2622a
/usr/lib/terminfo/2/2623
/usr/lib/terminfo/2/2623a
/usr/lib/terminfo/2/2624
/usr/lib/terminfo/2/2624-4p
/usr/lib/terminfo/2/2624a
/usr/lib/terminfo/2/2624b
/usr/lib/terminfo/2/2626
/usr/lib/terminfo/2/2626A
/usr/lib/terminfo/2/2626P
/usr/lib/terminfo/2/2626a
/usr/lib/terminfo/2/2626p
/usr/lib/terminfo/2/2640
/usr/lib/terminfo/2/2640a
/usr/lib/terminfo/2/2640b
/usr/lib/terminfo/2/2644a
/usr/lib/terminfo/2/2645
/usr/lib/terminfo/2/2648
/usr/lib/terminfo/2/2648A
/usr/lib/terminfo/2/2648a
/usr/lib/terminfo/3/31
/usr/lib/terminfo/3/36
/usr/lib/terminfo/3/3a
/usr/lib/terminfo/4/4012
/usr/lib/terminfo/4/4013
/usr/lib/terminfo/4/4014
/usr/lib/terminfo/4/4015
/usr/lib/terminfo/4/4023
/usr/lib/terminfo/4/4024
/usr/lib/terminfo/4/4025
/usr/lib/terminfo/4/4025cu
/usr/lib/terminfo/4/4027
/usr/lib/terminfo/4/4027cu
/usr/lib/terminfo/4/4112
/usr/lib/terminfo/4/4112-nd
/usr/lib/terminfo/4/4113
/usr/lib/terminfo/4/4114
/usr/lib/terminfo/4/4115
/usr/lib/terminfo/4/42
/usr/lib/terminfo/4/4410
/usr/lib/terminfo/4/4415
/usr/lib/terminfo/4/4418

```
        /usr/lib/terminfo/4/4424
        /usr/lib/terminfo/4/4424-1
        /usr/lib/terminfo/4/4424-2
        /usr/lib/terminfo/4/4424-3
    T   /usr/lib/terminfo/4/4425
        /usr/lib/terminfo/4/4426
        /usr/lib/terminfo/4/450
        /usr/lib/terminfo/5/513bct
        /usr/lib/terminfo/5/5410
        /usr/lib/terminfo/5/5418
        /usr/lib/terminfo/5/5420
        /usr/lib/terminfo/5/5420_2
        /usr/lib/terminfo/5/5425
        /usr/lib/terminfo/5/5620
        /usr/lib/terminfo/6/610bct
    T   /usr/lib/terminfo/6/630
        /usr/lib/terminfo/9/925
        /usr/lib/terminfo/9/950
        /usr/lib/terminfo/A/ATT4410
        /usr/lib/terminfo/A/ATT4415
        /usr/lib/terminfo/A/ATT4418
        /usr/lib/terminfo/A/ATT4424
        /usr/lib/terminfo/A/ATT4424-2
    T   /usr/lib/terminfo/A/ATT4425
        /usr/lib/terminfo/A/ATT4426
        /usr/lib/terminfo/A/ATT513
        /usr/lib/terminfo/A/ATT5410
        /usr/lib/terminfo/A/ATT5418
        /usr/lib/terminfo/A/ATT5420
        /usr/lib/terminfo/A/ATT5420_2
        /usr/lib/terminfo/A/ATT5425
        /usr/lib/terminfo/A/ATT5620
        /usr/lib/terminfo/A/ATT610BCT
        /usr/lib/terminfo/A/ATTPT505
        /usr/lib/terminfo/a/adm3
        /usr/lib/terminfo/a/adm31
        /usr/lib/terminfo/a/adm36
        /usr/lib/terminfo/a/adm3a
        /usr/lib/terminfo/a/adm42
        /usr/lib/terminfo/a/att4410
        /usr/lib/terminfo/a/att4415
        /usr/lib/terminfo/a/att4418
        /usr/lib/terminfo/a/att4424
        /usr/lib/terminfo/a/att4424-2
    T   /usr/lib/terminfo/a/att4425
        /usr/lib/terminfo/a/att4426
```

```
/usr/lib/terminfo/a/att513
/usr/lib/terminfo/a/att5410
/usr/lib/terminfo/a/att5418
/usr/lib/terminfo/a/att5420
/usr/lib/terminfo/a/att5420_2
/usr/lib/terminfo/a/att5425
/usr/lib/terminfo/a/att5620
/usr/lib/terminfo/a/att610bct
/usr/lib/terminfo/a/attpt505
/usr/lib/terminfo/b/bantam
/usr/lib/terminfo/c/c100
/usr/lib/terminfo/c/c100-4p
/usr/lib/terminfo/c/c104
/usr/lib/terminfo/c/c108
/usr/lib/terminfo/c/c108-8p
/usr/lib/terminfo/c/concept
/usr/lib/terminfo/c/concept100
/usr/lib/terminfo/c/concept108
/usr/lib/terminfo/c/concept108-8p
/usr/lib/terminfo/d/decwriter
/usr/lib/terminfo/d/diablo
/usr/lib/terminfo/d/dmd
/usr/lib/terminfo/d/dumb
/usr/lib/terminfo/d/dw
/usr/lib/terminfo/d/dw1
/usr/lib/terminfo/d/dw2
/usr/lib/terminfo/d/dw3
/usr/lib/terminfo/d/dw4
/usr/lib/terminfo/h/hp2621
/usr/lib/terminfo/h/hp2621-wl
/usr/lib/terminfo/h/hp2621A
/usr/lib/terminfo/h/hp2621a
/usr/lib/terminfo/h/hp2622
/usr/lib/terminfo/h/hp2622a
/usr/lib/terminfo/h/hp2623
/usr/lib/terminfo/h/hp2623a
/usr/lib/terminfo/h/hp2624
/usr/lib/terminfo/h/hp2624a
/usr/lib/terminfo/h/hp2624b
/usr/lib/terminfo/h/hp2624b-4p
/usr/lib/terminfo/h/hp2626
/usr/lib/terminfo/h/hp2626a
/usr/lib/terminfo/h/hp2626p
/usr/lib/terminfo/h/hp262x
/usr/lib/terminfo/h/hp2640a
/usr/lib/terminfo/h/hp2640b
```

/usr/lib/terminfo/h/hp2644a
/usr/lib/terminfo/h/hp2645
/usr/lib/terminfo/h/hp2648
/usr/lib/terminfo/h/hp2648a
/usr/lib/terminfo/h/hp45
/usr/lib/terminfo/i/intext
/usr/lib/terminfo/i/intext2
/usr/lib/terminfo/i/intextii
/usr/lib/terminfo/l/la120
/usr/lib/terminfo/l/lp
/usr/lib/terminfo/l/lpr
/usr/lib/terminfo/p/pe550
/usr/lib/terminfo/p/pe6100
/usr/lib/terminfo/p/print
/usr/lib/terminfo/p/printer
/usr/lib/terminfo/p/printing
/usr/lib/terminfo/p/pt505
/usr/lib/terminfo/r/regent100
/usr/lib/terminfo/r/regent20
/usr/lib/terminfo/r/regent200
/usr/lib/terminfo/r/regent25
/usr/lib/terminfo/r/regent40
/usr/lib/terminfo/r/regent60
/usr/lib/terminfo/t/tek
/usr/lib/terminfo/t/tek4012
/usr/lib/terminfo/t/tek4013
/usr/lib/terminfo/t/tek4014
/usr/lib/terminfo/t/tek4015
/usr/lib/terminfo/t/tek4023
/usr/lib/terminfo/t/tek4024
/usr/lib/terminfo/t/tek4025
/usr/lib/terminfo/t/tek4027
/usr/lib/terminfo/t/tek4112
/usr/lib/terminfo/t/teletec
/usr/lib/terminfo/t/televideo950
/usr/lib/terminfo/t/tex
/usr/lib/terminfo/t/tty4424
/usr/lib/terminfo/t/tty4424-2
/usr/lib/terminfo/t/tty4426
/usr/lib/terminfo/t/tty5410
/usr/lib/terminfo/t/tty5420
/usr/lib/terminfo/t/tty5425
/usr/lib/terminfo/t/tty5620
/usr/lib/terminfo/t/ttydmd
/usr/lib/terminfo/t/tvi925
/usr/lib/terminfo/t/tvi950

	/usr/lib/terminfo/v/viewpoint
	/usr/lib/terminfo/v/vt100
	/usr/lib/terminfo/v/vt100-am
	/usr/lib/terminfo/v/vt52
	/usr/lib/unittab
	/usr/lib/vplot
	/usr/options/630mls.name
	/usr/options/assist.name
	/usr/options/calc.name
	/usr/options/dfm.name
	/usr/options/ed.name
	/usr/options/eports.name
	/usr/options/graph.name
	/usr/options/help.name
	/usr/options/ipc.name
	/usr/options/lp.name
	/usr/options/mls.name
	/usr/options/perf.name
	/usr/options/shell.name
	/usr/options/spell.name
	/usr/options/sys.name
	/usr/options/sysadm.name
	/usr/options/term.name
	/usr/options/terminf.name
	/usr/options/usrenv.name
	/usr/options/windowing.name
T	/usr/spool/cron/crontabs/adm
T	/usr/spool/cron/crontabs/root
T	/usr/spool/cron/crontabs/sys
T	/usr/spool/cron/crontabs/sysadm
T	/usr/spool/lp/log
	/usr/spool/lp/member/lp1
	/usr/spool/lp/model/1640
T	/usr/spool/lp/model/5310
	/usr/spool/lp/model/dqp10
	/usr/spool/lp/model/dumb
	/usr/spool/lp/model/f450
	/usr/spool/lp/model/hp
	/usr/spool/lp/model/lqp40
	/usr/spool/lp/model/pprx
	/usr/spool/lp/model/prx
T	/usr/spool/lp/oldlog
T	/usr/spool/lp/outputq
T	/usr/spool/lp/pstatus
T	/usr/spool/lp/qstatu

TRUSTED PROCESS LIST

The following is a list of the trusted processes which are necessary for the operation of a System V/MLS system. The processes listed below do not enforce the system security policy; they are simply trusted not to violate that policy and to operate correctly when they are executed by a system administrator.

NOTE: The abbreviation "SA" as used below refers to a Simplified Administration Utility shell script.

NOTE: The number preceding each TCB item is the reason for its inclusion:

1. setuid/setgid to an administrative ID,

2. program/data file run or printed by the administrator,

3. directory requiring protection,

4. TCB data file.

3 / - the root directory.

3 /bck - the bck directory.

3 /bin - the bin directory.

2 /bin/ar - archive and library maintainer for portable archives.

2 /bin/basename, /bin/dirname - deliver portions of pathnames.

2 /bin/cat - concatenate and print files.

2 /bin/chmod - change mode of a file.

2 /bin/chown, /bin/chgrp - change owner or group of a file.

1 /bin/chpriv - change privilege associated with a file.

2 /bin/cp, /bin/ln, /bin/mv - copy, link, or move files.

2 /bin/cpio - copy file archives in and out.

2 /bin/date - print and set the date.

1 /bin/df - report number of free disk blocks.

2 /bin/diff - differential file comparator.

2 /bin/du - summarize disk usage.

2 /bin/echo - echo arguments.

2 /bin/ed - text editor.

2 /bin/expr - evaluate arguments as an expression.

2 /bin/file - determine file type.

2 /bin/find - find files.

2 /bin/grep - search a file for a pattern.

2   /bin/ipcrm - remove a message queue, semaphore set, or shared memory id.

1 /bin/ipcs - report interprocess communication facilities status.

2 /bin/kill - terminate a process.

1 /bin/labels - print security labels in various readable formats.

1 /bin/login - sign on.

2 /bin/ls - list contents of a directory.

1 /bin/mail - send mail to users or read mail.

2 /bin/mkdir - make a directory.

1 /bin/newgrp - log into a new group.

1 /bin/newpriv - change user's current privilege.

2 /bin/nice - run a command at low priority.

2 /bin/nm - print name list of common object file.

2 /bin/nohup - run a command immune to hangups and quits.

2 /bin/od - octal dump.

1 /bin/passwd - change login password.

1 /bin/ps - report process status.

2 /bin/pwd - working directory name.

2 /bin/rm, /bin/rmdir - remove files or directories.

2 /bin/setpgrp - set process group ID.

2 /bin/sh - shell, the standard command programming language.

2 /bin/sleep - suspend execution for an interval.

2 /bin/sort - sort and/or merge files.

2 /bin/stty - set the options for a terminal.

1 /bin/su - become superuser.

2 /bin/sum - print checksum and block count of a file.

2 /bin/sync - update super block.

2 /bin/tee - pipe fitting.

2 /bin/time - time a command.

2 /bin/touch - update access and modification times of a file.

2 /bin/true, /bin/false - provide truth values.

2 /bin/tty - get the name of a terminal.

2 /bin/u3b, /bin/pdp11, /bin/vax, /bin/u370,

2 /bin/u3b5, /bin/u3b15, /bin/u3b2

2 /bin/uname - print name of current UNIX system.

2 /bin/wc - word count.

2 /bin/who - who is on the system.

2 /bin/write - write to another user.

3 /boot - boot directory for drivers.

2 /boot/CONLOG - console login.

2 /boot/DISP - process dispatcher.

2 /boot/HDELOG - hard disk error log.

2 /boot/IDISK - internal disk.

2 /boot/IUART - universal asynchronous receive transmitter.

2 /boot/KERNEL - kernel.

2 /boot/MAU - math acceleration unit.

2 /boot/MEM - memory pseudodevice.

2 /boot/PORTS - ports board.

2 /boot/S5 - s5 filesystem.

2 /boot/SCSI - Small Computer System Interfaces.

2 /boot/STUBS - functions referenced in kernel, but not required by system.

2 /boot/VCACHE - virtual cache.

3 /dev - the device directory.

3 /dev/SA - SA device directory.

2 /dev/boot - boot device.

2 /dev/consbuf - console buffer device.

2 /dev/conslog - console log device.

2 /dev/console - console.

2 /dev/contty - contty device.

2 /dev/dsk/c0d0s0 - partition of disk.

2 /dev/dsk/c0d0s1 - partition of disk.

2 /dev/dsk/c0d0s2 - partition of disk.

2 /dev/dsk/c0d0s3 - partition of disk.

2 /dev/dsk/c0d0s4 - partition of disk.

2 /dev/dsk/c0d0s5 - partition of disk.

2 /dev/dsk/c0d0s6 - partition of disk.

2 /dev/dsk/c0d0s7 - partition of disk.

2 /dev/root - root pseudodevice.

2 /dev/scsi - scsi pseudodevice.

2 /dev/syscon - system console device.

2 /dev/systty - systty device.

2 /dev/tty - tty pseudodevice.

2 /dgmon - program to run diagnostics.

3 /dgn - diagnostics directory.

2 /dgn/MAU - math acceleration unit.

2 /dgn/PORTS - ports board.

2 /dgn/SBD - system board.

2 /dgn/VCACHE - virtual cache.

2 /dgn/X.MAU - math acceleration unit (additional).

2 /dgn/X.PORTS - ports board (additional).

2 /dgn/X.SBD - system board (additional).

2 /dgn/X.SCSI - Small Computer System Interface (additional).

2 /dgn/X.VCACHE - virtual cache (additional).

3 /edt - equipped device table directory.

3 /etc - etc directory.

2 /etc/TIMEZONE - set timezone environment to the default for the machine.

2 /etc/brc, /etc/bcheckrc - System initialization procedures.

2 /etc/bupsched - print time for backup schedule reminder.

2 /etc/bzapunix - force self-configuration bootstrap upon reboot.

4 /etc/checklist - list of filesystems processed by fsck.

2 /etc/chroot - Change root directory for a command.

2 /etc/ckauto - Find if the UNIX system was reconfigured at boot time.

2 /etc/ckbupscd - Check file system backup schedule.

2 /etc/clri - Clear i-node.

2 /etc/crash - Examine system images.

2 /etc/cron - Clock daemon.

2 /etc/dcopy - copy file systems for optimal access time.

1 /etc/devnm - Device name.

2 /etc/disketteparm - parameters for file systems made on diskette.

2 /etc/disks - Adds /dev/entries for hard disks in Equipped Device Table.

2 /etc/drvinstall - Install/uninstall a driver.

2 /etc/editsa - Add/delete entry from software application file.

2 /etc/edittbl - Edit edt_data file.

2 /etc/errdump - Print error log.

2 /etc/ff - list file names and statistics for a file system.

2 /etc/finc - fast incremental backup.

2 /etc/fltboot - Set NVRAM parameters for floating boot.

2 /etc/fmtflop - Physically format diskettes.

2 /etc/fmthard - Populates the Volume Table of Contents (VTOC) on hard disks.

2 /etc/format - formats floppy diskettes.

2 /etc/frec - fast recove of files by inode numbers from a tape.

2 /etc/fsck, /etc/dfsck - Check and repair file systems.

2 /etc/fsdb - file system debugger.

2 /etc/fsstat - Report file system status.

2 /etc/fstyp - Determine file system identifier.

3 /etc/fstyp.d - fstyp.d directory.

2 /etc/fstyp.d/s5fstyp - s5 filesystem type.

2 /etc/fuser - identify processes using a file or file structure.

2 /etc/getmajor - Print slot/major number(s) of hardware devices.

2 /etc/getty - Set terminal type, modes, speed, and line discipline.

4 /etc/gettydefs - used by /etc/getty to set up terminal connections.

4 /etc/group - sanitized privilege membership information file.

2 /etc/hdeadd - Add/delete hdelog (Hard Disk Error Log) reports.

2 /etc/hdefix - Report or change bad block mapping.

2 /etc/hdelogger - Hard Disk Error status report command and Log Daemon.

2 /etc/helpadm - make changes to the IhelpR database.

2 /etc/init, /etc/telinit - Process control initialization.

3 /etc/init.d - init.d directory.

2 /etc/init.d/ANNOUNCE - prints out change of state message.

2 /etc/init.d/MOUNTFSYS - calls /etc/mountall to mount file systems according to information
         in /etc/fstab.

2 /etc/init.d/RMTMPFILES - cleans up tmp directories.

2 /etc/init.d/autoconfig - generates a new unix file using /etc/mkunix.

2 /etc/init.d/cron - starts/stops cron.

2 /etc/init.d/disks - creates /dev entries for new disks.

2 /etc/init.d/firstcheck - prints welcome and setup message when machine is brought up for the
       first time.

2 /etc/init.d/sysetup - contains system setup requirements.

2 /etc/inittab - file containing processes to be run in various init states.

2 /etc/install - install commands.

2 /etc/killall - Kill all active processes.

2 /etc/labelit - Provide labels for file systems.

2 /etc/ldsysdump - load system dump from floppy diskettes.

2 /etc/link, /etc/unlink - link and unlink files and directories.

3 /etc/log - volcopy log directory - log of backup dumps.

4 /etc/magic - list of "magic" numbers for /bin/file command.

3   /etc/master.d - directory of device driver characteristics, external variable definitions, and
dependencies.

2 /etc/master.d/conlog - console log.

2 /etc/master.d/disp - process dispatcher parameter values.

2 /etc/master.d/gentty - generic tty.

2 /etc/master.d/hdelog - hard disk error logger.

2 /etc/master.d/idisk - internal disk.

2 /etc/master.d/iuart - universal asynchronous receive transmitter.

2 /etc/master.d/kernel - kernel.

2 /etc/master.d/mau - math acceleration unit.

2 /etc/master.d/mem - memory.

2 /etc/master.d/mls - MLS module.

2 /etc/master.d/ports - ports driver.

2 /etc/master.d/s5 - S5 file system.

2 /etc/master.d/sat - SAT module.

2 /etc/master.d/stubs - used to define defaults for functions referenced in the kernel, but not required by the system.

2 /etc/master.d/vcache - virtual cache.

2 /etc/mkboot - Convert an object file into a bootable object file.

2 /etc/mkfs - Construct a file system.

2 /etc/mknod - Build special file.

2 /etc/mkunix - Make a bootable system file with kernel and driver symbol tables.

4 /etc/mnttab - mount table; data used in checking and mounting file systems when system is booted.

2 /etc/motd - "message of the day" file displayed upon login.

2 /etc/mount, /etc/umount - Mount and unmount file systems.

2 /etc/mountall, /etc/umountall - Mount, unmount multiple file systems.

2 /etc/mvdir - move a directory.

2 /etc/ncheck - generate pathnames from i-numbers.

2 /etc/newboot - Load lboot and mboot onto the disk boot partition.

4 /etc/passwd - sanitized user information file.

2 /etc/ports - Create character device files and inittab entries.

2 /etc/prfld, /etc/prfstat, /etc/prfdc, /etc/prfsnap, /etc/prfpr - UNIX system profiler.

2 /etc/profile - initialization program run for all users upon logging in.

2 /etc/prtconf - Print system configuration.

3 /etc/prtconf.d - prtconf.d directory.

2 /etc/prtvtoc - Print the Volume Table of Contents (VTOC) of a device.

2 /etc/pump - Download B16 or X86 a.out file to a peripheral board.

2 /etc/pwck, /etc/grpck - password/group file checkers.

3 /etc/rc.d - directory of commands performed upon starting the system.

2 /etc/rc0 - Run commands performed to stop the operating system.

3 /etc/rc0.d - directory of commands to be performed before stoping the operating system.

2 /etc/rc2 - Run commands performed for multi-user environment.

3 /etc/rc2.d - directory of commands to be performed before entering multiuser mode.

2 /etc/rc2.d/S16satstart - calls /mls/bin/satstart.

2 /etc/rc2.d/S17mlsstart - calls /mls/bin/mlsstart.

2 /etc/rc2.d/S80errstart - starts error logging.

4 /etc/save.d/except - patterns of file names to be excluded from saving by savefiles.

2 /etc/savecpio - save file-systems in cpio format on removable media.

2 /etc/setclk - Set system time from hardware clock.

2 /etc/setmnt - Establish mount table.

2 /etc/shutdown - Shut down system, change system state.

3 /etc/shutdown.d - shutdown directory.

2 /etc/stdprofile - default standard .profile provided to a user.

2 /etc/swap - Swap administrative interface.

2 /etc/sysdef - output system definition.

2 /etc/system - defines device drivers used on machine.

2 /etc/uadmin - Administrative control.

4 /etc/utmp - user and accounting information.

2 /etc/volcopy - make literal copy of file system.

3 /etc/vtoc - vtoc directory.

2 /etc/wall - Write to all users.

1 /etc/whodo - who is doing what.

4 /etc/wtmp - user and accounting information.

2 /tilledt - fills equipped device table.

3 /install - install directory.

3 /lib - library directory.

2 /lib/lboot - defines address where UNIX will be loaded.

3 /lib/libp - libp library directory.

2 /lib/mboot - checks if /etc/system is newer than /unix and runs lboot.

3 /lib/pump - pump directory.

2 /lib/pump/ports - download and initialize device for ports board.

2 /lib/pump/ports.hpp - download and initialize device for highspeed peripheral ports.

3 /mls - mls directory.

3 /mls/bin - mls bin directory.

2 /mls/bin/LpSetup - define labels for use with lp.

2 /mls/bin/MailSetup - define labels for use with mail.

2 /mls/bin/mklbl - create a labels file from human- readable form.

2 /mls/bin/prlbl - print a labels file in human- readable form.

2 /mls/bin/group.cleanup - clean up removed privileges from /mls/group and remind file owners
        to remove/change files.

2 /mls/bin/satfmt - format the security audit trail data.

2 /mls/bin/satsave - satsave daemon; collects audit records.

2 /mls/bin/Permsetup - setup permissions on directories.

2 /mls/bin/mlsstart - set up multilevel directories for MLS.

2 /mls/bin/satstart - start the security audit trail.

2 /mls/bin/updatepwgr - maintain sanitized password and group files.

2 /mls/bin/sfsmap - file system map for audit trail from special device.

2 /mls/bin/SecadmSetup - define labels that may be declassified.

2 /mls/bin/UucpSetup - define labels for use with uucp.

2 /mls/bin/satstop - stop security audit trail.

2 /mls/bin/mkdevclr - make entry in device clearances database.

2 /mls/bin/rmdevclr - remove entry from device clearances database.

2 /mls/bin/sessions - administrative command displaying status of sessions database.

2 /mls/.mkgrp - file touched when group is added.

4 /mls/categories - definitions file for system categories.

4 /mls/clearances - file of user clearances.

/mls/cleardev - device clearances file.

4 /mls/levels - definitions file for system levels.

4 /mls/passwd - user information/password file.

4 /mls/group - privilege/group membership file.

4 /mls/labels - labels file.

4 /mls/sessions - sessions database directory

2 /mls/.mkuser - file touched when user is added.

3 /mnt - mount directory.

3 /save - save directory.

3 /shlib - shared library directory.

2 /shlib/libc_s - shared C library.

3 /tmp - the tmp directory.

3 /usr - the usr directory.

3 /usr/adm - administrative information directory.

3 /usr/adm/bin - administrative information query command directory.

3 /usr/admin - simplified administration (SA) directory.

2 /usr/admin/bupsched - SA to schedule backup reminder.

2 /usr/admin/checkfsys - SA to check a removable medium filesystem for errors.

2 /usr/admin/makefsys - SA to create a new filesystem on a removable medium.

3 /usr/admin/menu - SA menu directory.

3 /usr/admin/menu/diagnostics - SA diagnostics directory.

2 /usr/admin/menu/diagnostics/diskrepair - SA to advise about repairing errors on built-in disks.

2 /usr/admin/menu/diagnostics/diskreport - SA to report collected reading errors for built-in disks.

3 /usr/admin/menu/diskmgmt - SA disk management directory.

2 /usr/admin/menu/diskmgmt/cpdisk - SA to make exact copies of a removable medium.

2 /usr/admin/menu/diskmgmt/erase - SA to erase data from a removable medium.

2 /usr/admin/menu/diskmgmt/format - SA to format a removable medium.

October 18, 1989

2 /usr/admin/menu/diskmgmt/harddisk/display - SA to display hard disk partitioning.

2 /usr/admin/menu/diskmgmt/harddisk/format - SA to format a hard disk.

2 /usr/admin/menu/diskmgmt/harddisk/partitioning - SA to partition a hard disk.

2 /usr/admin/menu/diskmgmt/harddisk/rmdisk - SA to remove a hard disk.

2 /usr/admin/menu/filemgmt/backup - SA to backup files or sets of files.

2 /usr/admin/menu/filemgmt/diskuse - SA to display how much of the built-in disks is being used.

2 /usr/admin/menu/filemgmt/fileage - SA to list files older than a particular date.

2 /usr/admin/menu/filemgmt/filesize - SA to list largest files in a particular directory.

2 /usr/admin/menu/filemgmt/restore - SA to restore from "backup" & "store" media to built-in disks

2 /usr/admin/menu/filemgmt/store - SA to store files and directories of files onto removable media.

3 /usr/admin/menu/machinemgmt - SA machine management directory.

2 /usr/admin/menu/machinemgmt/autold - SA to set automatic boot device, default manual boot program.

2 /usr/admin/menu/machinemgmt/firmware - SA to stop all programs and enter firmware mode.

2 /usr/admin/menu/machinemgmt/floppykey - SA to create a "floppy key" removable disk.

2 /usr/admin/menu/machinemgmt/powerdown - SA menu call to /usr/admin/powerdown.

2 /usr/admin/menu/machinemgmt/reboot - SA to stop all running programs and reboot the machine.

2 /usr/admin/menu/machinemgmt/whoson - SA to print a list of users currently logged on to the machine.

3 /usr/admin/menu/softwaremgmt - SA software management directory.

2 /usr/admin/menu/softwaremgmt/installpkg - SA to install new software package onto the built-in disk.

2 /usr/admin/menu/softwaremgmt/listpkg - SA to list packages already installed.

2 /usr/admin/menu/softwaremgmt/removepkg - SA to remove previously installed package
          from the built-in disk.

2 /usr/admin/menu/softwaremgmt/runpkg - SA to run software package without installing it.

3 /usr/admin/menu/syssetup - SA system setup directory.

2 /usr/admin/menu/syssetup/admpasswd - SA to assign or change administrative passwords.

2 /usr/admin/menu/syssetup/datetime - SA to set the date and time.

2 /usr/admin/menu/syssetup/nodename - SA to set the machine node name.

2 /usr/admin/menu/syssetup/syspaswd - SA to assign system passwords.

3 /usr/admin/menu/tapemgmt - SA tape management directory.

2 /usr/admin/menu/tapemgmt/compress - SA to reorganize filesystem to remove fragmentation.

2 /usr/admin/menu/tapemgmt/format - SA to format removable cartridge tapes.

2 /usr/admin/menu/tapemgmt/info - SA to display information about cartridge tape.

2 /usr/admin/menu/tapemgmt/resetusage - SA to reset usage count on cartridge tape.

3 /usr/admin/menu/ttymgmt - SA tty management directory.

2 /usr/admin/menu/ttymgmt/lineset - SA to show tty settings and hunt sequences.

2 /usr/admin/menu/ttymgmt/mklineset - SA to create new tty line settings and hunt sequences.

2 /usr/admin/menu/ttymgmt/modtty - SA to show and optionally modify characteristics of tty lines.

3 /usr/admin/menu/usermgmt - SA user management directory.

2 /usr/admin/menu/usermgmt/addgroup - SA to add a group to the system.

2 /usr/admin/menu/usermgmt/adduser - SA to add users to the system.

2 /usr/admin/menu/usermgmt/delgroup - SA to delete a group from the system.

2 /usr/admin/menu/usermgmt/deluser - SA to delete users from the system.

2 /usr/admin/menu/usermgmt/lsgroup - SA to list groups in the system.

2 /usr/admin/menu/usermgmt/lsuser - SA to list users in the system.

2 /usr/admin/menu/usermg:nt/modadduser - SA to modify defaults used by adduser.

3 /usr/admin/menu/usermgmt/modgroup - SA modify group directory.

2 /usr/admin/menu/usermgmt/modgroup/chgname - SA to change the name of a group on the system.

3 /usr/admin/menu/usermgmt/moduser - SA modify user directory.

2 /usr/admin/menu/usermgmt/moduser/chgloginid - SA to change a user's login ID.

2 /usr/admin/menu/usermgmt/moduser/chgpasswd - SA to change a user's password.

2 /usr/admin/menu/usermgmt/moduser/chgshell - SA to change a user's login shell.

2 /usr/admin/mountfsys - SA to mount a removable medium file system.

2 /usr/admin/powerdown - SA to power down the system.

2 /usr/admin/setup - set the machine up the very first time it is used.

2 /usr/admin/umountfsys - SA to unmount a removable medium file system.

2 /usr/bin - the /usr/bin directory.

1 /usr/bin/addgrp - add members to a group.

1 /usr/bin/addpriv - add members to a privilege.

2 /usr/bin/adduser - add users to the system.

1 /usr/bin/at, /usr/bin/batch - execute commands at a later time.

2 /usr/bin/checkfsys - simplified administration script to check filesystem.

1 /usr/bin/clearances - print user clearances.

1 /usr/bin/crontab - user crontab file.

1 /usr/bin/delgrp - delete members from a group.

1 /usr/bin/delpriv - delete members from a privilege.

2 /usr/bin/deluser - delete users from the system.

1 /usr/bin/dominates - determine dominate relationship between labels.

1 /usr/bin/enable, /usr/bin/disable - enable, disable LP printers.

2 /usr/bin/id - print user, group ID and names.

1 /usr/bin/lp, /usr/bin/cancel - send/cancel requests to an LP printer.

1 /usr/bin/lpstat - print LP status information.

1 /usr/bin/lsgrp - list group information.

1 /usr/bin/lspriv - list privilege information.

1 /usr/bin/mailcheck - Check for and/or forward multilevel mail.

1 /usr/bin/mailx - interactive message processing system.

2 /usr/bin/makefsys - simplified administration script to make a filesystem.

2 /usr/bin/maxclear - establish user clearances.

1 /usr/bin/mkgrp - make new groups.

1 /usr/bin/mkpriv - make new privileges.

1 /usr/bin/mkuser - add a user entry to the password file.

2 /usr/bin/mountfsys - simplified administration script to mount a filesystem.

1 /usr/bin/mvpriv - move a privileged file.

2 /usr/bin/pack, /usr/bin/pcat, /usr/bin/unpack - compress and expand files.

2 /usr/bin/pg - file perusal filter for CRTs.

2 /usr/bin/powerdown - simplified administration script to power down the system.

1 /usr/bin/rmgrp - flag groups for removal from the system.

1 /usr/bin/rmpriv - flag privileges for removal from the system.

1 /usr/bin/rmuser - remove a user entry from the password file.

2 /usr/bin/sadp - disk access profiler.

2 /usr/bin/sag - system activity graph.

2 /usr/bin/sar - system activity reporter.

2 /usr/bin/sdiff - side-by-side difference program.

2 /usr/bin/setup - initialize the system for first user.

1 /usr/bin/shl - shell layer manager.

2 /usr/bin/sysadm - menu interface to do system administration.

2 /usr/bin/tabs - set tabs on a terminal.

2 /usr/bin/tic - terminfo compiler.

2 /usr/bin/timex - time a command; report process data and system activity.

2 /usr/bin/tput - query terminfo database.

2 /usr/bin/tr - translate characters.

2 /usr/bin/umountfsys - simplified administration script to unmount a filesystem.

3 /usr/lbin - the /usr/lbin directory.

2 /usr/lbin/admerr - Issue a standard error message for an administrative command.

2 /usr/lbin/agefile - age files by moving to older and older names.

2 /usr/lbin/askx - prompt with a question.

2 /usr/lbin/checklist - get an answer that is one from a list.

2 /usr/lbin/checkre - check an answer against a series of regular expressions.

2 /usr/lbin/checkyn - get a yes/no response from a user or check answer to question.

2 /usr/lbin/chkyn - get a yes/no response from a user or check answer to question.

2 /usr/lbin/devinfo - return information about a storage device.

2 /usr/lbin/disklabel - print the label of a diskette.

2 /usr/lbin/diskumount - perform a umount and complain if it does not work.

2 /usr/lbin/drivename - derive a device name from its pathname.

2 /usr/lbin/fdate - print file modifcation, creation, or access date/time.

2 /usr/lbin/filecheck - check for files added and deleted below the given directory.

2 /usr/lbin/fsinfo - print filesystem information.

2 /usr/lbin/getedt - get external device table.

2 /usr/lbin/ignore - no op command.

2 /usr/lbin/labelfsname - return file system name and volume label as shell variable assignments.

2 /usr/lbin/largest - find largest files under a given directory.

2 /usr/lbin/lightenfs - routine to clean up filesystems.

2 /usr/lbin/mklost+found - make a lost and found directory.

2 /usr/lbin/mkmenus - descend a tree directory looking for menus of commands.

2 /usr/lbin/mktable - concatenate files, stripping comments and empty lines.

2 /usr/lbin/ncpio - modified cpio.

2 /usr/lbin/num - check for all numeric arguments.

2 /usr/lbin/oldfile - look for files older than a specified number of days.

2 /usr/lbin/optparttn - allocate any disk free space into user partitions.

2 /usr/lbin/restorefiles - restore a file from the save area to the regular file systems.

2 /usr/lbin/rmjunk - remove files of dubious worth.

2 /usr/lbin/rmkdir   recursively make directories.

2 /usr/lbin/rrmdir - recursively remove directories.

2 /usr/lbin/samedev - determine if device files refer to the same actual device.

2 /usr/lbin/selectdevice - select SA name for a device.

2 /usr/lbin/selpattern - select which pattern matches a given device file.

2 /usr/lbin/spacewatch - look at file system space.

2 /usr/lbin/spclsize - determine the size of a special file.

2 /usr/lbin/vmkfs - make a file system within a hard disk partition.

2 /usr/lib - the /usr/lib/directory

1 /usr/lib/accept - allow LP to accept requests.

3 /usr/lib/cron - cron directory.

2 /usr/lib/cron/at.allow - file of users authorized to use the at command.

2 /usr/lib/cron/cron.allow - file of users authorized to use the cron facility.

2 /usr/lib/cron/log - log of cron activities.

1 /usr/lib/lpadmin - LP administrative interface.

1 /usr/lib/lpmove - move LP request to a specified printer.

1 /usr/lib/lpsched - LP scheduler.

1 /usr/lib/lpshut - stop LP scheduler.

3 /usr/lib/mailx - mailx directory

2 /usr/lib/mailx/mailx.help - mailx help file.

1 /usr/lib/mailx/rmmail - remove empty mail file from mail directory.

1 /usr/lib/mv_dir - move a directory within its parent (i.e., rename).

2 /usr/lib/pgmark - add sensitivity marks to line printer output.

1 /usr/lib/reject - prevent LP from accepting requests.

1 /usr/lib/sadc - samples system data and writes in binary format. Used with "sar".

3 /usr/mail - mail directory

3 /usr/spool - spool directory.

3 /usr/spool/cron - cron spool directory.

3 /usr/spool/cron/atjobs - atjobs directory.

3 /usr/spool/cron/crontabs - cron requests.

2 /usr/spool/cron/crontabs/adm - adm cron jobs.

2 /usr/spool/cron/crontabs/root - root cron jobs.

2 /usr/spool/cron/crontabs/sys - sys cron jobs.

2 /usr/spool/cron/crontabs/sysadm - sysadm cron jobs.

2 /usr/spool/lp/interface/5310 - writes a header and trailer page to the lp printer for each job.

3 /usr/tmp - usr temporary directory.

This page intentionally left blank.

BIBLIOGRAPHY

Department of Defense, *Trusted Computing System Evaluation Criteria,* publication number CSC-STD-001-85

System V/MLS Trusted Facility Manual, AT&T Technologies Federal Systems, August 1989.

The 630/MLS Trusted Facility Manual; Multi-Level Secure Window Management, AT&T Technologies Federal Systems, August 1989.

The 630/MLS User's Guide; Multi-Level Secure Window Management, AT&T Technologies Federal Systems, August 1989.

System V/MLS User's Guide, AT&T Technologies Federal Systems, August 1989.

System V/MLS 1.1 UNIX Multi-Level Security Testing Documentation, AT&T Technologies Federal Systems, April 1989.

The System V/MLS Family of Products; Security Enhanced UNIX Systems, AT&T Technologies Federal Systems, Sept 1988.

High-Level Requirements for System V/MLS AT&T Federal Systems' B1 Certifiable UNIX System V-Based Product, AT&T Technologies Federal Systems, Sept 1988.

System V/MLS; Requirements and Documentation Cross Reference, AT&T Technologies Federal Systems, Sept 1988.

AT&T System V/MLS Porting Strategy and Kernel Interface Specification, AT&T Technologies Federal Systems, Sept 1988.

The System V/MLS Security Audit Trail - ˜ ˄quirements, Design, and Implementation, AT&T Technologies Federal Systems, March 1988.

System V/MLS Multilevel Security Module (MLS); Requirements, Design and Implementation, AT&T Technologies Federal Systems, Sept 1988.

Design of System V/MLS Additional Security Features, AT&T Technologies Federal Systems, March 1988.

System V/MLS; Installation Guide, AT&T Technologies Federal Systems, Aug 1988.

The sys3b System Call; System V/MLS Security Analysis, AT&T Technologies Federal Systems, Aug 1988.

System V/MLS Release Notes (for release 1.1), AT&T Technologies Federal Systems, Oct 1988.

Rating Maintenance Plan for System V/MLS, AT&T Technologies Federal Systems, August 1989.

WE 32101 Memory Management Unit Information Manual, AT&T Information Systems, 1986.

WE 32100 Microprocessor Information Manual, AT&T Information Systems, 1986.

Bach, Maurice J., *The Design of the UNIX Operating System*, Prentice-Hall Inc., Englewoood Cliffs, NJ, 1986.

Bell, D. E. and LaPadula, L. J., *Secure Computer Systems: Unified Exposition and Multics Interpretation*, MITRE Corp., Bedford, MA, MTR-2997, July 1976.

Brown, Patrick G., *DSB 440330; AT&T 3B2/500 Computer Test Plan & Specification*, Issue 1, AT&T Information Systems, August 1987.

Coss, Michael J. *A Study of Object Reuse on System V/MLS*, AT&T Technologies Federal Systems, July 1988.

Coss, Michael J. *The Bourne Shell; Design and Security Analysis*, AT&T Technologies Federal Systems, Oct 1988.

Flink, C. W., *Multi-Level Secured Directories- An Experimental System V/MLS Feature*, AT&T Technologies, Federal Systems, November 1987.

Flink, C. W. and Powers, M. C., *System V/MLS Multilevel Security Module (MLS) - Requirements, Design, and Implementation*, AT&T Technologies, Federal Systems, April 1988.

Grenier, J. R., *WPROC Internals*, AT&T Technologies, Federal Systems, March 1986.

Juhn, H., *Programming Environment ID in the 630 MTG Terminal Software Development System*, AT&T Technologies, Federal Systems, March 1988.

Nguyen, H. T, *630 MTG Memory Management System*, AT&T Technologies, Federal Systems, March 1986.

Nguyen, H. T, *Cache System in the 630 MTG Terminal*, AT&T Technologies, Federal Systems, March 1986.

Nguyen, H. T, *Default Initialization of Variables in Cached Applications*, AT&T Technologies, Federal Systems, April 1987.

Schrofer, E. P., *DSB 440200; AT&T FALCON Computer System Requirements*, Issue 2, AT&T Information Systems, March 1987.

Schumann, Al, *Memory Management Subsystem; Design Documentation*, Review Issue 1, AT&T Information Systems, November 1987.

Sharp, R. S. and Weiss, J. D. *Use of the AT&T 630 Terminal with System V/MLS*, AT&T Technologies, Federal Systems, December 1987.

Smith, M. E. and Smith-Thomas, B. *Securing the 630 MTG Intelligent Terminal - Requirements, Design and Implementation*, AT&T Technologies Federal Systems, April 1989.

Weiss, J.D. *Design of System V/MLS Additional Security Features*, AT&T Technologies, Federal Systems, April 1988.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT **Approved for public release; Distribution Unlimited**
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	

4. PERFORMING ORGANIZATION REPORT NUMBER(S) CSC-EPL-89/003	5. MONITORING ORGANIZATION REPORT NUMBER(S) ~~S33,122~~ S233121

6a. NAME OF PERFORMING ORGANIZATION National Computer Security Center	6b. OFFICE SYMBOL (If applicable) C12	7a. NAME OF MONITORING ORGANIZATION

6c. ADDRESS (City, State and ZIP Code) 9800 Savage Road Ft. George G. Meade, MD 20755-6000	7b. ADDRESS (City, State and ZIP Code)

8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS.

	PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.

11. TITLE (Include Security Classification)
FINAL EVALUATION REPORT ATT SYSTEM V/MLS RELEASE 1.1.2

12. PERSONAL AUTHOR(S)
Bielat, Karen M., Crescenzi, Caralyn A., Gabriele, Mark D.; Havener, William N., Kass, Sharon G., [MITRE Corporation]; Traxler, Holly M. [IDA]

13a. TYPE OF REPORT Final	13b. TIME COVERED FROM ___ TO ___	14. DATE OF REPORT (Yr, Mo., Day) 891018	15. PAGE COUNT 174

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) NCSC TCSEC
FIELD	GROUP	SUB. GR.	

19. ABSTRACT (Continue on reverse side if necessary and identify by block number)
The security protection provided by the AT&T System V/MLS (System V/MLS) operating system, configured as described in the Trusted Facility Manual, when running on the AT&T 3B2/500 or 3B2/600 mini computers, has been evaluated by the National Computer Security Center (NCSC). The security features of System V/MLS were examined against the requirements specified by the DOD Trusted Computer System Evaluation Criteria (the Criteria or TCSEC) dated 26 December 1985 in order to establish a TCSEC rating. The NCSC evaluation team has determined that the highest class at which System V/MLS satisfies all the specified requirements of the Criteria is class B1. A system that has been rated as being a B1 class system provides a Trusted Computing Base (TCB) that preserves the integrity of sensitivity labels and uses them to enforce a set of mandatory access control rules.

This report documents the findings of the formal evaluation of AT&T System V/MLS operating system.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL DENNIS E. SIRBAUGH	22b. TELEPHONE NUMBER (Include Area Code) (301)859-4458	8b. OFFICE SYMBOL C12