NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

INTEGRATING A TRUSTED COMPUTING BASE EXTENSION SERVER AND SECURE SESSION SERVER INTO THE LINUX OPERATING SYSTEM

by

Mark V. Glover

September 2001

Thesis Advisor:	Cynthia E. Irvine
Associate Advisor:	David Shifflett

Report Documentation Page

Report Date	Report Type	Dates Covered (from... to)
30 Sep 2001	N/A	-

Title and Subtitle Integrating a Trusted Computing Base Extension Server and Secure Session Server into the Linux Operating System.	**Contract Number**
	Grant Number
	Program Element Number
Author(s) Glover, Mark V.	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) Research Office Naval Postgraduate School Monterey, Ca 93943-5138	**Performing Organization Report Number**
Sponsoring/Monitoring Agency Name(s) and Address(es)	**Sponsor/Monitor's Acronym(s)**
	Sponsor/Monitor's Report Number(s)
Distribution/Availability Statement Approved for public release, distribution unlimited	
Supplementary Notes	
Abstract	
Subject Terms	
Report Classification unclassified	**Classification of this page** unclassified
Classification of Abstract unclassified	**Limitation of Abstract** UU
Number of Pages 84	

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 2001	Master's Thesis

4. TITLE AND SUBTITLE
Integrating a Trusted Computing Base Extension Server and Secure Session Server into the Linux Operating System.

5. FUNDING NUMBERS

6. AUTHOR(S) Glover, Mark V.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT

The Multilevel Secure Local Area Network (MLS LAN) Project at the Naval Postgraduate School's, Center for Information Security (INFOSEC) Studies and Research (NPS CISR) is building a trusted network system that is both necessary and sufficient to provide a multilevel networking solution for real world use. The current configuration provides the necessary trusted network services on the TCSEC Class B-3 evaluated XTS-300, which is a combination of the STOP version 4.4.2 multilevel secure operating system, and a Wang-supplied Intel x86 hardware base. The interface for the STOP operating is based on the System V.3 UNIX implementation.

System V.3 lacks many of features available in more modern UNIX implementations such as System V.4 and BSD 4.3, and also lacks many of the features in POSIX and ANSI C standards. Finally, the CPU is several generations older than the more current Intel processors. This thesis discusses the port of several MLS trusted network services on the XTS-300 to a Linux operating system running on an Intel Pentium Processor. The new Linux TCB Server configuration will permit further experimentation with MLS architectural issues in a more modern, flexible and easily modifiable environment.

The port was accomplished by identifying and modifying the necessary software modules needed, to adapt to a Linux environment.

This thesis proves that XTS-300 TCB services can be ported to Linux system without any negative effects on performance thus allowing a move toward a more security enhanced implementation.

14. SUBJECT TERMS
Multilevel Secure Local Area Network (MLS LAN) Project, Trusted Computing Base, Trusted Computing Base Extension, Linux, XTS-300, IPSEC, Trusted Path, API

15. NUMBER OF PAGES
65

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited

Integrating a Trusted Computing Base Extension Server and Secure Session Server into the Linux Operating System

Mark V. Glover
Lieutenant Commander, United States Navy

B.S., Norwich University, 1990
M.S. Information Technology Management, Naval Postgraduate School 1998

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2001

Author: Mark V. Glover

Mark V. Glover

Approved by: Cynthia E. Irvine

Cynthia E. Irvine, Thesis Advisor

David Shifflett, Associate Advisor

Chris Eagle, Chairman
Information Systems Academic Group

ABSTRACT

The Multilevel Secure Local Area Network (MLS LAN) Project at the Naval Postgraduate School's, Center for Information Security (INFOSEC) Studies and Research (NPS CISR) is building a trusted network system that is both necessary and sufficient to provide a multilevel networking solution for real world use. The current configuration provides the necessary trusted network services on the TCSEC Class B-3 evaluated XTS-300, which is a combination of the STOP version 4.4.2 multilevel secure operating system, and a Wang-supplied Intel x86 hardware base. The interface for the STOP operating is based on the System V.3 UNIX implementation.

System V.3 lacks many of features available in more modern UNIX implementations such as System V.4 and BSD 4.3, and also lacks many of the features in POSIX and ANSI C standards. Finally, the CPU is several generations older than the more current Intel processors. This thesis discusses the port of several MLS trusted network services on the XTS-300 to a Linux operating system running on an Intel Pentium Processor. The new Linux TCBE Server configuration will permit further experimentation with MLS architectural issues in a more modern, flexible and easily modifiable environment.

The port was accomplished by identifying and modifying the necessary software modules needed, to adapt to a Linux environment.

This thesis proves that XTS-300 TCB services can be ported to Linux system without any negative effects on performance thus allowing a move toward a more security enhanced implementation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First and foremost, I acknowledge my wife Anitra. Without your continued encouragement and patience, this time at NPS would not have been as successful.

I wish to express my sincere appreciation to Professor Irvine and Dave Shifflett. Their keen insight and enthusiasm for the subject matter made this a very worthwhile and enjoyable learning experience.

I. INTRODUCTION

The primary objective of this thesis is to port two Trusted Services – The Trusted Computing Base Extension (TCB) Server and the Secure Session Server, which are currently implemented for the XTS-300 in NPS's Multilevel Secure Local Area Network (MLS LAN) Project, to the Linux Operating System (kernel version 2.2.16, build 22) running on an Intel Pentium Processor.

A. BACKGROUND

1. Purpose

This thesis is a continuation of an ongoing Naval Postgraduate School effort to develop a high assurance Multilevel Secure Local Area Network (MLS LAN) that incorporates commercial-off-the-shelf client workstations to provide multiple users with simultaneous access to stored data at different sensitivity levels. As an additional requirement, it should be cost effective and easy to use in an office environment. This thesis will build upon this project by including in the network environment a modern, open source operating system, Linux.

Currently, in the MLS LAN, the TCB and Secure Session Servers are part of a TCB that includes the TCB of the previously evaluated Class B3 high assurance server, the Wang Government Services Incorporated XTS-300™ Operating System.

Currently, the NPS research team has modified the XTS-300 to support many MLS LAN TCB requirements, such as Secure Attention Key (SAK) recognition and processing, user access identification and authentication (I&A), session control and TCP/IP

configuration management. The port of these services, which are not all inclusive, to Linux will permit further experimentation with MLS architectural issues in a more flexible and more easily modifiable environment.

This thesis will not attempt to implement any Mandatory Access Control (MAC) based policies used in the current MLS LAN. This will be left for future work.

After completion of this thesis, the Linux-incorporated TCB Extension Server and Secure Session Server will show that the XTS-300 TCB services can be ported to Linux system as a single level system, ultimately moving towards a more security enhanced implementation.

2. Project Overview

One of the major components of the MLS LAN is the Trusted Computing Base (TCB). However, in order to extend a "Trusted Path" between TCB services and remote clients over the MLS LAN, a Trusted Computing Base Extension (TCBE), MLS LAN Connection Protocols, TCB Extension Server, and Secure Session Server are also needed. These components ensure that "communications via the trusted path shall be activated exclusively by a user of the TCB and shall be logically isolated and unmistakably distinguishable from other paths."[Ref.1] This thesis will focus primarily on the TCB Extension Server and the Secure Session Server. The TCB Extension Server's function is to extend the TCB perimeter securely over a LAN to the requesting TCBE-equipped workstation, whereas the Secure Session Server is responsible for accepting connections from TCBE-equipped client workstations and establishing the TCP/IP protocol service for the user. In the target architecture, the TCB and Secure Session Servers will be imported into the Linux interface. Figure 1 illustrates the target architecture.

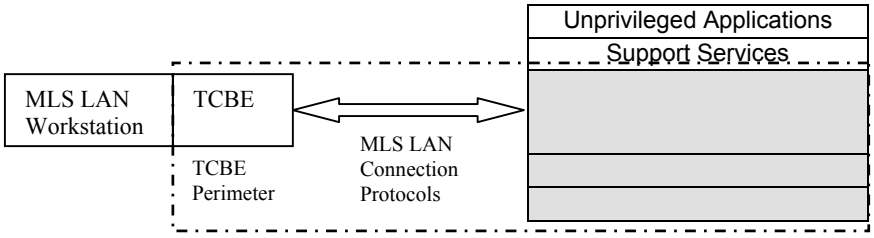

Figure 1-1: Proposed System Architecture

3. MLS LAN Project Goals

The MLS LAN is an effort to provide governmental and commercial organizations with a cost effective, multilevel networking solution by leveraging existing high assurance technology. The ultimate goal of the project is to demonstrate a prototype network design that offers the ability to provide concurrent high assurance access for network users to data at multiple sensitivity levels through the incorporation of inexpensive commercial personal computers and software. The intended design of the network is to integrate the security features of a previously evaluated Class B3 high assurance server, the Wang Government Services Incorporated XTS-300, with the conveniences of up-to-date operating systems and the latest commercial office automation software. The current plan for the MLS LAN network is to provide this functionality using the universally accepted TCP/IP protocol suite to allow our multilevel networking functionality to be layered on top of any chosen technology used in the lower layers of the OSI model. When completed, the MLS LAN will provide a cost effective multilevel solution within an easy-to-use office environment.

4. Thesis Goals

This thesis is focused on researching the Linux Operating System, (Kernel version 2.2.16 build 22), modifying Application Program Interfaces (APIs) and relevant functions

with the purpose of incorporating the MLS TCB Extension Server and Secure Session Server into the Linux Operating environment. It is not the intent of this thesis to overhaul the Linux operating system, nor will the solution be designed to satisfy any of the International Standards Organization's (ISO) Common Criteria ratings relating to Multilevel Secure Systems.

A major benefit of porting the Trusted Computing Base Extension Server and Secure Session Server, used in the MLS LAN architecture, to a Linux Kernel is to take advantage of the functionalities incorporated in a more modern operating system, including shared libraries, shared copy-on-write executables, demand loading, TCP/IP networking, true multitasking, and more advanced memory management.

5. Advantages of Linux vs. the XTS-300

There numerous UNIX versions, however Linux has been chosen for this port based on capabilities, legal issues, useability, economical issues and ease of portability benefits.

a. Capabilites

The STOP 4.4.2 operating system could possibly be termed a "traditional" UNIX system, since it is based upon the UNIX System V.3 implemetaion and lack many of the System V.4 features. Generally, "traditional" UNIX systems are designed to run on a single processor and lack the ability to protect their data structures from concurrent access by multiple processors. Moreover, the "traditional" UNIX kernel is not very versatile, supporting only a single type of file system, process scheduling policy, and executable format. The "tradtional" UNIX kernel is not designed to be extensible and has few facilities for code reuse. The result is that, as new features have been added to the various

"traditional" UNIX versions, a lot of new code had to be added, yeilding a bloated and unmodular kernel [Ref 3].

Another disadvantage is that the XTS-300 is a hardware dependent platform. It's source code is written to run on an i386 based processor.

Linux tries to maintain a distinction between hardware-dependent and hardware independent source code. Both the */arch* and */include* directories include nine subdirectories corresponding to the nine hardware platforms supported. The standard platforms are outlined in table 1-2.

Chip Architecture	**Platform**
arm	Acorn personal computers (Now part of Broadcom Inc.)
alpha	Compaq Alpha Workstations
i386	IBM compatible PC's based upon Intel 80x86 or Intel 80x86 compatible processors
m68k	PC's based upon Motorola MC680x0 microprocessors
mips	Workstations based on Silicon Graphics MIPS processors
ppc	Workstations based on Motorola-IBM PowerPC microprocessors
sparc	Workstations based on Sun Microsystems SPARC microprocessors
sparc64	Workstations based on Sun Microsystems 64-bit Ultra SPARC microprocessors
s390	IBM System /390 mainframes

Table 1-1: Hardware Architectures supported by Linux

b. Linux is Readily Available

Linux has been copyrighted under the terms of the GNU General Public

License (GPL)[1]. This is a license written by the Free Software Foundation (FSF) that is designed to prevent people from restricting the distribution of software. In summary, a user can charge as much as he or she likes for some particular software, but can't prevent the person it was sold to from giving it away for free. It also means that the source code is open source and must also be available. This is useful for programmers and developers. The license also says that anyone who modifies the program must also make his or her version freely redistributable.

c. Usability

Another adavantage Linux has is its wide support for graphical user interfaces, including The X Window System. This program, from MIT, allows computers to create graphical windows, and is used on many different UNIX platforms. Although the XTS-300 does provide support for X Windows, it does not readily support many of the freely available desktops such as GNOME and KDE.

d. Portability

Linux is highly compatible with many common operating systems. For example, you can mount file systems from many other operating systems such as MS-DOS MS Windows, SVR4, OS/2, Mac OS, Solaris, SunOS, NeXTSTEP, and many BSD variants, and so on. You can operate with many network architectures like ethernet, Fiber Distributed Data Interface (FDDI), etc., and by using suitable libraries, Linux is able to run programs written for other operating systems. [Ref. 8]

[1] The GNU Project is coordinated by the Free Software Foundation, Inc (http://www.gnu.org); its aim is to implement a whole operating system freely useable by everyone. The availability of the GNU C compiler has been essential for the success of the Linux project [Ref. 8].

Linux has been ported to a number of platforms. It runs on Compaq Alpha AXP, Sun SPARC and UltraSPARC, Motorola 68000, PowerPC, PowerPC64, ARM, Hitachi SuperH, IBM S/390, MIPS, HP PA-RISC, Intel IA-64 and DEC VAX. Ports are currently in progress to the AMD x86-64 architecture.

e. Economic Benefits

Linux is free. You can install a complete UNIX system at no expense other than that for hardware. Therefore, porting to the Linux operating system allows us to go from a more limited environment to a more flexible one, with little or no cost.

6. Disadvantages of Linux vs. XTS-300

The only true disadvantage relevant to this thesis is that Linux only incorporates Discretionary Access Controls (DAC) and does not provide Mandatory Access Controls (MAC). MAC and DAC will be explained in Chapter II.

7. Methodology

- Linux Red Hat 7.0 (Kernel version 2.2.16 Build 22) and Wang XTS-300 interfaces will be analyzed to identify the existing design, primary data structures, libraries, Application Program Interfaces, functional dependencies (internal and external), etc.
- Use a UNIX based development environment to develop and/or modify code for porting to the Linux environment.
- Use a run/test environment to debug and test new code.

Interact with the designers and users of the MLS LAN project in order to keep the implementation within the scope of the system's original design while enhancing its appeal and potential use in a more modern, less expensive environment.

B. CHAPTER OVERVIEW

1. Introduction

This thesis is composed of six chapters. This chapter provides the motivation, objectives, research questions, scope and methodology employed to conduct the research. Chapter II provides descriptions and comparisons of the XTS-300 and Linux security and software architectures. Chapter III discusses XTS-300 and Red Hat Linux porting environments. Chapter IV describes the ported modules and databases. Chapter V describes the major source code changes in the port. Chapter VI presents Conclusions and Recommendations. Appendix A provides a glossary of terms. Each of these Chapters is outlined below.

2. The XTS-300 and Linux System Architectures

a. XTS-300

The XTS-300 product is a combination of STOP 4.4.2, a multilevel secure operating system, and a Wang-supplied x86 hardware base. Its system architecture is designed to implement the concept of a Trusted Computing Base. The primary software components are the Security Kernel, TCB System Services (TSS), Trusted Software, and the Commodity Application System Software (CASS). The Security Kernel provides basic

system operating services and enforces system security. The TSS software provides general trusted services to the XTS-300 application and system software. Trusted Software provides additional security services outside the Security Kernel. Finally, CASS provides an execution environment on the XTS-300 for UNIX-based applications. The TCB Extension Server and Secure Session Server run as trusted software.

b. Linux

The Linux operating system was initially developed in 1991 by Linus Torvalds for IBM-compatible 32-bit x86-based PCs (386 or higher). It is a clone of the UNIX operating system and was written entirely from scratch,. It has all the features of a modern (System V Release 4 and 4.4BSD) fully-fledged UNIX, including true multitasking, virtual memory, shared libraries, demand loading, shared copy-on-write executables, better memory management, and TCP/IP networking.

Unlike the XTS-300, where the CPU runs in the four execution state (Rings), the Linux kernel only makes uses of Kernel mode (Ring 0) and User Mode (Ring 3). The TCB Extension Server and Secure Session Server will be ported to run in the User Mode environment.

3. XTS-300 and Linux Porting Environments

The XTS-300's STOP, version 4.4.2, operating system is designed to support much of the UNIX System V.3 interface for applications software. Linux implements many features found in the System V.4 and 4.4BSD strains of UNIX, but does not necessarily adhere to them in all cases. It is mostly POSIX.1 compliant (IEEE Std 1003.1-1988).

There a many specific API services, library and system calls, used in XTS-300's System V.3 interface that may be incorporated differently or in some cases more effectively in a POSIX.1 interface. This chapter will delineate portability issues between System V.3 interface and the POSIX.1 interface, specific to an XTS-300-to-Linux port.

4. Ported Mechanisms

This chapter describes the TCB Extension and SSS servers and how their execution domains compare between the XTS-300 and Linux environments.

5. Source Code Transforms to the Linux Environment

This chapter details the changes to source code, and the significant compile and link errors that occurred during the XTS-300 to Linux port.

6. Results and Recommendations

This chapter outlines results such as performance, dependency issues, security and future recommendations.

C. APPENDIX OVERVIEW

1. Appendix A: Glossary of Terms

Appendix A provides a glossary of common terms.

II. THE XTS-300 AND LINUX SECURITY AND MAJOR COMPNENTS

A. BACKGROUND

Developed by Bell Laboratories in 1969, UNIX has evolved from a single operating system to a family of operating systems, examples of which include AIX, BSDI, FreeBSD, HP-UX, IRIX, Linux, NetBSD, OpenBSD, Pyramid, SCO (UnixWare and OpenServer), Solaris, SunOS, and Tru64 UNIX. Other mainstream operating systems, such as Windows NT, have UNIX roots. Additionally, the XTS-300 has a UNIX-like interface.

This chapter will briefly discuss the security policies enforced by the XTS-300 and Linux. Additionally, the conceptual software and hardware components will also be discussed.

B. DISCRETIONARY ACCESS CONTROL POLICY

Discretionary Access Control (DAC) is a means of restricting access to objects based on the identity of subjects and/or groups to which they belong [Ref. 15]. Typical UNIX file system objects are discretionary, because the user can change access modes of the object at his or her discretion. In UNIX and many other systems, DAC is always tied directly to User Identification Numbers (UID's). The simplest implementation of this is that of permission bits which support owner/group/world permission bits or read/write/execute.

The most common object that a typical user deals with is a file. File permissions are defined by the file mode. The file mode contains nine bits that determine access permissions to the file, plus three special bits. Access permissions are defined for three classes of users: the owner, the group, and the rest of the world.

In Linux, a concept that is typically applied is that of a root (UID 0) or super-user account. This account is trusted and can bypass system security mechanisms. It has all DAC permissions, thus overriding access controls on all Linux objects, and often, other implemented security settings.

The POSIX.1 DAC standard specifies the use of a permission bit to signify a particular mode of access. This standard allows the defined permissions of read, write and execute to be specified for:

1. the file owner,
2. the group of users specified as the ``owning group,'' and
3. all other users (named ``other'').

This mechanism can be cumbersome to use if permissions need to be specified for a named user who is not the owner (and it is nearly impossible to specify separate permissions for two users, neither of whom is the owner). It is also not possible to provide specific permissions for different named groups of users.

Another concern is that DAC does not provide any way of restricting information flow once a valid subject has “legally” accessed an object. This weakness makes DAC policies vulnerable to Trojan Horses maliciously leaking information. Therefore the need arises for providing additional controls -- such as Mandatory Access Controls, that limit the indiscriminate flow of information in the system.

C. MANDATORY ACCESS CONTROL POLICY

Systems enforce Mandatory Access Control (MAC) by restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the

objects and the formal authorization (i.e. clearance) of subjects to access information of such sensitivity. [Ref. 15]. In mandatory access control schemes, the system protects the files and resources. This type of access can be implemented to handle a number of different security levels, such as Top Secret, Secret, Confidential, and Unclassified, Official Use Only, etc.

The system applies an access control sensitivity label to every user, file, and resource in the system. By comparing the labels the system (not the system manager or user) determines which user can access what information in the system.

A label consists of two parts:

• A classification — a single hierarchical level to permit access, for example, Top Secret, Secret, Confidential, Unclassified, etc.

• A set of categories — a nonhierarchical set representing distinct areas of information, such as Combat Systems, Supply, Engineering, Operations, or Squadron, etc.

D. TRUSTED PATH

A trusted path is a mechanism by which a user may directly interact with trusted software, which can only be activated by either the user or the trusted software and may not be imitated by other software. In the absence of a trusted path mechanism, malicious software may impersonate trusted software to the user or may impersonate the user to trusted software. Such malicious software could potentially obtain sensitive information, perform functions on behalf of the user in violation of the user's intent, or trick the user into believing that a function has been invoked without actually invoking it. In addition to supporting trusted software in the base system, the trusted path mechanism should be

extensible to support the subsequent addition of trusted applications by a system security policy administrator.

The concept of a trusted path can be generalized to include interactions beyond just those between trusted software and users. The MLS LAN uses the concept of a trusted channel for communication between trusted software on different network components such as a TCBE client and the TCB Extension Server. More generally, its mechanism guarantees a mutually authenticated channel, or *protected path*, to ensure that critical system functions are not being spoofed.

E. XTS-300 COMPONENTS

1. Introduction

The major components of the XTS-300 are the STOP 4.4.2 multilevel secure operating system, and a Wang-supplied x86 hardware base. Its system architecture is designed to support both a reference validation mechanism, or security kernel, and a Trusted Computing Base.

This section will provide a brief description of the software and hardware components of the XTS-300. If the reader wishes to obtain a detailed description of the software architecture and security model, refer to [Ref. 1]. Additionally, the security policies implemented in the XTS-300 will also be discussed.

2. Primary Software Components

The primary software components are the Security Kernel, TCB System Services (TSS), Trusted Software, and the Commodity Application System Software (CASS). Figure 2-1 shows the XTS-300 System Design [Ref. 11].

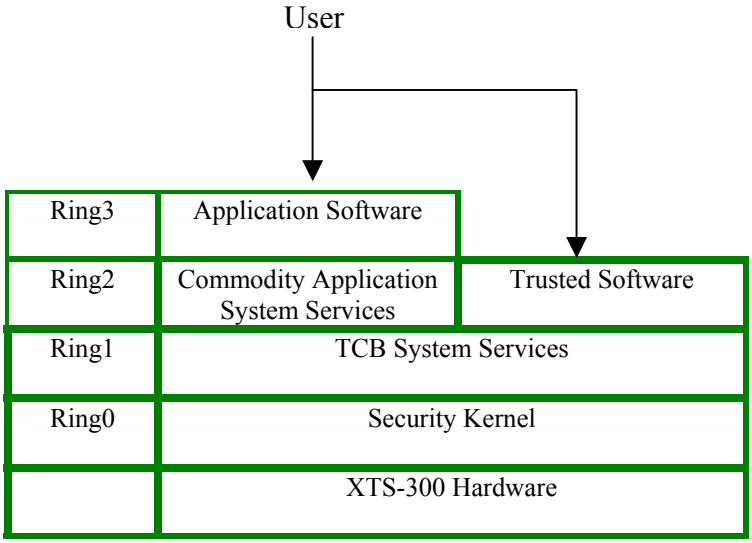

Figure 2-1: XTS-300 Hardware Abstraction

3. Hardware Abstraction

An Intel *x86* microprocessor has four hardware privilege levels, also known as rings, that are used to control such things as memory access and access to certain sensitive CPU instructions (such as those related to security). At any given moment, a process executes at one of these privilege levels.

As shown in Figure 2-1, the Security Kernel operates in the most privileged ring (Ring 0). It provides all Mandatory Access Control, subtype, and a portion of the Discretionary Access Control (DAC) policy enforcement for process and device objects, resource management, process handling, and interrupt handling.

The TSS software, which operates in the next-most-privileged ring (Ring 1) is controlled by the security kernel. It implements a hierarchical file system, supports user I/O, and implements the remaining discretionary access control for both trusted and untrusted processes. Throughout this thesis, the TSS software will also be referred to as Operating System Services (OSS) domain.

Trusted Software and Commodity Application System Services (CASS), executing in Ring 2, provides additional security services and user functions outside the Security Kernel. The ring is shared by the trusted software such as the STOP operating system or user developed trusted code and the untrusted CASS. CASS provides an execution environment for UNIX-based applications on the XTS-300. The CASS is not considered a part of the TCB. While executing in the CASS domain, a process may access information residing in a ring of the same or lesser privilege, but not in a ring of greater privilege [Ref. 11]. Trusted Software functions allow system operators and administrators to perform security-related housekeeping and other privileged tasks not supported by the STOP components [Ref. 1].

Ring 3, Application Domain, is reserved for user processes and is the least privileged.

4. Security Policy

a. XTS-300 MAC

The XTS-300 enforces the DOD policy for multilevel secure computing as

formalized in the obsolete Trusted Computer Security and Evaluation Criteria (TCSEC)[2] and provides MAC that supports both mandatory security and integrity policies. The multi-level features of the XTS-300 allow separation of users who are at different clearance levels, and prevents a lower level user from reading a higher level user's files or data. The use of categories allows separation of the data between users at the same level, giving access to certain users on a mandatory need-to-know basis [Ref. 12]. The TCB provides sixteen (16) hierarchical security classifications and sixty-four (64) mutually independent security compartments or categories. Eight (8) hierarchical integrity classifications and sixteen (16) mutually independent integrity compartments or categories are also provided by the TCB.

The mandatory security policy enforced by the XTS-300 is based on the Bell and LaPadula security model [Ref. 9]; the mandatory integrity policy is based on the Biba integrity model [Ref. 10].

b. XTS-300 DAC

The XTS-300 also implements discretionary access control (DAC), where access to an object is determined by the identity of the user and/or groups to which a user belongs.

Specifically, the TCB enforces the following discretionary access rule:

- Access modes – a user is only allowed to access a data object in the mode(s) granted by the owner of the object. Each object has allowed permissions (read, write, execute) for the owner of the object, for the

[2] ISO/IEC 15408, the Common Criteria (CC) for Information Technology (IT) Security Evaluation is the new standard for specifying and evaluating the security features of computer products and systems. It has replaced the TCSEC [Ref. 19].

members of the owner's group, for other specifically identified users and groups, and for all others.

The XTS-300 also enforces a general configurable policy that strengthens the traditional mandatory and discretionary access rules called Subtype policy [Ref 12].

Finally, the system provides for user identification and authentication needed for user ID-based policy enforcement.

F. LINUX COMPONENTS

1. Introduction

The major components of the Linux port are comprised of Linux Red Hat 7.0 – kernel 2.2-16 build 22 and an Intel Pentium III hardware base. Unlike the XTS-300, the Linux system architecture is not designed to support any reference validation mechanism such as a security kernel, or a Trusted Computing Base. The Linux kernel mainly implements a DAC security policy.

This section will provide a brief description of the software and hardware components of the Linux port, and its implemented security policies.

2. Primary Software Components

The Linux system architecture is designed to support a single monolithic kernel that runs in kernel mode, while all user programs run in user mode. The kernel contains code for the file system, device drivers as well as code for process management. In contrast,

UNIX has always managed large parts of many system functions, such as networking etc, outside the kernel, in user mode processes.

Linux is a true UNIX kernel, although it is not a full UNIX operating system. This is because does not include all the applications such as file system utilities, windowing systems and GUI desktops, system administrator commands, text editors, compilers, etc. However, since many of these programs are freely available under the GNU General Public License, they can be installed into one of the file systems supported by Linux [Ref. 8].

3. Hardware Abstraction

As stated earlier, an Intel *x86* microprocessor has four hardware privilege levels. However, Unlike the XTS-300, which uses all four hardware privilege levels, Linux systems support a Process/Kernel Model that only uses two levels of privilege-- Ring 0 (kernel) is the most privileged level, with complete access to all memory and CPU instructions, and Ring 3 (user programs) is the least privileged level. Figure 2-2 provides a general hardware abstraction model of the Linux operating system.

When a process is running in Ring 3, it is said to be in User Mode and cannot directly access the kernel data structures or kernel programs. Therefore the process should only be able to make well-defined kernel calls through kernel system interface. However, when a process is running in Ring 0, it is said to be in kernel mode, and the aforementioned restrictions no longer apply. A process will switch from user mode (Ring 3) to kernel mode (Ring 0) when making certain API function calls that require a higher privilege level, such as those that involve accessing files or performing graphics-related functions. In fact, many user-level threads can spend more time in kernel mode than in user mode.

A system (kernel) call is implemented by a trap, which is a "software exception" that transfers control to the kernel (Ring 0); in Linux, for the *x86*, this is a software interrupt (also called a "gate") [Ref. 16]. When the Kernel Mode functions are completed, the kernel transfers control back to user mode. Transferring modes between the kernel and user space prevents a programmer from being able to write user mode instructions that can run privileged instructions in kernel mode, thus not providing any security or fault tolerance mechanisms. In order to implement these mechanisms, the hardware must permit the kernel to verify the input parameters e.g. valid buffer addresses, restore the saved states during the mode switches, return to user mode (Ring 3), and resume execution.

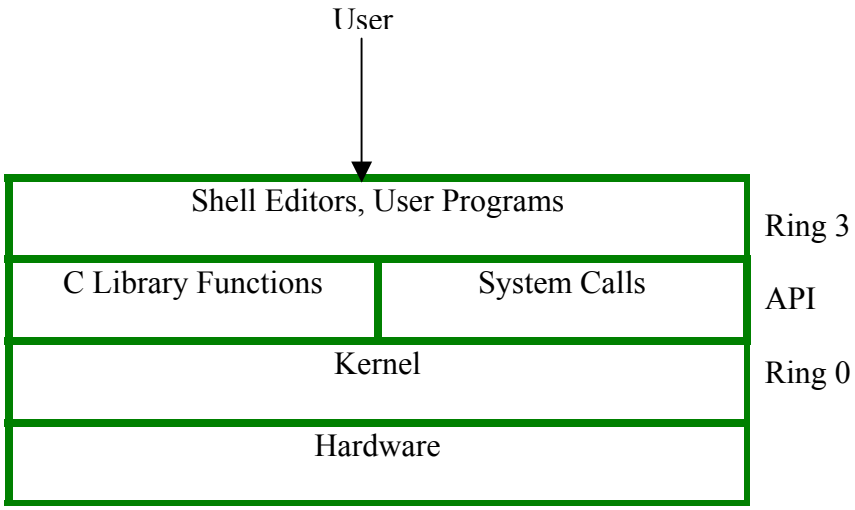

Figure 2-2: Linux Hardware Design

4. Security Policy

In the Linux kernel only a few mechanisms are provided for security. The most significant is DAC. In its Linux implementation, DAC is implemented in the form of file permissions known as Access Control lists (ACL). ACLs allow a list of user-specific rights

to be placed on objects, thus allowing users to make a determination as to the security that should be applied to their data. Typically DAC controls are placed on all Linux file system objects such as semaphores, shared memory, files, devices, etc.

Linux developers have slowly been implementing extra security features, called capabilities, which have been added to the 2.1.x and 2.2.x kernels. Still, not a lot of software takes advantage of these new features [Ref. 13]. The premise is to strip out the root, or super-user account (almost every attack revolves around obtaining "root") and replace it with a concept of Least Privilege. The idea behind Least Privilege is to create a system where processes are given specific abilities that are limited solely to the tasks that they are performing.

The main advantage of using capabilities is its simplicity. With just a couple of bits, many permission scenarios can be modeled. Also, without the existence of root, many of today's Linux security vulnerabilities can be eliminated since there would be no root account to access. As stated earlier, the root account is trusted and can bypass system security mechanisms. It has all DAC permissions, thus overriding access controls on all Linux objects, and often, other implemented security settings.

G. SUMMARY

Although Linux and XTS-300 are both UNIX-like Architectures, Linux is not designed as a trusted computing system. The XTS-300 implements Mandatory and Discretionary security policies whereas Linux only implements minimal DAC. Linux has no concept of a TCB or trusted and untrusted code, therefore the system administrator can not implement security policies and procedures related to a trusted computing system.

Consequently, when porting from the XTS-300 to Linux, all of the MAC security polices that the XTS-300 offer are lost. Linux will only execute the TCB Extension and SSS Servers in the user domain (Ring 3). Table 2-1 summarizes the architectural abstractions and Security Models implemented in the XTS-300 and Linux.

Environment	**Arch. Abstraction (Rings)**	**Security Models**			**Kernel**
		DAC	**MAC**		
Linux	0,3	X	None		Linux 2.2.16-22
XTS-300	0,1,2,3	X	Bell-LePadula	Biba	STOP 4.4.2

Table 2-1: XTS-300 and Linux Architectures

Chapter III will outline the how the XTS-300 to Linux port was accomplished.

III. XTS-300 and LINUX PORTING ENVIRONMENTS

A. BACKGROUND

The latest commercial variants of UNIX, such as Linux, are derived from either System V Release 4 (SVR4)[3] or 4.4 Berkley Software Distribution (BSD) release from the University of California at Berkley (4.4 BSD).

SVR4 is a product of AT&T's UNIX System Laboratories. It is a merging of AT&T UNIX System Release 3.2, Sun Microsystems SunOS system, the 4.3BSD release from the University of California, and the Xenix system from Microsoft. The source code was released in late 1989 with the first end-user copies being available during 1990 [Ref. 18]. It is almost a total rewrite of the System V kernel and produced a clean, if complex implementation [Ref. 3].

4.4 BSD is widely used in academic installations and has served as the basis for a number of commercial UNIX products. Released in 1992, 4.4BSD is the final version of BSD to be released, with the design and implementation organization subsequently dissolved [Ref. 3].

The SVR4 and 4.4 BSD implementations tend to agree on some common standards like IEEE's Portable Operating Systems based on UNIX (POSIX); the existing IEEE Std 1003.1, 1996 version, IEEE Standard for POSIX--Part 1.

Theoretically, UNIX applications ported to the Linux operating system can be completed with very little effort. Linux, and the GNU C library used by it, have been

designed with application portability in mind, meaning that many applications will compile simply through the use of the *make*[4] utility. Even so, it may not be trivial to port from the XTS-300 to Linux since the SVR4 UNIX interface was a major upgrade from SVR3's. Linux derives much of its functionality from SVR4.

This chapter provides an analysis of the major issues associated with porting from the XTS-300 to Linux, highlighting the differences between the Linux interfaces, which attempt to be POSIX.1 compliant and the XTS-300 interfaces which are based on the System V Release 3.

1. Linux

Linux is combination of the System V Release 4 and 4.4BSD interfaces. Linux Kernel Version 2.2, and beyond, aims to be compliant with the IEEE Std 1003.1-1988 (POSIX.1) standard. [Ref. 8]. By itself, the Linux kernel is not very innovative. When Linus Torvalds wrote the first Linux kernel, he referenced some classical books on UNIX internals, like Maurice Bach's *The Design of the UNIX Operating System* (Prentice Hall, 1986). Therefore, Linux has some bias towards the UNIX baseline described in Bach's book (i.e. SVR4). However Linux does not stick to any particular variant of UNIX. Instead, it tries to adopt good features and design choices of several different UNIX kernels [Ref. 8].

In general, Linux has been designed to be compatible with older UNIX implementations. This compatibility is intended to make application porting easier; and, in

---

[3] SVR4 draws on the efforts of both commercial and academic designers and was developed to provide a uniform platform for commercial UNIX deployment.[Ref 3]

[4] The *make* utility is a tool for organizing and facilitating the update of executables or other files which are built from one or more constituent files.

a number of instances, has improved upon or corrected behavior found in those implementations.

Moreover, Linux includes all of the features of a modern operating system, like virtual memory, a virtual file system, lightweight processes, reliable signals, SVR4 interprocess communications, support for Symmetric Multiprocessor (SMP) systems, etc.

2. XTS-300

The XTS-300's STOP operating system is designed not only to support much of the UNIX System V.3 interface for applications software, but to produce and run object programs that adhere to a subset of the "Intel386 Family Binary Compatibility Specification 2" as well [Ref. 17]. The XTS-300 includes many of the Linux features mentioned above, but many of them are provided in different ways. For example, SVR3 provides reliable signals in a way that differs from POSIX.1 [Ref. 18].

B. PORTING MODEL

The porting model used in this thesis represents a simple example of source-level porting. The XTS-300 source code is transported from its environment to Linux's, and adaptation occurs primarily in the Linux target environment. The TCB Extension Server, Secure Session Server and related source code are transported to the Linux environment, while adapting the configuration file flags, options and Macros. After various modifications to the source code in the Linux environment, the code will be compiled and linked (using gcc) in order to create useable executable files. Figure 3-1 shows the model abstraction.

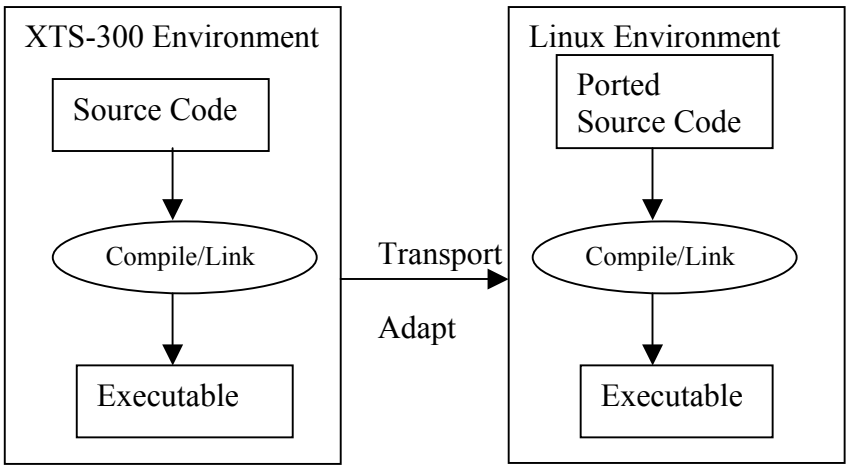

Figure 3-1: Porting Model

C. COMPARISONS

The following is a list of items in the XTS-300, which cannot be implemented without change in the target (Linux). These items have been isolated as far as possible to a small number of modules.

When isolation is not feasible -- for example, I/O statements, the functionality has been encapsulated by defining intermediate modules, making changes to libraries, defining new macros, or modifying compiler or linker flags.

1. Libraries and System Interfaces

The XTS-300 and Linux source code are linked to specify the appropriate process initialization routines and libraries that contain the proper system interfaces and library routines.

The XTS-300 uses TCB gate interfaces and OSS domain library routines defined in Ref. 12. On the XTS-300, most of the include files required for invoking the TCB gates and OSS domain library routines are located in */usr/include/stop* directory. The library contents include the OSS domain library routines, TCB gate interfaces and some standard C library routines. The OSS domain library is contained in the file */lib/liboss.a*.

The OSS domain library, liboss.a, cannot be used in the Linux environment because the source code is not available for compilation in the Linux environment. Therefore many of the OSS function calls will be replaced with function calls in Linux's libc.a library, located in */usr/lib/libc.a.* Additionally, the TCB gate calls used in the XTS-300 are replaced by Linux system interface calls.

2. Signals

The basic *signal()* function is a feature of ISO C, while *sigaction()* is a function defined as part of the POSIX.1 standard. Although *sigaction()* would be the preferred function, *signal()* is used in some instances to support cleaner portability.

It is possible to use both the *signal()* and *sigaction()* functions within a single program, but caution must be observed because they can interact in slightly strange ways. The *sigaction()* function, through its arguments, specifies more information than the *signal()* function, so the return value from *signal()* cannot express the full range of *sigaction()* possibilities. Therefore, if *signal()* is used to save and later reestablish an action, it may not be able to properly reestablish the handler that was initially created with *sigaction()*. If both *sigaction()* and *signal()* must be used together, the problem can be avoided by always using *sigaction()* to save and restore a handler if a program uses *sigaction()* at all. Since

sigaction() is more general, it can properly save and reestablish any action, regardless of whether it was established originally with *signal()* or *sigaction()*.

On some systems, if an action is established with *signal()* and then examined with *sigaction(),* the handler address that was specified by *signal()* may not be the one provided to *sigaction()*. It may not even be suitable for use as an action argument with *signal()*. But it can be relied upon if it is used as an argument to *sigaction()*. This problem is not an issue in the targeted Linux environment.

Many of the OSS domain signal-like functions were replaced with SV4 signal function calls where practical. Since signals on SV4 platforms have some backwards compatibility with SV3 signals, in some instances SV3 signals were used in order to make the change cleaner. Table 3-1 lists the XTS-300 signal calls that were transformed to work in the Linux environment.

XTS-300	Linux Translation
set_ipc_event_handler (ANY_EVENT, &(sig_event));	signal(ANY_EVENT, &(sig_event));
suspend_event (ANY_EVENT,0,0,NULL,NULL,NULL);	wait(ANY_EVENT);
send_event(getppid(), RAISE_SIGNAL_EVENT,NULL,0);	kill(getppid(),RAISE_SIGNAL_EVENT);
set_ipc_event_handler(RAISE_SIGNAL_EVENT, handler);	sigaction((RAISE_SIGNAL_EVENT, , *sigact.sa*_handler ,NULL);
set_ipc_event_handler(TERMINATE_PROCESS_EVENT, handler);	sigaction(TERMINATE_PROCESS *EVENT, sigact.sa*_handler,NULL);
suspend_event(ANY_EVENT, ONE_SECOND,0,NULL,NULL,NULL)	sleep(1);

Table 3-1: Linux and XTS-300 Signal Translations

3. System V IPC

In the XTS-300 environment, various Inter-Process Communication (IPC) mechanisms exist for OSS domain programs and for Application domain programs (CASS

programs). Some of these mechanisms are only available for one type of program, and therefore, cannot be used as an IPC mechanism between domains.

Linux does not have an OSS or CASS domain, however programs that run in the CASS domain on the XTS-300 can run in the user domain (Ring 3) in Linux. OSS domain functions ported to Linux are replaced with equivalent SV4 IPC functions for messages, shared memory and semaphores.

a. Shared Memory

When compiling, there is a conflict resolving the storage size of the structure *shmid_ds* defined in /stop/sys/shm.h. It was not clear why this conflict exists, but the solution was to prohibit use of the shm.h file used by the /include directory in the XTS-300 and instead use /usr/include/sys/shm.h in Linux.

b. Messages

Table 3-2 shows the messaging IPC mechanisms. The XTS-300 mechanisms can only run in the OSS domain (Ring 2). Therefore, they are translated to POSIX.1 compliant mechanisms capable of running in the Linux user (Ring 3) domain.

XTS-300	Linux Translation
send_ipc_message(procuid id, ring target ring, bool upgrade_allowed, ipc_event_type message_type, *far const void *data, size*t data size);	msgsnd(int msgid, const void *ptr, size_t nbytes, int flag;)
receive_ipc_message(_far ipc_event_type *message_type, *far procuid *sending*process, *far ring *sending_ring, far void *data, far size_t *data_size*);	msgrcv(int msgid, void *ptr, size_t nbytes, long type, int flag)

Table 3-2: XTS-300 and Linux Message Translations

c. Semaphores

The *semget* and *semop* IPC mechanisms are available for both OSS domains and Application domain programs. The only major change with semaphores is that the union variable

*semun{int val; sem_ds *buf;unsigned short *array}sem_arg*;

was added in /util/msem.c. Also external functions *semget(ket_t, int,int)* , *semop(int,sem_operation, int),* and *semctl(int, int, int, union semun arg)* are commented out because there is no advantage declaring these as external functions when all references to them can be made via declarations in the /include/msem.h header file.

4. Access Control

SSS programs must take on the user/group access class of the user of the TCBE. In order to accomplish this, the XTS-300 kernel call (TCB-Gate) *set_user_group(ushort euid, ushort egid,ushort ruid, ushort, rgid)* is used. This function sets the real and effective user/group identifiers for the current process. In the port to Linux, the function is replaced with *setuid(uid_t uid)*. If groups need to be addressed, the *setgid(gid_t gid)* may be used.

5. Variables

The GNU C compiler (gcc) in Linux returned numerous errors during the porting process because function-local variables were not declared at the beginning of the functions. This problem was resolved by placing local variable declarations at the beginning of functions.

There are many variable types declared in the XTS-300 STOP directory header files that are also declared in Linux header files. When such conflicts developed, the XTS-300

declared variables were commented out in favor of Linux declarations, many of which are declared in /usr/include/sys/types.h.

6. Processes

In Linux, the *fork()* function is used to create a child process. The XTS-300 uses *fork_process()*. However, the exact semantics of how it works, how much of the parent's environment, and how many of its facilities are inherited by the child process are the same as those for the Linux mechanism.

The XTS-300 *load_process()* TCB gate (system call) provides the mechanism for starting a process with a new personality. Linux does not have a *load_process()* system call, therefore a *fork()/exec()* sequence was used to obtain the equivalent results.

7. Networking

The XTS-300's OSS Domain uses the socket library (libsocket) for client/server communications by specifying the –lsocket argument when compiling. However, when ported to Linux, this library is not required and all of the socket functions are in Linux's libc.a library in directory */usr/lib/libc.a*.

8. Reference and Debug File Locations

All references to logs and debug files were changed to directory /home/mvglover/thesis/logs.

D. SUMMARY

This chapter provided an analysis of the major issues associated with porting from the XTS-300 to Linux, highlighting the differences between the interfaces of the two systems.

In theory, UNIX applications ported to the Linux operating system can be completed with very little effort. However, the XTS-300 implements various Signals and IPCs that are specific to its TCB, thus not a found in POSIX implementations. These program communication mechanisms elicit the majority of changes when porting from the XTS-300 to Linux.

The port environments are outlined in Table 3-3.

	Linux	**XTS-300**
Kernel	2.2.16-22	STOP 4.4.2
C Library	libc 2.192	Non-standard
Compiler	gcc 2.96-54 (which gcc; rpm –qf /usr/bin/gcc), based on GNU CC 2.95.2	Non-standard
Editor	GNU emacs 20.7.1	N/A
API	Mostly Compliant with POSIX.1	UNIX System V version 3
C Standard	ANSI C	ANSI C
Operating System	Red-Hat Linux 7.0	XTS-300
Java Runtime Environment	J2sdk-1.4.0 (needed for TCBE test client)	N/A
Processor Type	Pentium III 700 (i686)	Pentium 75Mhz

Table 3-3: Linux and XTS-300 Port Environments

IV. PORTED MECHANISMS

A. DESCRIPTION OF PORTED SERVERS

1. TCB Extension Server

This section provides a synopsis of the TCB Extension Server description provided by Wilson, [Ref. 1].

The Naval Postgraduate School previously developed the TCB Extension Server process. As shown in Figure 1.1, its purpose is to extend the TCB perimeter securely over the network to the requesting TCBE-equipped workstation. The TCB Extension Server process is comprised of a single parent and multiple child processes that are responsible for accepting connections from the TCBE-equipped client workstations. The parent process will initially listen on an assigned port for incoming secure attention key (SAK) requests initiated from the user at the remote TCBE-equipped client. Once a request is received, the parent process will verify the identification and authorization of the requesting TCBE. If the verification is successful, a child process is created and the parent is able to relinquish control of the communications to the child. This frees the parent to listen for new connection requests. If the Identification & Authorization is in error, the connection is terminated and no child is created [Ref. 4].

Each TCBE connection to the MLS LAN is therefore assigned an individual child TCB Extension Server process that will handle all of the security-related operations necessary to establish and maintain a session on the MLS LAN. The current MLS LAN

design enables the child process to present the user with menus, with which he or she may conduct all trusted path security related operations such as “login” and “session negotiation”. This process also controls the actions of the connected TCBE through specific TCBE state commands. At any time, the user make invoke the Secure Attention Key (SAK) which will prompt the TCB Extension Server to interrupt the current running process, verify the TCBE, and present an appropriate menu of trusted path operations.

a. TCB Extension Server on Linux

In the XTS-300, the TCB Extension Server executes in the OSS domain, Ring 2 (where CASS and Trusted software execute), of the XTS-300. However, the port to Linux requires execution of the TCB Extension Server is in the user domain, (Ring 3) using ported CASS domain services. Yet, there are instances where OSS domain services will be required. When OSS services are required, they need to be modified to run in the Linux environment.

2. Secure Session Server (SSS)

This section recaps the Secure Session Server description provided by Wilson [Ref. 1].

Previously developed at The Naval Postgraduate School, the Secure Session Server process is comprised of a single parent and multiple child processes for each platform on which a given application protocol is hosted The Secure Session Server parent process is responsible for accepting connections from TCBE-equipped client workstations and establishing the TCP/IP protocol services for the user. Up to ten child processes can be used per session to service a connection for a parent SSS. The parent process will initially listen

on an assigned port for incoming requests for protocol service. Once a request is received, the parent processes will verify the user's MLS LAN session with the User Database. Each protocol service request is assigned an individual child Secure Session Server process that will handle all of the protocol transmissions to and from the Application Protocol Server. The child process is responsible for the creation of a unique Application Protocol Server process tied directly to the user through a handle created from the session data received from the Session Database Server (username, session level). The User Database[5] is used to bind communications from a particular TCBE to a specific user and session level [Ref. 4]. If the verification is successful, a child process is created, with a valid session, to service the connection. This frees the parent to listen for new connection requests. If the verification is in error, the connection is terminated and no child process is created. [Ref. 4].

a. Secure Session Server in Linux

In the XTS-300, the Secure Session Server executes in the OSS domain, Ring 2 (where CASS and Trusted software execute), of the XTS-300. However, the port to Linux requires execution of the SSS executes in the user domain, (Ring 3) using ported CASS domain services. Yet, there are instances where OSS domain services will be required. When OSS services are required, they need to be modified to run in the Linux environment.

Also, in the current Linux-based implementation, the Secure Session Server observes the session level in the User database. In the current ported configuration, the session level indicates a security level four and integrity level zero. However, the Linux

[5] This database is initialized (a new record will be created) each time a user on a valid TCBE logs in and is updated when the user changes session level, or initiates a session. The entry for a user is cleared when the user logs out. It is maintained (readable/writable) by the TCB Extension Server and shared (readable) by the SSS. The SSS reads this database to obtain the user name, session level, and session status, each time the SSS receives a protocol session request from a client.

operating system does not support security and integrity levels. Therefore, in the current port, the handle (descriptor) will be based only upon the socket creation between the remote client and the Secure Session Server. The security and session levels may be re-implemented if the SSS is ported to future Linux platform that supports those security mechanisms.

B. DESCRIPTION OF PORTED DATABASES

The following lists of databases files are required for the port to the Linux environment. They provide data necessary for the compilation and execution of the TCB Extension and SSS Servers. The descriptions of the databases are from Shifflett [Ref. 4].

1. proto_list

The 'Protocol DB' is used to list the application protocols that will be provided by the MLS LAN. The 'Protocol DB' will contain the following fields:

- Protocol ID – a descriptive identifier of the protocol
- Port Num – the TCP port number to listen to for service requests
- Path – the path to the untrusted executable that encodes the protocol service (e.g. IMAP, sendmail, etc.)

This database is:

- Administratively set.
- Used to start all necessary SSS processes
- Managed by the 'Allowed Protocols' module.

2. tcbe_list

The ‘tcbe_list” file is a list of TCBEs that can be used to login to the MLS LAN TCB. The ‘tcbe_list’ will contain the following fields:

- TCBE ID – this field will be a unique identifier of the TCBE instance
- A potential future enhancement would include session level minimum and maximum for each TCBE, and/or a list of users allowed using each TCBE.

The TCB Extension Server will use this database to restrict login access (e.g. trusted path access) to the MLS LAN TCB.

This database is administratively set and cannot be modified by the TCB Extension Server, or the SSS.

This database will be read (cached) by the TCB Extension Server on startup.

This database will be managed by the ‘Allowed TCBE’ module.

3. pmap_db.txt

The ‘Pseudo-Socket Map DB’ is used to map a particular user and session level to a Pseudo-Socket (PSKT). This database will contain the following fields:

- User ID – the ID of a user
- Session level – the session level of the user
- Handle – a handle to a PSKT.

This database will be maintained by the SSS child processes. Since this database is shared by multiple processes, a locking capability will be needed to prevent more than one SSS child process from allocating the same PSKT handle.

This database will be read-only by the protocol server processes that use it to obtain the PSKT handle.

This database will be managed by the 'PSKT Map' module.

C. SUMMARY

The Naval Postgraduate School previously developed the TCB Extension and Secure Session Servers for use on the XTS-300. The TCB Extension Server's purpose is to extend the TCB perimeter securely over the network to the requesting TCBE-equipped workstation. The SSS parent process is responsible for accepting connections from TCBE-equipped client workstations and establishing the TCP/IP protocol services for the user.

In the XTS-300, both servers execute in the OSS domain. In the Linux environment, they execute in the user domain, (Ring 3).

There are also four database files that need to be copied to the Linux environment. These databases provide services such as users databases, allowed clients, allowed protocols and socket services.

V. SOURCE CODE TRANSFORMS TO LINUX ENVIRONMENT

A. INTRODUCTION

The ported software has the directory structure shown in Fig. 5-1. All preprocessor directives and directory searches are made based upon this structure.

Big programs are made up of many modules [3]. These modules provide the user with functions and data structures that are to be used in a program. The modules in the port essentially come in two parts: the interface and implementation. The interfaces that specifies what the TCB Extension and SSS modules do are located in the /home/mvglover/thesis/include and /home/mvglover/thesis/stop directories, while the implementations are specified in the /home/mvglover/thesis/tps, /home/mvglover/thesis/util and /home/mvglover/thesis/sss directories. The interface declares all of the data types, function prototypes, global information, macros, or whatever the module requires. The implementation adheres to the specifications set forth by the interface. This is how the XTS-300 port is designed.

The ~/logs directory contains the databases required for the TCB Extension and SSS servers and it also contains the debugging logs used when the debugging flag option is set.

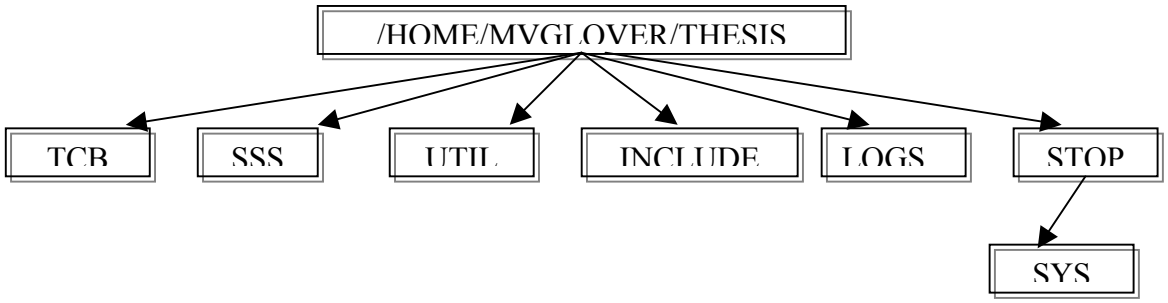

Figure 5-1: Ported Directory Structure

B. COMPILATION AND LINK RESOLUTIONS

The following tables list the significant compile and link errors that occurred during the XTS-300 to Linux port.

1. STOP

This directory contains most of the include files required for invoking TCB-Gates (STOP kernel calls) and OSS domain library routines.

a. accsess.h

Compile/link/logical error	Location	Resolution	Remarks
structure *access* is redeclared as a different kind of symbol and conflicts with a definition of access in /usr/include/unistd.h.	Line 94	Renamed variable *access* to *accessinfo*	All instances of structure variable *access* in source code changed

b. Message.h

Compile/link/logical error	Location	Resolution	Remarks
-DAEMON_NAME undeclared here (not a function) -size of array “daemon_name” has non integer type	Line 94	In /stop/limits.h, commented out “#if defined (SUBSYSTEM_DEF)” block	

c. limits.h

Compile/link/logical error	Location	Resolution	Remarks
In /stop/message.h DAEMON_NAME undeclared		commented out "#if defined(SUBSYSTEM_DEF)" block	
CLK_TCK already defined in time.h.	64	Commented out #define CLK_TCK	CLK_TICK in limits.h is required for a SUBSYSTEM_DEF

d. tcb_gates.h

Compile/link/logical error	Location	Resolution	Remarks
sss.c, received parse error that **ulong** not recognized	Line 65	added #include </usr/include/sys/types.h>	Used Linux ulong definition
Structure *access* in /stop/access.h redeclared as a different identifier	Lines 61,63,77,207,273,282	Changed instances of 'access' to 'accessinfo'	

e. Stdtyp.h

Compile/link/logical error	Location	Resolution	Remarks
FALSE redefined; already defined in /usr/include/sys/types.h	55	Commented out and added #include </usr/include/sys/types.h>	Warning, not error
TRUE redefined; already defined in /usr/include/sys/types.h	56	Commented out and added #include </usr/include/sys/types.h>	Warning, not error
ushort redefined; already defined in /usr/include/sys/types.h	39	Commented out and added #include </usr/include/sys/types.h>	Warning, not error
ulong redefined; already defined in /usr/include/sys/types.h	40	Commented out and added #include </usr/include/sys/types.h>	Warning, not error

2. TCB Extension Server Module

a. Makefile

The –I../stop option was added to the INC_OSS and INC_CASS flags.

The domain flags,

CFLAGS_OSS = -oss -DOSS_OPTION

CFLAGS_OSS_DAEMON = -oss -DOSS_OPTION –daemon -DIS_DAEMON

were commented out so that source code that executes in the OSS domain will only use CASS domain services.

In the tps and tpsd compile commands the –lsocket option was removed because this library is not required. All of the socket functions are in Linux's libc.a library in directory */usr/lib/libc.a*.

b. tps.c

In tps.c, there are various OSS Domain Library functions that need to be changed to conform to Linux signals and messages.

Compile/link/logical error	Location	Resolution	Remarks
		Added #include "..stop/message.h"	
		<signal.h> <sys.ipc.h> <sys.msg.h>	needed for sysV.4 IPC calls
		Commented out; #include <sys/byteorder.h>	Warning, not error
		Changed the directory location for the log files	
send_event(child_pids[idx],TERMINATE_PROCESS_EVENT, NULL, 0);	296	Replaced with kill(child_pids[idx],SIGSTOP);	Needs sysV.4 signal
receive_ipc_message((_far ipc_event_type *)(&e_type), (_far procuid *)(&pid), (_far ring *)(&ring_num) (_far char *)data, (_far_size_t *)(&data_len));	265	msgrcv(((_far ipc_event_type *)(&e_type), (_far procuid *)(&pid), (_far ring *)(&ring_num) (_far char *)data, (_far_size_t *)(&data_len));	
set_ipc_event_handler (ANY_EVENT, &(sig_event));	115	signal(ANY_EVENT, &(sig_event));	Needs sysV.4 signal
suspend_event (ANY_EVENT,0,0,NULL,NULL,NULL);	120	wait(ANY_EVENT);	Needs sysV.4 signal

c. tps_utilc.

Compile/link/logical error	Location	Resolution	Remarks
		Added #include "..stop/message.h"	needed message declarations
		Added <time.h>	
		Added <types.h>	Needed Linux type declarations
set_ipc_event_handler(ANY_EVENT, &(sig_event));	1119	signal(ANY_EVENT, &(sig_event));	
set_ipc_event_handler(ANY_EVENT, &(sig_event));	1120	signal(ANY_EVENT, &(sig_event));	
*get_trusted_info			

3. INCLUDE

a. level.h

In the XTS-300, the level.h header file is not in /include directory. However, for the Linux port it was added because many of the access.h is dependent on level.h for many variable declarations.

b. userdb.h

Compile/link/logical error	Location	Resolution	Remarks
parse errors in function prototype int *user_logged_in()*	Line 118	Added <access.h> to else block of macro at beginning of file.	
parse errors in function prototype int *get_session_level()*	140	Same as above	
parse errors in function prototype int *set_session_level()*	172	Same as above	

c. util.h

Compile/link/logical error	Location	Resolution	Remarks
parse errors in function prototype char **disp_sess()*	Line 96	Added <access.h> to else block of macro at beginning of file user_db.h	

parse errors in function prototype void *get_network_level()*	104	Same as above	
parse errors in function prototype void *_get_current_level()*	112	Same as above	
parse errors in function prototype int *check_access()*	122	Same as above	

d. msem.h

Compile/link/logical error	Location	Resolution	Remarks
parse error before *access_ma*	Lines 38,48	Added <access.h> to else block of macro at beginning of file	

e. pmap_db.h

Compile/link/logical error	Location	Resolution	Remarks
parse error before *access_ma*	Lines 66,79,101,123	Added <access.h> to else block of macro at beginning of file	

f. pskt.h

Compile/link/logical error	Location	Resolution	Remarks
parse error before *access_ma*	Line 62	Added <access.h> to else block of macro at beginning of file	

g. shm.h

Compile/link/logical error	Location	Resolution	Remarks
Parse error before *access_ma* when compiling user_ia.c	Line 40	Added <access.h> to else block of macro at beginning of file	
Parse error before *access_ma* During compile of shm.c	Line 50,64	Added <access.h> to else block of macro at beginning of file	

h. alw_tcbe.h

Compile/link/logical error	Location	Resolution	Remarks
		Changed default location of TCBE_FILE to /home/mvglover/thesis/logs/tcbe_list	Old path defined for XTS-300

i. hrl_db.h

Compile/link/logical error	Location	Resolution	Remarks

parse error before *access_ma*	Lines 79,148	Added <access.h> to else block of macro at beginning of file	

4. UTIL Module

a. Makefile

The domain flag,

CFLAGS_OSS = -DOSS_OPTION

was commented out so that source code that executes in the OSS domain only use CASS domain services.

In order to build the lbutil_oss.a library, the –oss flag was removed. This was done because of an error stating that too many –c –o or –S flags in multiple compilations. It seems that this error occurred because the compiler did not recognize the –oss flag and thought that that the user was using the –o flag twice.

The –oss flag is necessary so that the /stop directory and the liboss.a library can be utilized. However, the Makefile was changed so that /stop directory is searched, and the liboss.a library is not used in the Linux port. Therefore the –oss flag is not necessary.

b. buff_io.c

Compile/link/logical error	Location	Resolution	Remarks
parse errors idx	Lines 153,213,340		

c. menu.c

Compile/link/logical error	Location	Resolution	Remarks
parse errors idx	Line 100		

d. msem.c

Compile/link/logical error	Location	Resolution	Remarks
Conflicting types for *semop()* and *semctl*(), function prototypes	75-77	Commented out *semop(), semctl()*, and *semge()t* external function declarations.	Commented out external value *semget*() so of SYS V semaphore functions are used consistently
Parse error before union, *sem_arg* undeclared	178	Added union added *semun{int val; sem_ds *buf;unsigned short *array}sem_arg;*	

e. pmap_db.c

No Change

f. pskt.c

Compile/link/logical error	Location	Resolution	Remarks
Dereferencing pointer to incomplete type	Line 1257	commented out countdown = timeout->tv_sec	
Dereferencing pointer to incomplete type	Line 1336	commented out countdown = timeout->tv_sec	

g. shm.c

Compile/link/logical error	Location	Resolution	Remarks
parse error for the *shmid_ds* in header file shm.h		Commented out <sys/shm.h> and specifically added </usr/include/sys/shm.h	This also got rid of an error at lines 115,189,265, that stated that 'storage size of *shmid_ds* isn't known.

h. alwe_tcbe.c

No Change

i. priv_util.c

Compile/link/logical error	Location	Resolution	Remarks
Parse error before *access_ma*	Line 26	Added <access.h> to else block of macro at beginning of file	

j. user_db.c

No Change

k. user_ia.c

Compile/link/logical error	Location	Resolution	Remarks
'uauth_entry' undelcared	45	In the /stop/user_group.h file I commented out the if "SUBSYSTEM_DEF = = SS_TRUSTED || defined (NEED_AUTH)"	
Link Error when compiling tps.c: Undefined references to *Get_user_numbe()* *Get_uauth_entry()* *Level_valid_for_user()*		Commented out all code in body of function	

All source code that referred to privilege code was deleted since privileged code would not be run in the Linux environment.

l. util.c

Compile/link/logical error	Location	Resolution	Remarks
Link Error when compiling tps.c: Undefined references: *Get_current_leve()*		Commented out section that addressed privilege code	Need to get source code or library for these function call. Library is cass.a

The *getlevel()* function call was deleted and the include file </stop/access.h> was added.

5. SSS Module

a. Makefile

The XTS-300 environment, the compiler used options flag *–I /usr/include/sys*, however this option is not required in the Linux environment because the

GNU gcc in Linux Redhat 7.0 by default searches for this directory path. Also, –I../stop was added so that the /stop directory would be searched.

The domain flags,

CFLAGS_OSS = -DOSS_OPTION –daemon –DIS_DAEMON

were commented out so that source code that executes in the OSS domain will only use CASS domain services.

In the Makefile configuration file, *getpwnam* and *getpwent* are commented out, because the function calls are already defined in libc.a. Also usr/libc is changed to conform to the Linux path /usr/include/libc.

b. sss.c

Compile/link/logical error	Location	Resolution	Remarks
		Commented out sys/byteorder.h	
Undefined variable errors for NO_GROUP_CHANGE and NO_USER_CHANGE	Line 127	Replaced with *setuid(pwent->pw_uid)*	
set_ipc_event_handler(RAISE_SIGNAL_EVENT, &(sss_sig_event)); undefined		Replaced with *signal((RAISE_SIGNAL_EVENT, &(sss_sig_event));*	
suspend_event(ANY_EVENT, ONE_TENTH_SECOND,0,NULL,NULL,NULL) undefined	229	Replaced with *sleep(.1);*	Link Error; Also ONE_SECOND is defined as 1000000/128L in kernel.h
the_err = *send_event(sss_pids[idx], TERMINATE_PROCESS_EVENT,NULL,0)*; undefined	253	Replaced with the_err = *kill(sss_pids[idx], TERMINATE_PROCESS_EVENT);*	
Commented out the_err = *send_event(getppid(), RAISE_SIGNAL_EVENT,NULL, O);* undefined	262	Replaced with the_err = *signal(getppid(), RAISE_SIGNAL_EVENT);*	
Commented out *print_error("send_event_error :",the_err);* undefined	266	Replaced with *debugd("send_event_error :",the_err);*	

Commented out: #define *sleep(a) suspend_event(NO_EVENT,(a)* ONE_SECOND, 0,NULL,NULL,NULL);*		Replaced with *sleep(a)*	

c. sss_util.c

Compile/link/logical error	Location	Resolution	Remarks
		Commented out sys/byteorder.h	
Error: variable-size type declared outside of any function in statement temp_buff[read_limit + 1];		Changed 'read limit +1' to '4097'	
Struct timeval was not defined		Added #include <time.h>	
	410	Changed directory of ssschild_%d.tmp to /home/mvglover/thesis/logs/	
Undefined variable errors for NO_GROUP_CHANGE and NO_USER_CHANGE	Line 472	Replaced with setuid(pwent->pw_uid)	
send_ipc_message(child_proc,2,TRUE,DEVICE_AVAILABLE EVENT,(_far const void *) NULL,0)… undefined	521	msgsnd(child_proc,(_far const void *)NULL,0,DEVICE_AVAILABLE_EVENT);	Args 2 and three from send_ipc_message are not necessary in Linux translation.
set_ipc_event_handler(ANY_EVENT), &(ssd_sak_handler)); undefined	181	Replaced with signal(ANY_EVENT, &(ssd_sak_handler));	Link error
Commented out suspend_event(ANY_EVENT,0,0,NULL,NULL,NULL) undefined	185	Replaced with sleep(0);	Link error
Commented out suspend_event(ANY_EVENT, ONE_SECOND,0,NULL,NULL, NULL)	318	Replaced with sleep(1);	Link Error Also ONE_SECOND is defined as 1000000/128L in kernel.h
Commented out the_err = send_event(sss_pids[idx], TERMINATE_PROCESS_EVENT,NULL,0);	352	Replaced with the_err = kill(sss_pids[idx], TERMINATE_PROCESS_EVENT);	
Commented out suspend_event(ANY_EVENT, 5*ONE_TENTH_SECOND,0,NULL,NULL,NULL)	189	Replaced with sleep(.5);	Link Error Also ONE_SECOND is defined as 1000000/128L in kernel.h
Commented out print_error("send_event_error :",the_err);	409	Replaced with debugd("send_event_error :",the_err);	
Commented out the_err = send_event(getppid(), RAISE_SIGNAL_EVENT,NULL,0);	406	Replaced with kill(getppid(),RAISE_SIGNAL_EVENT);	

Link_error: undefined reference to *load process(far const char *)my_aps_path,my_user_sess, FALSE, (_far short *)NULL,(_far procuid *)(&child_proc));*	509	Used fork()/exec() sequence: if((child_proc = fork()) < 0){ err_sys("fork_error"); done=TRUE; } else if (child_proc == 0){ if (execle((_far const char *)my_aps_path,NULL,NULL,NULL,NULL) < 0) err_sys("execle error");…….	*load_process()* is a TCB gate system call which requires the oss.a library, which not available in Linux Environment
Commented out set_ipc_event_handler(RAISE_SIGNAL_EVENT, handler);	540	Changed to sigaction((RAISE_SIGNAL_EVENT, *sigact.sa*_handler ,NULL);	
Commented out set_ipc_event_handler(TERMINATE_PROCESS_EVENT, handler);	542	Changed to sigaction(TERMINATE_PROCESS *EVENT, sigact.sa*_handler,NULL);	
Commented out set_ipc_event_handler(TERMINAL_ATTENTION_EVENT, handler);	544	Changed to sigaction (TERMINAL_ATTENTION, *sigact.sa*_handler ,NULL);	
Commented out suspend_event(ANY_EVENT, ONE_TENTH_SECOND,0,NULL,NULL,NULL)	189	Replaced with sleep(.1);	
Commented out suspend_event(ANY_EVENT, ONE_SECOND,0,NULL,NULL, NULL)	189	Replaced with sleep(1);	

d. ssd.c

Compile/link/logical error	**Location**	**Resolution**	**Remarks**
Commented out sys/byteorder.h			
Commented out: #define sleep(a) suspend_event(NO_EVENT,(a)* ONE_SECOND, 0,NULL,NULL,NULL);		Will use sleep(a) function	
Commented out: the_err = set_user_group(pwent->pw_uid,NO_GROUP_CHANGE,pwent->uid,NO_GROUP_CHANGE);	Line 131	Changed to the_err=setuid(pwent->uid);	
Commented out: the_err = set_user_group(orig_uid,NO_GROUP_CHANGE,orig_uid,NO_GROUP_CHANGE);	Line150	Changed to the_err=setuid(orig_uid);	
ANY_EVENT in function suspend_event(ANY_EVENT,0,0,NULL,NULL,NULL) is undeclared.	Line 181	In message.h file, removed marco: #if SUBSYSTEM_DEF==SS_TRUSTED	

E. SUMMARY

The changes outlined in this chapter represent the major changes that were made to the ported modules so that compilation and linking could be executed. After the changes were made the TCB Extension and SSS servers had basic interoperability, however, MAC policies could not be enforced.

THIS PAGE IS INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND RECOMMENDATIONS

A. SUMMARY OF FINDINGS

The premise of this thesis has shown that the TCB Extension Server and Secure Session Server can be successfully ported to a Linux Environment. Based upon this port, these are the findings.

Because the Linux API has some backwards compatibility to the XTS-300 interface, it readily facilitated many of the services of the XTS-300. Most of the difficulties that were encountered were due to interfaces requiring services specific to the XTS-300 or when some CASS functionality depended on Trusted Services.

Now that it has been demonstrated that the Linux environment can run and support many of the TCB Extension Server and SSS functionalities, many modern OS support of functionalities such as concurrency, multiprocessor support, etc. can be supported in a future implementation.

The Linux kernel will need to incorporate some additional internal dependencies in order to support MAC polices in the TCB Extension Server and Secure Session Server. The current security policy enforcement in Linux is based upon a minimal DAC policy. Therefore the current port to Linux does not support the security policy enforcement required within the MLS LAN project.

B. LINUX SECURITY POLICY ENFORCEMENT CHALLENGES

There are many challenges in extending Linux's functionality to support a secure operating system. The most straightforward attempt would be to use a micro-kernel kernel based approach. Micro-kernel operating systems demand a very small set of functions from the kernel, generally including a few synchronization primitives, a simple scheduler, and interprocess communications mechanisms [Ref. 8]. However even this approach presents some serious challenges, because of Linux's current kernel size and hardware abstraction.

The Linux kernel is monolithic. It is a large, complex program, composed of several logically different components,[Ref. 8], and large kernel data structures. There are approximately 1.5 million source lines of code in *.c, *.h, and *.S files. Therefore verifying that the kernel correctly enforces the protection requirements of the security model means examining all possible violations of kernel security. This would be a daunting task.

Also Linux's hardware abstraction is designed to support only two execution modes, user and kernel. There are no intermediate domains that can run trusted processes that implement trusted functions but do not need to be part of the policy enforcement mechanism of the kernel. Based upon the current architecture, if trusted processes are to run either the kernel size has to be increased or the trusted process has to be a user domain process.

C. PERFORMANCE ISSUES

There did not appear to be any degradation in performance of the TCB Extension Server and SSS when run in the Linux environment. However, quantification of the performance is imprecise since the XTS-300 is running on Pentium verses a Pentium III in the Linux environment. Even so, Linux should perform better, because the XTS-300

implements a security kernel, thus yielding some performance advantages in comparison to Linux. For example, the XTS-300 adds further intermediate levels (Rings 1,2) between the user and OS resources causing more explicit message passing. Moreover, it is difficult to implement a smaller and less complex Linux kernel that can isolate the necessary security functions (i.e. dependencies exist in the XTS-300 between security and nonsecurity functions -- CASS and Trusted Software).

D. FUTURE DESIGN CONSIDERATIONS

1. Security Enabled Linux

A vast amount of research and development has gone into extending UNIX-like systems to support security needs of various communities. In addition to the XTS-300, several other UNIX-like systems have been extended to support the U.S. military's desire for multilevel security.

Research using a Linux-based approach is new. However, there are a few promising efforts with Linux systems for creating restricted execution environments, with many different approaches.

The U.S. National Security Agency (NSA) has developed Security-Enhanced Linux (Flask), which supports defining a security policy in a specialized language and then enforces that policy.

Medusa DS9, is a free open project that extends Linux by supporting, at the kernel level, a user-space authorization server. Another open project, LIDS, is a kernel patch and admin tool to enhance the Linux kernel security by attempting to Implement a reference monitor concept and MAC policy in the kernel.

The Rule Set Based Access Control system for Linux (RSBAC) is based on the Generalized Framework for Access Control (GFAC) by Abrams and LaPadula and provides a flexible system of access control based on several kernel modules.

An addition, an NPS effort presents a Linux-based approach to provide system supporting MAC policies in a Linux environment. This security-enhanced Linux specifies the implementation of label-based interfaces for the mandatory portions of the Bell and Lapadula secrecy model and Biba integrity model [Ref. 21].

2. IPSEC Capabilites

Another goal is to implement IPsec. The results of this effort can be conducted in the context of a larger research project to provide highly protected communication channels between Clients and Servers for security critical and mission critical information transfers. It also follows work from [Ref. 1].

E. CONCLUSIONS

The port of the TCB Extension Server and SSS to the Linux platform was a success. The port process came with positive and negative experiences. The TCB Extension Server and SSS source code was well commented and the debugging code was very useful. However, knowing what modules and databases to port took a little time. This was based primarily because of the author's inexperience with porting software. Using emacs as the development environment made the work easier. Built in functions such as compilation, debugging, Current Version System (CVS) and speed-bar were very useful. The overall experience from this thesis was very positive, thanks to the talented team of NPS personnel that supported and helped the author.

APPENDIX A. GLOSSARY OF TERMS

Access: A specific type of interaction between a subject and an object that results in the flow of information from one to the other.

API: Application Programming Interface; the generalized term for a defined software interface for software applications.

Authenticate: To establish the validity of a claimed security.

Bell-LaPadula Model: A formal state transition model of computer security policy that describes a set of access control rules. In this formal model, the entities in a computer system are divided into abstract sets of subjects and objects. The notion of a secure state is defined and it is proven that each state transition preserves security by moving from secure state to secure state; thus, inductively proving that the system is secure. A system state is defined to be "secure" if the only permitted access modes of subjects to objects are in accordance with a specific security policy. In order to determine whether or not a specific access mode is allowed, the clearance of a subject is compared to the classification of the object and a determination is made as to whether the subject is authorized for the specific access mode.

BSD Berkeley System Distribution: A version of UNIX based on the AT&T System V UNIX developed by the Computer System Research Group of the University of California at Berkeley.

Capability: The ability of a process to perform certain functions. This function allows kernel programs, running as root, to drop all special root privileges or capabilities. For instance an FTP server could start as root, then drop all root-specific capabilities except the capability to open low-numbered ports such as port 20, the default FTP-DATA port.

Compartment: Think of a compartment as a container for objects and a permission for subjects. For instance, if we have a compartment for "unclassified" data, and a compartment for "secret" data, people with access to the "unclassified" compartment would not be able to access data labeled as "secret" and visa versa. People with access to both compartments could access both sets of data.

Data Integrity: The state that exists when computerized data is the same as that in the source documents and has not been exposed to accidental or malicious alteration or destruction.

Discretionary Access Control (DAC): Users are able to change the permissions or compartments of information at their discretion if they have control over the object

containing the information. For instance a user with administrator access could allow an unprivileged user access to the system's shadow password file by changing the file permissions.

Domain: The set of objects that a subject has the ability to access.

Free Software Foundation (FSF): A tax-exempt charity that raises funds for work on the GNU Project.

GNU General Public License (GPL): The GNU Project was launched in 1984 to develop a complete Unix-like operating system which is free software: the GNU system. (GNU is a recursive acronym for ``GNU's Not Unix"; it is pronounced "guh-NEW".)

Host: The generalized term for any device that can be a source or sink of information on a network. Generally, a host is a single-networked computer.

IETF: Internet Engineering Task Force; the body associated with the Internet that recommends and approves "standards" for use on the Internet.

IP: Internet Protocol; the network layer (layer 3) of TCP/IP. Network layer addresses are used by routers for routing purposes.

IPC: Inter-process Communication. Linux supports signals, pipes and the System V IPC mechanisms named after the UNIX release in which they first appeared.

IPSEC: Internet Protocol Security. An extension to TCP/IP, which allows encryption to protect all of the traffic between hosts. Free/SWAN is the beginnings of a Linux IPSEC project.

kernel thread (daemon): a kernel thread is a process that runs in kernel mode. These processes are properly part of the kernel since they have access to all of the kernel's data structures and functions, but they are treated as separate processes that can be suspended and resumed. Kernel threads do not have virtual memory, but access the same physical memory as the rest of the kernel.

Mandatory Access Control (MAC) - The system enforces access to information based on the information's sensitivity level or other criteria. So if the shadow password file mentioned above was labeled "Secret", the administrator could not allow a normal unprivileged user access to the file unless that administrator had special permission to change the file's attributes.

Object - A passive entity that contains or receives information. Access to an object potentially implies access to the information it contains. Examples of objects are: records, blocks, pages, segments, files, directories, directory trees, and programs, as well as bits, bytes, words, fields, processors, video displays, keyboards, clocks, printers, network nodes, etc.

International Standards Organization (OSI) Model: A seven-layer model of data communications protocols standardized by the ISO.

Process: A program in execution. It is completely characterized by a single current execution point (represented by the machine state) and address space.

Rainbow Series - A set of U.S. Government books on information security topics, so-called because each volume had a different colored cover. Some are available on-line at: http://www.radium.ncsc.mil/tpep/library/rainbow/

Red Book - The Trusted Networking Implementation (TNI). A criteria for building TCSEC systems in a network environment, as well as trusted networks.

RFC: Request for Comment; the document that the IETF uses to define standards for use and recommend practices in the Internet.

setuid, seteuid: The system calls under UNIX and UNIX-like systems such as Linux that allow a process to change the ID it runs under.

Secure Session Server: A process comprised of a single parent and multiple child processes for each platform on which a given application protocol is hosted. The Secure Session Server parent process is responsible for accepting connections from TCBE-equipped client workstations and establishing the TCP/IP protocol services for the user.

Security Kernel: The hardware, firmware, and software elements of a Trusted Computing Base that implement the reference monitor concept. It must mediate all accesses, be protected from modification, and be verifiable as correct.

Security Level: The combination of a hierarchical classification and a set of non-hierarchical categories that represents the sensitivity of information.

Security Policy: The set of laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information.

Subject: A subject on the system is basically a user, process or other active entity.

System V: A version of UNIX developed by the Bell Laboratories of AT&T..

TCP/IP: The protocol suite used in the Internet. The most important protocol suite used in networking.

TCB Extension Server: A server process comprised of a single parent and multiple child processes that are responsible for accepting connections from the TCBE-equipped client workstations.

Trojan Horse: A computer program with an apparently or actually useful function that contains additional (hidden) functions that surreptitiously exploit the legitimate authorizations of the invoking process to the detriment of security. For example, making a "blind copy" of a sensitive file for the creator of the Trojan Horse.

Trusted Computer System: A system that employs sufficient hardware and software integrity measures to allow its use for processing simultaneously a range of sensitive or classified information.

Trusted Computing Base (TCB): The totality of protection mechanisms within a computer system -- including hardware, firmware, and software -- the combination of which is responsible for enforcing a security policy. A TCB consists of one or more components that together enforce a unified security policy over a product or system. The ability of a trusted computing base to correctly enforce a security policy depends solely on the mechanisms within the TCB and on the correct input by system administrative personnel of parameters (e.g., a user's clearance) related to the security policy.

Trusted Path: A mechanism by which a person at a terminal can communicate directly with the Trusted Computing Base. This mechanism can only be activated by the person or the Trusted Computing Base and cannot be imitated by untrusted software.

Trusted Software: The software portion of a Trusted Computing Base.

UNIX: An operating system originally designed at the Bell Laboratories of AT&T in the 1970s

LIST OF REFERENCES

1. Wilson, J.D. *A Trusted Connection Framework for Multilevel Secure Local Area Networks*, Masters Thesis Naval Postgraduate School, June 2000.

2. Schroeder, M. D., Clark, D. D., and Saltzer, J. H., *The Multics Kernel Design Project. Proceeding of sixth ACM symposium on Operating Systems Principles*, November 1977, p. 43 – 56.

3. Stallings, W., *Operating Systems, Internals and Design Principles, Third Edition*. Prentice – Hall, Inc. 1998.

4. Shifflett, David, *Multi-level Secure Local Area Network Project Design Document Draft*. Naval Postgraduate School Center for Information Systems Research, May 2000.

5. Schroeder, M.D., Saltzer, J.H., *The Protection of Information in Computer Systems*. April 1975.

6. *The Linux Kernel Archives.* http://www.kernel.org/

7. Lehey, G., *Porting UNIX Software From Download to Debug,* O'Reilly & Associates, Inc. 1995.

8. Bovet, Daniel P., Cesati, Marco, *Understanding the Linux Kernel,* O'Reilly & Associates, Inc. 2001.

9. Bell, D.E. and LaPadula, L.J. – *Secure Computer Systems: Unified Exposition and Multics Interoperation*, MTR-2997 Rev. 1, MITRE Corp., Bedford Mass., March 1976.

10. Biba, K, *Integrity Considerations for Secure Computer Systems*, ESD-TR-76-372 ESD/AFSC,Hanscom AFB, Bedford, MA Apr.1977[NTDS AD A039324].

11. Wang Government Services, Inc., *XTS-300 Application Programmer's Reference Manual*, March, 1998.

12. Wang Government Services, Inc., *XTS-300 Trusted Programmer's Reference Manual*, March, 1998.

13. Paul D. Robertson *Serious Security for Linux?* http://www.linuxsecurity.net/feature_stories/feature_story-2.html 2/17/2000 21:59

14. Carnahan, Lisa*, POSIX Security Interfaces and Mechanisms* http://csrc.nist.gov/publications/nistpubs/800-7/node17.html#SECTION05100000000000000000

15. *Department of Defense Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, National Computer Security Center, December 1985.

16. Alessandro Rubini, *Making System Calls from Kernel Space,* Linux Magazine, http://www.linux-mag.com/2000-11/gear_01.html, November 2000.

17. Wang Government Services, Inc., *XTS-300 Release 4.4.2*, 31 March, 1998.

18. Stevens, Richard W., *Advanced Programming in the Unix Environment*, Addison-Wesley, Reading, MA. 1993.

19. *Common Criteria for Information Technology Security Evaluation Version 2.1*, Common Criteria Project Sponsoring Organizations, August, 1999

20. Garfinkal, S., Spafford, G., *Practical Unix and Internet Security,* Oreilly and Associates Inc, Cambridge, MA. 1996

21. Clark, Paul C., *A Linux Based Approach to Low –Cost Support of Access Control Policies*, Masters Thesis, Naval Postgraduate School, September 1999.

22. Bach, Maurice J*., The Design of the Unix Operating System*, Prentice Hall, Englewood Cliffs, NJ. 1990.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   8725 John J. Kingman Road, Suite 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Carl Siel
   Space and Naval Warfare Systems Center
   PMW 161
   Building OT-1, Room 1024
   4301 Pacific Highway
   San Diego, CA 92110-3127
   sielc@spawar.navy.mil

4. Commander, Naval Security Group Command
   Naval Security Group Headquarters
   9800 Savage Road
   Suite 6585
   Fort Meade, MD 20755-6585
   San Diego, CA 92110-3127

5. Ms. Deborah M. Cooper
   Deborah M. Cooper Company
   P.O. Box 17753
   Arlington, VA 22216
   d.cooper@computer.org

6. Ms. Louise Davidson
   N643
   Presidential Tower 1
   2511 South Jefferson Davis Highway
   Arlington, VA 22202
   davidson.louise@hq.navy.mil

7. Mr. William Dawson
   Community CIO Office
   Washington DC 20505
   williamf@odci.gov

8. Ms. Deborah Phillips
Community Management Staff
Community CIO Office
Washington DC 20505
deborlp@odci.gov

9. Capt. James Newman
N64
Presidential Tower 1
2511 South Jefferson Davis Highway
Arlington, VA 22202
Newman.James@HQ.NAVY.MIL

10. Major Dan Morris
HQMC
C4IA Branch
TO: Navy Annex
Washington, DC 20380
MorrisDE@hqmc.usmc.mil

11. Mr. Richard Hale 1
Defense Information Systems Agency, Suite 400
5600 Columbia Pike
Falls Church, VA 22041-3230
haler@ncr.disa.mil

12. Ms. Barbara Flemming 1
Defense Information Systems Agency, Suite 400
5600 Columbia Pike
Falls Church, VA 22041-3230
flemingb@ncr.disa.mil

13. Mr. Michael Green, Director
Public Key Infrastructure Program Management Office
National Security Agency
9800 Savage Road
Ft. Meade, Maryland 20775
rmgree2@missi.ncsc.mil

14. Dr. Cynthia E. Irvine
Computer Science Department
Code CS/IC
Naval Postgraduate School
Monterey, CA 93943
irvine@cs.nps.navy.mil

15. Mr. Daniel Warren
Computer Science Department
Code CS/Wd
Naval Postgraduate School
Monterey, CA 93943
warren@cs.nps.navy.mil

16. LCDR Mark Glover
1637 Legaye Ave.
Cardiff, CA. 92007
mvglov@aol.com

17. Mr. Aaron Judd
judda@spawar.navy.mil

18. David Shifflett
Computer Science Department
Code CS/Z
Naval Postgraduate School
Monterey, CA 93943
shifflett@cs.nps.navy.mil