NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

USE OF OPENSSH SUPPORT FOR REMOTE LOGIN TO A MULTILEVEL SECURE SYSTEM

by

Christopher Fred Herbig

December 2004

Thesis Advisor:	Cynthia E. Irvine
Thesis Co-Advisor:	Thuy D. Nguyen

Approved for public release; distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2004	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: USE OF OPENSSH SUPPORT FOR REMOTE LOGIN TO A MULTILEVEL SECURE SYSTEM			**5. FUNDING NUMBERS**
6. AUTHOR(S) Christopher Fred Herbig			
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) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author 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 (maximum 200 words)

Complex multilevel secure (MLS) architectures are emerging that require user identification and authentication services not only from multilevel connections, but from pre-existing single level networks. The XTS-400 can be used as a server in such environments. Trusted devices are required for user login via multilevel connections; however, single level remote login facilities do not require such client-side devices. Instead, a more lightweight mechanism is possible.

Remote login capabilities do not exist on the XTS-400 for use over the single level networks and this capability is a desired feature for use in complex multilevel architectures. OpenSSH is an application, developed for OpenBSD, that uses the SSH protocol to provide secure remote logins and an interactive command interface. A secure remote login application, OpenSSH, was ported to the XTS-400 in order to provide remote login capabilities.

The porting process identified differences between the original development platform for OpenSSH and the XTS-400. Solutions, in the form of source code modifications, were made to overcome problems resulting from the compatibility differences encountered during the port. Testing was conducted to ensure that the port was successful and did not violate any security policies enforced by the XTS-400.

14. SUBJECT TERMS OpenSSH, XTS-400, Remote Login			**15. NUMBER OF PAGES** 225
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

USE OF OPENSSH SUPPORT FOR REMOTE LOGIN TO A MULTILEVEL SECURE SYSTEM

Christopher F. Herbig
Civilian, Naval Postgraduate School
B.S., St. Edward's University, 2002

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 2004

Author: Christopher Fred Herbig

Approved by: Cynthia E. Irvine
Thesis Advisor

Thuy D. Nguyen
Co-Advisor

Peter J. Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Complex multilevel secure (MLS) architectures are emerging that require user identification and authentication services not only from multilevel connections, but from pre-existing single level networks. The XTS-400 can be used as a server in such environments. Trusted devices are required for user login via multilevel connections; however, single level remote login facilities do not require such client-side devices. Instead, a more lightweight mechanism is possible.

Remote login capabilities do not exist on the XTS-400 for use over the single level networks and this capability is a desired feature for use in complex multilevel architectures. OpenSSH is an application, developed for OpenBSD, that uses the SSH protocol to provide secure remote logins and an interactive command interface. A secure remote login application, OpenSSH, was ported to the XTS-400 in order to provide remote login capabilities.

The porting process identified differences between the original development platform for OpenSSH and the XTS-400. Solutions, in the form of source code modifications, were made to overcome problems resulting from the compatibility differences encountered during the port. Testing was conducted to ensure that the port was successful and did not violate any security policies enforced by the XTS-400.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACRONYMNS AND ABBREVIATIONS

ACL	Access Control List
API	Application Programmer's Interface
DAC	Discretionary Access Control
DSA	Digital Signature Algorithm
IP	Internet Protocol
JWICS	Joint Worldwide Intelligence Communications System
LAN	Local Area Network
MAC	Mandatory Access Controls
MLS	Multilevel Secure
MYSEA	Monterey Security Architecture
NIPRNET	Non-secure Internet protocol router network
OS	Operating System
PGP	Pretty Good Privacy
PKI	Public Key Infrastructure
RSA	Rivest, Shamir, Adelman
SIPRNET	Secret Internet Protocol Router Network
SSH	Secure Shell
STOP	Secure Trusted Operating Program
TCM	Trusted Channel Module
TPE	Trusted Path Extension

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank Dr. Irvine, Thuy, Jean Khosalim and David Shifflett for their help and support with this project. I would also like to thank Tanya Raven and Naomi Falby for their support and encouragement during my stay at the Naval Postgraduate School. I thank my parents and sister for their love and support.

This material is based upon work supported by the National Science Foundation under Grant No. DUE-0114018. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

It is general opinion that high assurance systems present challenging user interfaces. Therefore users tend to use untrusted, low assurance systems that do not provide sufficient security. The MYSEA project incorporates both high and low assurance systems with trusted and untrusted applications. The high assurance system used in the MYSEA project is the DigitalNet XTS-400. This system provides high assurance enforcement of policies to protect information from both unauthorized disclosure and unauthorized modification. Both commercial-off-the-shelf and open source productivity applications are provided to gain user acceptability.

The MYSEA project currently provides logon services for users in a multilevel secure LAN. These services require trusted devices at the client systems that are not available to the users on the single level networks. The motivation for porting OpenSSH is to provide users with remote access to the XTS-400 from a single level network.

The methodology used for this research involved platform analysis, source code analysis, source code modifications, debugging and integration testing. The XTS-400 provides a Linux Binary Compatible Interface. In most cases, applications developed in Linux will run on the XTS-400 with no modifications. OpenSSH is a special case, where the source code had to be modified and the modified source code had to be tested to ensure there was no major loss in functionality. Although some functionality was lost in the porting process, the goal of providing a secure remote interactive session to the user across the single level LAN was achieved. Users are constrained by the security policies enforced on the XTS-400 when they are logged in through OpenSSH as verified by the developmental testing results.

The conclusion of this study offers suggestions for future projects to extend this work.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION OF STUDY

In military and commercial contexts, information may be classified according to its criticality to its owners. The classification process involves labeling information with sensitivity levels. For the Military, the standard levels are UNCLASSIFIED, CONFIDENTIAL, SECRET, and TOP SECRET. In commerce, the levels might be PUBLIC, PROPRIETARY, and SENSITIVE. The use of these classifications is intended to prevent the unauthorized disclosure of the information requiring protection. As a military example, military plans must be kept confidential so as to defeat opposing forces. If the opposing forces discovered a secret military operational plan, then they could counter that operation and our military forces would suffer great losses in the form of lives, equipment, and technology. For a commercial example consider the following: information regarding a product of a company is labeled proprietary and this information is not intended for public dissemination; its disclosure, perhaps to competitors, could cause a company to lose its market edge and ultimately money.

Sensitive information needs sufficient protection, but access to this information needs to be granted when appropriate. What is required is a multilevel secure (MLS) architecture. One such architecture is the Monterey Security Architecture (MYSEA) project. The MYSEA project incorporates the protection mechanisms required to ensure only the authorized disclosure of information to authorized users. So far, the MYSEA project has focused on the military sector, but it can also be used in the commercial sector with little modification. To facilitate the sharing of information with appropriate and authorized uses, the MYSEA project permits access from a MLS local area network (LAN) to one of several single level networks. To aid in the sharing of information, a service was required so that users on the single level networks could access resources with the multilevel network.

The motivation for this study is to provide secure remote login capabilities for use over the single level networks in support of the Monterey Security Architecture (MYSEA) project. The MYSEA project uses the XTS-400 as its multilevel secure server

called the MYSEA server. The MYSEA server is connected to a number of single level networks and a MLS LAN. From the MLS LAN, users can use the Trusted Path Extension (TPE) devices to connect to the MYSEA server and view information at varying secrecy and integrity levels. From the single level networks, only information that has the same security classification as that of the network may be seen on these networks. Users on these networks do not have TPE devices to authenticate with the MYSEA server. Hence there is a need to provide a secure login mechanism for those users. The tool chosen to provide the remote interactive session is OpenSSH, a network security application that uses the SSH protocols to implement secure remote login capabilities.

B. PURPOSE OF STUDY

The purpose of this study is to port OpenSSH to the XTS-400 run on a multilevel secure (MLS) server to provide secure remote logins for users from different single-level networks. The XTS-400 does not provide such a capability. The XTS-400 provides a Linux Binary Compatible Interface that allows Linux programs to run on the XTS-400 with little or no modifications [DIG03b]. This study includes source code modifications and a series of developmental tests used to demonstrate that OpenSSH provides a remote shell to the user and that the security policies enforced on the XTS-400 are still enforced through OpenSSH. The organization for this study will now be discussed.

C. ORGANIZATION OF PAPER

This paper is organized as follows: Chapter I provides the purpose, motivation and organization of this study. Chapter II provides background information on the MYSEA project, OpenSSH, differences between OpenBSD and the XTS-400 and the software packages required by OpenSSH to function properly. Chapter III covers the goals of this study, the methodology used to port OpenSSH, and the modifications made to the source code for OpenSSH. Chapter IV describes the types of testing required for a software port and provides the test plans and the results for each type of test in the context of this study. Chapter V provides a summary of this study, the lessons learned from this study and future work that can extend this study.

II. BACKGROUND

This chapter provides background information relating to this project. The first section discusses the MYSEA project, and provides an overview of OpenSSH including its history, available clients, available authentication methods and its modes of operation. The second section provides information about the port of OpenSSH to the XTS-400. Within this section, features used by OpenSSH but have a different behavior on the XTS-400 are discussed. Then, the way the XTS-400 handles those features is discussed along with a brief description of security policies available on the XTS-400. The last section will cover the software dependencies of OpenSSH.

A. ARCHITECTURAL BACKGROUND

1. MYSEA Project

"[MYSEA] provides a trusted distributed operating environment for enforcing multilevel security policies, and utilization of support for incorporation of unmodified commodity productivity applications for user activities" [IRV04]. This means that the MYSEA project uses a client-server architecture where the server, called the MYSEA server, is responsible for the enforcement of security policies. This server is one of the very few specialized hardware components required by MYSEA. The other specialized hardware components are the Trusted Path Extensions (TPE) and the Trusted Channel Modules (TCM). The TPE is a device that will provide an unforgeable communications link between the server and the client machine. The TCMs authenticate network sensitivity levels to the MYSEA server so that the information received from that network may be labeled correctly. The clients are intended to have no permanent writeable storage. A Knoppix client as well as a specialized version of Microsoft Windows XP Embedded called "state-less professional" are part of the design. The use of a popular operating system such as Microsoft Windows supports user acceptance because users may continue to use their favorite, and familiar, office productivity applications. The MYSEA server uses an XTS-400 as its base. It will be discussed in a future section.

The XTS-400 provides an unforgeable communications link called a "trusted path" between the target of evaluation (TOE) security functions (TSF) and the user. The TSF is "a set consisting of all hardware, software, and firmware of the TOE that must be

relied upon for the correct enforcement of the [TOE security policy]" [DIG04]. The TOE in this context refers to the XTS-400. The trusted path can be invoked by the user with a secure attention key (SAK). The trusted path ensures to the user that he is communicating with the TSF and ensures to the TSF that it is communicating with the user. Outside the context of the MYSEA project, the XTS-400 only allows users at the console or serial terminals to invoke the SAK to use the trusted path. Within the context of the MYSEA project, the TPEs are high assurance components that allow users to login to the MYSEA server from a multilevel secure (MLS) LAN. The TPEs are not available for use on single level LANs such as the NIPRNET, SIPRNET and JWICS. There is a need to provide remote login capabilities with strong authentication over the single level LANs. The remote login utility chosen is OpenSSH, which will be described in the next section.

2. OpenSSH

a. Overview of OpenSSH

OpenSSH is the OpenBSD version of SSH, the secure shell. It is available under the OpenBSD license [SSH04]. SSH – pronounced s-s-h – is a protocol that specifies a secure way to login to a remote host [BAR01]. The creator of the SSH protocol is Tatu Ylonen, a researcher at the Helsinki University of Technology in Finland [BAR01]. There are two versions of the protocol, SSH-1 and SSH-2. SSH-1 was developed rather quickly and has numerous flaws [BAR01]. SSH-2 was developed to fix these flaws and add more functionality to the protocol [BAR01]. There is no backwards compatibility from SSH-1 to SSH-2[BAR01]. SSH-1 is also monolithic and tries to provide for confidentiality, integrity, authentication and communication of user commands and data within one single protocol. According to Saltzer and Schroeder, a secure system should be very modular based on the principle of economy of mechanism [SAL75]. SSH-2 follows this recommendation and divides the protocol into three main components: SSH-TRANS, SSH-USERAUTH, and SSH-CONNECT. In essence, SSH is a protocol and not an application; OpenSSH is an application that supports the SSH protocols.

Applications based on SSH usually only support one protocol, either SSH-1 or SSH-2. Thus certain clients may not be compatible with certain servers. OpenSSH is

an application that incorporates both protocol versions. OpenSSH clients and servers can negotiate with other clients and servers as to which protocol version to use. OpenSSH is the preferred implementation of SSH because it recognizes both versions of the protocol and it is highly portable – although the server must run on a Linux- or BSD-like system. OpenSSH clients have been developed for many popular operating systems such as Microsoft Windows and the various Linux Distributions.

A discussion of the PuTTY OpenSSH client follows.

b. SSH Clients

There are numerous clients for OpenSSH as can be seen on http://www.freessh.org. Under the Windows section of the website, 27 different listings can be found for clients that can be run on the Windows Operating System. Some of these clients are freeware and others are shareware. One popular Windows-based client is PuTTY.

PuTTY is very modular; there are separate executables for about every function available to the OpenSSH client such as an SSH client, a telnet client, a secure copy and secure FTP client, a secure tunneling client and a key-generation client. PuTTY is compatible with OpenSSH because like OpenSSH, it supports the two versions of the SSH protocol. PuTTY was developed and is maintained by a small team lead by Simon Tatham in Cambridge, England [TAT04]. PuTTY was selected as the Windows SSH client for use in this study.

c. Authentication

One important requirement for remote logins is strong authentication. OpenSSH provides eight authentication mechanisms: none, public-key, RhostsRSA, Rhosts, password, s/key, Kerberos, and PAM.

The first authentication method, none, does not perform any authentication. It allows a user to login assuming the user supplies a valid username. This method is built into OpenSSH and is part of the default mode of operation for OpenSSH. This method can be disabled by altering the OpenSSH daemon configuration to deny empty passwords.

The public-key authentication method works in the following way: the server issues a numerical challenge to the client. The client, acting on behalf of a user, must sign the challenge and send it back to the server. The server then uses the user's public key to verify the signature. If the signature can be verified, the user is authenticated. There are three types of keys used in OpenSSH: RSA, DSA and OpenPGP. RSA keys are used by the RSA cryptosystem which was developed by Rivest, Shamir and Adelman [BAR01]. DSA, which stands for digital signature algorithm, has keys similar to RSA keys but this cryptosystem was developed by the U.S. National Security Agency and distributed by the U.S. National Institute of Standards and Technology through the digital signature standard because of patent restrictions on the RSA cryptosystem [BAR01]. OpenPGP is the free version of PGP. PGP is the "pretty good privacy" cryptosystem developed by Phil Zimmerman [BAR01]. The default keys used for OpenSSH running under Version 2 of the SSH protocol are DSA keys.

RhostsRSA is similar to public-key authentication but it only provides host authentication, not user authentication. Each host generates a pair of RSA keys. The server then sends a numerical challenge to the client host and the client host must sign and send the message back to the server. This method differs from the previous method because the user does not have to specify the key or its passphrase.

Rhosts is a very insecure method of authentication; it involves creating a file that will allow a host listed in that file to establish a connection without any further authentication checks. This is the method used by the *rlogin*, *rsh*, *rcp* commands. In this method, when the server receives a connection, it checks the IP address and the hostname of the remote host against the "/etc/rhosts" file. If the IP address and hostname are located in the file, the host is authenticated and the user is granted access to the server assuming that a valid username was supplied. Rhosts is very insecure because it assumes that the client machine can protect itself from compromise. If the machine is compromised, then the server could be compromised as well, because this authentication method does not require the user to prove their identity with a password or private key.

The use of passwords has always been popular because users can remember passwords easily if they create them and there is no need to carry around a file

holding keys. The drawback to passwords is usually if people generate their own, then they are often easy to guess. In the case that the user is not allowed to generate their own password, then the user may not remember it as easily and will write it down, which could lead to the compromise of their account.

S/key is a form of one-time password challenge-response authentication. The server issues a challenge in the form of a string of characters and the user can either enter the string into a device that will provide the response that the user enters into the command line and sends to the server or the user carries a list of pre-calculated passwords and provides the appropriate response to the server from the list. Use of this authentication mechanism requires extra technology and devices.

Kerberos is an authentication method where the user authenticates to a server and receives a ticket that will grant access to other servers as long as these servers have been configured to use and receive the Kerberos tickets. This mechanism requires the installation and maintenance of a Kerberos server.

OpenSSH also provides compatibility with the Pluggable Authentication Modules (PAM) developed by Sun Microsystems. According to [BAR01], PAM "is an infrastructure for supporting multiple authentication methods." This allows for other authentication methods to be developed and used without modifying the OpenSSH source to directly implement the new method. All that OpenSSH has to provide is support for PAM.

d. Modes of Operation

OpenSSH can run in one of two modes: with privilege separation and without privilege separation. Privilege separation will be discussed in a later section, but first the need for it will be discussed.

When OpenSSH first appeared, it only provided secure remote login capabilities by encrypting the network traffic and providing numerous authentication methods. OpenSSH followed a standard client–server architecture: the server, known as a daemon, would listen for connections on a specific Internet Protocol (IP) address and port and when a connection is received it would spawn a child to handle the requests of the client. Every client–server application is different and each server may run as a specific user or a

special user without login capabilities. In a few special cases, the server needs to run as the root user because the server needs to execute some privileged commands. OpenSSH falls into this last category.

The OpenSSH daemon and its child need to possess privileges so that all client requests, such as password authentication, can be serviced. In a Linux or UNIX environment, user identities and passwords are stored in two different files: “passwd” and “shadow” in the “/etc” directory. The “passwd” file which stores information such as user name, user id, real name, home directory, and user path is normally readable by anyone on the system. However, it is only writeable by the root user. The “shadow” file contains the user id and password pairs for each user in an encrypted text format that is only readable by the root user. In order for the OpenSSH child to authenticate the user using password authentication, it would have to access both the “passwd” and “shadow” files. Thus, the OpenSSH child would have to run as the root user. There are some vulnerabilities [PRO03] that can result in privilege escalation to the root user. If root-user status can be acquired maliciously, then the system is compromised and the confidentiality, integrity and availability of the information on that system can no longer be guaranteed.

A solution to combat this threat was needed. Among the many proposed solutions that will mitigate the privilege escalation threat, one solution is the concept of privilege separation. Privilege separation is a generic concept with the objective of “[reducing] the amount of code that runs with special privilege without affecting or limiting the functionality of the service” [PRO03]. The implementation of privilege separation requires the use of two processes, a parent with privileges and a child without privileges. The child handles all user transactions and when a user transaction requires privileges, the child must ask the parent to process the transaction for the child [PRO03]. Privilege separation has the benefit of confining an intruder, if they manage to compromise the child, to the child’s address space and prevents the inheritance of privileges. The work done in [PRO03] proposed a framework for implementing privilege separation and OpenSSH was chosen as the test application that would demonstrate privilege separation. Privilege separation has now been fully integrated into OpenSSH and is a default option.

If privilege separation is disabled, then the OpenSSH daemon follows the standard client-server model, i.e., the daemon handles all user transactions.

B. PORTING BACKGROUND

In porting an application from one platform to another it is useful to know the differences between the original development platform and the target platform. This section will discuss how the features required by OpenSSH differ between OpenBSD and the XTS-400.

1. BSD Discussion

OpenSSH uses many features available on its development platform, OpenBSD. Many of these features are also available on many of the other platforms to which OpenSSH has been ported such as the various distributions of Linux. Table 1 provides a description of the features used by OpenSSH that do not have the same behavior on the XTS-400 as they do on OpenBSD.

Category	Name	Description
System Features	root user	a user with unlimited access to the system; all privileges are given to this user
	file-descriptor passing	ability for processes to pass file-descriptors to other processes
System Functions	chroot	create a new root directory for a process
	setgroups	sets the supplementary group IDs for a process
	initgroups	initializes the supplementary group list
	socketpair	creates a pair of UNIX domain sockets that are linked together
	setuid, seteuid, setreuid	sets the real and effective user ID of a process
	setgid, setgegid, setregid	sets the real and effective group ID of a process
	daemon	forks the process into the background and disconnects it from the controlling terminal
System Files	passwd	provides user information such as username, user ID, default group ID, home directory and shell
	shadow	provides the user's encrypted password
	utmp	contains a record of users logged in to the system
	wtmp	records all of the logins and logouts to the system
	group	provides information about the groups in the system and which users belong in each group
Environment	daemon environment	the init process provides an environment for all processes

Table 1. OpenBSD Features Used by OpenSSH

The next section will discuss how these features are different on the XTS-400.

2. XTS-400

The XTS-400 provides a Linux Binary Compatible Interface. This interface is not complete because not all features of Linux are supported through this interface. OpenSSH has been ported to numerous Linux distributions and is compatible with the

interface provided by the XTS-400. However, not all features required by OpenSSH as presented in Table 1 behave in the same way on the XTS-400 as they do in Linux or OpenBSD. The differences in the features are listed in Table 2.

Category	Name	XTS-400 Implementation Difference
System Features	the root user	there is no root user on the XTS-400
	file-descriptor passing	this feature is not implemented
System Functions	chroot	there is an API, but there is no underlying system support
	setgroups	there is an API, but there is no underlying system support
	initgroups	there is an API, but there is no underlying system support
	socketpair	there is an API, but there is no underlying system support
	setuid, seteuid, setreuid	these system calls require the privilege: set_owner_group
	setgid, setgegid, setregid	these system calls require the privilege: set_owner_group
	daemon	there is an API, but there is no underlying system support
System Files	passwd	stub file provided for Linux compatibility, not used for XTS-400 authentication
	shadow	does not exist on the system
	utmp	does not exist on the system
	wtmp	does not exist on the system
	group	stub file provided for Linux compatibility
Environment	daemon environment	no init process for daemons, daemons are started from the daemon database

Table 2. XTS-400 Implementation Differences

The features presented in Table 1 are not the only differences between the XTS-400 and other operating systems. The XTS-400 differs from most popular operating systems because unlike Linux, which only enforces discretionary access control policies, the XTS-400 also enforces mandatory access control policies. The XTS-400 enforces a

total of three policies: a mandatory secrecy policy, a mandatory integrity policy, and a discretionary policy.

The mandatory secrecy policy is represented by the Bell and La Padula secrecy formal model. This model prevents the unauthorized disclosure of sensitive information by maintaining two properties: the simple security property and the *-property [BEL76]. The simple security property prevents a subject from accessing an object if the secrecy level of the subject is dominated by the secrecy level of the object [BEL76] –this is referred to as read-up. The *-property only allows write-access if the secrecy level of the object is equal to or dominates the secrecy level of the subject [BEL76].

The mandatory integrity policy is represented by the Biba integrity formal model. This model prevents the unauthorized modification of information. Like the previous model, this model also has two properties that must be maintained: the simple integrity property and the *-property [BIB77]. The simple integrity property does not allow for a subject to have observe-access to an object if the integrity level of the subject dominates the integrity level of the object [BIB77] – this is referred to as read-down. The *-property allows for write-access if the integrity level of the subject is equal to or dominates the integrity level of the object [BIB77].

The discretionary access control policies are enforced by two mechanisms: access control lists (ACLs) and capability lists – referred to as subtypes in the context of the XTS-400. For a detailed discussion of these two types of access control mechanisms, refer to [LAM74].

The differences between the development and target platforms provided the greatest challenges in porting OpenSSH to the XTS-400. Software dependencies created a few minor challenges for the port. OpenSSH relies on other software packages and libraries in order to function. These extra packages were not present on the XTS-400. A discussion of these packages follows.

C. SOFTWARE DEPENDENCIES

According to [BAR01] and [SSH04], OpenSSH requires the following software in order for it to run: Zlib 1.1.4 or greater and OpenSSL 0.9.6 or greater. Further inspection revealed that OpenSSL depends on a random number generator that is normally available

in Linux through the "/dev/random" or "/dev/urandom" device entries. The XTS-400 does not provide a kernelized random number generator; the above mentioned devices are not listed anywhere in the file system. Both the OpenSSL and OpenSSH installation instructions recommended the use of either the PRNGd (pseudo random number generator daemon) or the EGD (entropy gathering daemon). EGD was chosen for use because the PRNGd documentation referenced the EGD. Installation instructions for these three software packages are in Appendix A of this report. A discussion of these three packages will follow.

1. Zlib

Zlib 1.1.4 is a compression library with utilities. OpenSSH and OpenSSL use zlib in order to compress data when communicating over a network. The compression occurs before encryption so the delay in transmission due to encryption is reduced [BAR01]. The use of compression is also a daemon runtime configuration option as can be seen in Appendix C: SSH daemon configuration file. However, this file is not used until the SSH daemon executes. Simply changing the option "Compression Yes" to "Compression No" will not stop the tests by the configuration file to look for the zlib libraries and header files. In order to bypass the configuration file tests, either false libraries and header files would have to be created or the configuration file would have to be modified to not check for zlib. After a quick analysis of the zlib documentation, it was determined that the installation of zlib would not be difficult. In this case, it was easier to install the required software than to provide false libraries or modify the configuration file.

2. OpenSSL

OpenSSL is a project designed to provide "a robust, commercial-grade, full-featured, and Open Source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols as well as a full-strength general purpose cryptography library" [SSL04]. The OpenSSL libraries and header files provide the cryptographic ciphers needed by OpenSSH to perform encryption. Without the use of the libraries, then OpenSSH would be reduced to the ordinary telnet and file transfer protocols. A quick glance at the XTS-400 "/lib" directory showed that the OpenSSL shared libraries were already installed; however, the OpenSSH configuration file did not accept the libraries. Unlike zlib, this application is absolutely necessary in order for

OpenSSH to work, so simple file substitution was not an option and neither was modifying the configuration file. A quick review of the OpenSSL documentation revealed that the installation, like zlib, should not be difficult. The latest source,openssl-0.9.7d, was downloaded and installed.

3. Entropy Gathering Daemon

The entropy gathering daemon is needed to provide a software-based pseudo-random number generator. It was quickly discovered that without some type of random number generator, OpenSSH will not run; the application will exit with the error, "prng not seeded." A quick alternative was needed. As mentioned, both the OpenSSH and OpenSSL documentation mentioned two software-based random number generators: the pseudo-random number generator daemon (PRNGd) and the entropy gathering daemon (EGD). Both daemons stated compatibility with OpenSSL and OpenSSH. The PRNGd documentation also mentioned that the PRNGd also provided an interface to the EGD. This implies that the EGD provides greater compatibility with more systems. A test of each random number generator was not conducted and is beyond the scope of this report. The EGD is a perl script that monitors processes and provides random data based on information gathered from the processes. This data is then stirred each time a request for random data is made. It should be mentioned that the mandatory security policies prevent the EGD from viewing all possible processes resulting in a smaller pool of processes from which to gather random data. The EGD can only view processes running at the same secrecy and integrity levels as itself. For each OpenSSH daemon running, there must be an EGD running at the same level. This is very important because if the random number generator is exhausted, then OpenSSH will stop functioning either by waiting for random data to become available or by the daemon refusing to start. Both would result in an inability to function. It is suggested that an alternative to the EGD be found or developed.

4. MYSEA Libraries

As shown in Table 2, some of the system calls require privileges. The MYSEA libraries provide APIs for acquiring and revoking privileges. The MYSEA libraries must be installed so that OpenSSH can operate properly.

III. INTEGRATION OF OPENSSH ONTO THE XTS-400

This chapter will provide the goals of this project, the methodology used to accomplish those goals, the results of the research, and solutions to the problems encountered.

A. GOALS

The goals of this project are to port OpenSSH to the XTS-400 with as much of the functionality preserved as possible. The key functions desired are the interactive session and the use of PKI for authentication. An extra benefit of this work is the provision of a better understanding of the functions supported by the XTS-400. This may inform future porting projects on the XTS-400.

B. METHODOLOGY

To port a particular application to the XTS-400, a number of steps must be completed. One step is to look at the documentation of the application that will be ported. Here, it is important to identify any references to other software packages that the application requires. If these additional software packages are not available on the XTS-400, then they must be ported first. Another item to look for in the documentation is the process architecture of the application. When the process architecture is not available, a thorough examination of the source code will help to determine the process architecture. The process architecture is important because the XTS-400 enforces mandatory security policies and will not allow processes at different sensitivity levelsto communicate with one another unless privileges are given to both processes. Another step is to review the documentation for the XTS-400 and look for any system calls not supported by the operating system that are required by the application. Each of these items: software dependencies, the process architecture and system compatibility, can either produce a delay in the port or a complete roadblock.

For this research the above steps were followed but the process was not sequential, but cyclic. First, the OpenSSH documentation was reviewed, then the source code, then the XTS-400 documentation. When a challenge, usually in the form of an unsupported system call, was discovered, all three sources of information were consulted.

A helpful tool used to review the source code deserves mention: the Linux Cross Reference (LXR). This tool allows for the browsing of source code through a web-browser. Like a compiler, the LXR will generate a table of symbols and those symbols will be treated as links in the source code. This feature allows for the quick lookup of variable and function declarations, definitions and references. The source code for this tool can be downloaded from http://sourceforge.net/projects/lxr.

The documentation for the XTS-400 was a little vague or incomplete at times. In order to verify that a normal feature on Linux or UNIX was not supported on the XTS-400, the XTS-40 manuals were consulted and then test programs were used. Theses test programs were written for a Fedora Core 1 Linux platform and then tested on that platform. The test programs were then transferred to the XTS-400 for testing and the results from each platform were compared. This was done to verify that support for the given feature was not implemented if proper documentation could not be found so that a blind assumption would not be made.

C. PORTING RESULTS

OpenSSH was successfully ported to the XTS-400, with some limited functionality. Privilege separation could not be implemented because file-descriptor passing is not supported by the XTS-400. To simplify the port, the only authentication method available is public-key authentication. Each single level LAN requires an OpenSSH daemon and there is no communication between any of the daemons as a consequence of the mandatory access control policies.

D. CHALLENGES ENCOUNTERED

The challenges encountered are the features listed in Tables 1 and 2. A discussion of the solutions used to circumvent these limitations is presented in the order they are presented in Tables 1 and 2. All source code modifications can be found in Appendix B: Source Code Listing. All modified files are in the “openssh-3.7.1p2” source code directory.

1. System Features

The two system features listed in Table 1 are the root user and file descriptor passing. As mentioned in Table 2, the root user does not exist. The XTS-400 does not give a single user all system privileges; rather the integrity policy is used to mediate

access to privileged operations. In an OpenBSD system, the OpenSSH daemon runs as the root user. The OpenSSH code performs user ID checks to verify that the daemon is running as the root user. If these checks fail, then the daemon assumes that it is not running with root user privileges and will not function properly. In the XTS-400, the OpenSSH daemon must run as the network user so that port 22 can be opened. Port 22 is the default port for the SSH protocol. To account for the non-existence of the root user, the user ID checks were altered to check for the user ID of the network user. The modifications were made to the *temporarily_use_uid* and the *permanently_set_uid* functions in the "uidswap.c" file and the *do_setusercontext* function in the "session.c" file.

File descriptor passing is required for SSH privilege separation. The system feature of file descriptor passing through UNIX domain sockets is not supported by the XTS-400 [DIG03a]. The only solution would be to build support for this feature into the operating system. The operating system for the XTS-400 is called the Secure Trusted Operating Program (STOP). STOP is proprietary and cannot be modified without invalidating the evaluated assurance level assigned by the National Institute of Standards and Technology and the National Security Agency. Because there is no feasible solution for this problem, privilege separation has been disabled.

These problems were the only ones encountered for this category of challenges. The next category to be discussed is the system functions provided by the XTS-400.

2. System Functions

There are many system functions that are used by OpenSSH. The untrusted environment of the XTS-400 strongly resembles Redhat Linux 8.0. This strong resemblanceis only superficial. In fact, many of these functions were not implemented on the XTS-400 and were only APIs with functional stubs.

The *chroot* system call was identified by the configuration file as being available on the XTS-400; however, it would exit with the error "Function not implemented." In the XTS-400 documentation, *chroot* is listed as an unsupported system call [DIG03c]. Support for this system call must be built into the operating system, but this is not an option. The solution used involved commenting out all references to the *chroot* system

call in the source code. The modifications occurred in the *privsep_preauth_child* function in the "sshd.c" file. Privilege separation is not implemented and this function should never be called, but it has remained commented out to ensure that the daemon does not exit prematurely.

The *setgroups* function is also not implemented on the XTS-400 as mentioned in the User's Manual for the XTS-400 [DIG03c]. All references to this system call were commented out. The modifications are in the *main* and *privsep_preauth_child* functions of the "sshd.c" file and the *temporarily_use_uid* and *restore_uid* functions in the "uidswap.c" file.

The *initgroups* system call is not supported either. All references to this system call have been commented out. The modifications are in the *temporarily_use_uid* function in the "uidswap.c" file and the *do_setusercontext* function in the "session.c" file.

The *socketpair* system call is not implemented. This call returns with the error "invalid argument". The XTS-400 documentation states that unsupported interprocess communication (IPC) mechanisms will return the error "Invalid Argument" [DIG03c]. OpenSSH provides compatibility for the use of UNIX domain socket pairs or pairs of pipes for IPC. A modification had to be made to force OpenSSH to use pairs of pipes instead of socket pairs. The modification was to uncomment the line, *#define USE_PIPES 1*, in the "defines.h" file. *Socketpair* is used extensively in the monitor code that is used when privilege separation is enabled. In an attempt to support privilege separation, pairs of pipes were created to replace the socket pair. Privilege separation appeared to work through the pre-authentication phase of a user login, but because file descriptor passing is not supported, no tests could be conducted to verify that this phase of privilege separation did work as intended. All modifications made to the monitor-specific files have been commented out. The files that were modified are "monitor.c", "monitor_wrap.c" and "monitor.h".

There are many functions that can be used to set the real and effective user and group IDs of a process. These functions are listed in Table 1 and Table 2. These system calls are supported by the XTS-400, but require special XTS-400 privileges to operate properly. The specific privilege that they require is *set_owner_group* on the XTS-400

[DIG03b]. Privileges can be granted to programs by installing the program with *tp_edit*. To follow the principle of least privilege, the privileges should be granted when necessary and then be revoked when not needed. The MYSEA libraries provide APIs to request and revoke the *set_owner_group* privilege. The modifications made to the source code were to the *temporarily_use_uid*, *restore_uid* and *permanently_set_uid* functions in the "uidswap.c" file. The header files for the MYSEA libraries had to be included in the "uidswap.c" file in order for the C compiler to locate and link the appropriate functions when the executable is being constructed. The OpenSSH configuration script had to be given extra arguments to specify the location of the libraries and which libraries to use. The options given to the configuration script can be seen in Appendix A.

The *daemon* system call is not supported in the XTS-400 untrusted environment. The OpenSSH source code provides a directory called "openbsd-compat" under the "openssh-3.7.1p2" source directory that provides certain functions that may not be implemented on the target platform. *Daemon* is one of the functions provided in the "openbsd-compat" directory. However, the OpenSSH configuration file identified the *daemon* API as being supported by the XTS-400, but this is incorrect because the *daemon* API is only a functional stub. The linking order when the executable is being built will not permit the daemon function in "openbsd-compat" to be used instead of the *daemon* system call provided by the XTS-400. The solution for this was to copy the daemon code from the "openbsd-compat/daemon.c" file into the "sshd.c" file. A new function prototype was added to the "sshd.c" file and the function definition was added to the end of the "sshd.c" file. The new daemon function is called *daemonize* to prevent the linker from calling the *daemon* system call provided by the XTS-400.

These are the solutions for the system calls that were either not supported or that required privileges. The next set of challenges involved system files that OpenSSH expects to be present on the system.

3. System Files

There are four system files that OpenSSH assumes to be present on the system: "/etc/passwd", "/etc/shadow", "/var/run/utmp", "/var/log/wtmp" and "etc/group". The passwd file is present on the XTS-400 system while the remaining three files are not.

The “/etc/passwd” file is used by the untrusted environment to assign each user a user ID, a home directory and a default shell. The “/etc/passwd” file is generated by the *xtsmkpasswd* command. This command does not produce a correct “passwd” file because the default group listed in the file assigns all users to the system group. The system group is reserved for the STOP kernel and no user should ever have this group assigned as their default group. OpenSSH uses the “/etc/passwd” file to associate the supplied username with the appropriate user ID and the default group ID of the user when the user attempts to login. The user’s default group is assigned by the system administrator when the user’s account is created. The “/etc/passwd” file had to be modified to reflect the proper default group association for the users of the system. The modifications made were to the fourth field of every line in the “/etc/passwd” file. In order to make the mdofications, the trusted command, *ua_edit*, was used to lookup the default group for every user in XTS-400.

The “/etc/shadow” file is not present on the XTS-400. The functionality supported by the “/etc/shadow” file is replaced by two trusted databases on the XTS-400: the “user access authorization” database and the “user access information” database. The “user access authorization” database contains the following information: a password history list, a change password flag, a default group identifier, maximum mandatory session levels, default mandatory session levels, last login time, number of failed login attempts, time of last password change, and a list of user capabilities. The “user access information” database contains: the username, the user’s home directory and a default shell. This last database is similar to the “/etc/passwd” file mentioned earlier. The “user access authorization” database is stored at the highest secrecy and integrity levels and is protected by the subtype DAC mechanism. Without the “/etc/shadow” file, password authentication will not succeed. For password authentication to work, the OpenSSH daemon would have to be granted access to the user databases and, in order to accomplish this, more privileges would have to be granted to the OpenSSH daemon. The specific privileges are *simple_security_exempt* and *subtype_exempt*. It was determined that further and more detailed analysis would be needed to determine where to insert the privileged code. There was no solution for fixing the absence of the “/etc/shadow” file.

The workaround was to disable password authentication so that the OpenSSH daemon would not attempt to access a nonexistent file.

The next two files are related and are discussed together. The "/var/log/wtmp" file contains a record of all user logins and logouts. This file is not present on the XTS-400 as its functionality is provided by the trusted "user access authorization" database discussed earlier. The "/var/run/utmp" file contains a record of all users currently logged in to the system. This file is not present either. These files are used for account login auditing in a Linux system. Because these files are not present and the functionality of only one is provided through a trusted database that would require privileges to access it, the solution was to disable support for these files. This was accomplished by giving the OpenSSH configuration script a few extra options that disabled support for these files. The options given to the configuration script can be seen in Appendix A.

The last system file presented is the "/etc/group" file. This file stores information on all groups including a list of users associated with each group. This file would normally be used by the *setgroups* and *initgroups* system calls, but as mentioned, these calls are not supported. The "/etc/groups" file is generated by the *xtsmkgroup* command. This command does not produce a correct "/etc/group" file. After examining the file, no users were listed with their associated groups. In the event that *setgroups* and *initgroups* are implemented, this file will have to be modified manually to contain the correct information that these system calls will require. The functionality of the "/etc/group" file is provided by the two trusted group databases: the "group access authorization" database and the "group access information" database. These two databases are stored at the highest secrecy level and protected by the subtype DAC mechanism. To support the proper association of users to groups, access to these databases would have to be granted through privileges. The analysis required to identify where the privileged code should be added was beyond the scope of this thesis.

These were the only problems encountered regarding protected system files on the XTS-400 required by OpenSSH. The next problem presented is the daemon startup environment.

4. Environment

In an OpenBSD system, daemons are started by the *init* process and this process creates all other processes in the system [STE93]. When a daemon is started either by the init process or through a shell, an environment is created that includes the allocation of three file streams: STDIN, STDOUT and STDERR. These three streams are associated with the following file descriptors: 0, 1 and 2 respectively. OpenSSH expects these three streams and corresponding file descriptors. On the XTS-400, daemon processes are started from the *start_daemon* command. This command does not allocate the three file streams expected by OpenSSH so when OpenSSH starts to allocate files, the file descriptor numbering starts at 0. When a user logs in through the OpenSSH daemon, a pseudo-terminal is requested and it is referenced by a file descriptor. If the file descriptor used to reference the pseudo-terminal is one of the three reserved file descriptors that OpenSSH assumes to be provided, then all I/O will be sent to the pseudo-terminal across the network connection. During the authentication session setup sequence of a user login, specifically structured messages are sent between the client and the server. If all of the pseudo-terminal I/O is sent to the client, then the client gets confused and exits. In the *main* function of the "sshd.c" file, the developers of OpenSSH placed comments stating that the file descriptors 0, 1 and 2 should be reserved and never closed. To solve this problem, the "/dev/null" file was opened three times so that the first three entries in the file descriptor table would be in use. This will cause the file descriptor numbering to begin after 2. The modifications made were in the *main* function of the "sshd.c" file.

IV. INTEGRATION TESTING

After porting an application to a different platform, testing is required to ensure that the application still functions as specified. The functionality of the ported application may have been altered by the modifications to the source code. In order to detect any difference in functionality, comprehensive testing is required. Developmental testing tests the functionality of the ported application in a simple environment. To ensure that the ported application can function in a more realistic environment, a larger and more realistic testing environment is needed. The testing in this environment is called Testbed testing.

Due to time constraints, only the developmental testing was performed for the OpenSSH port to the XTS-400. Both testing methods will be discussed in this chapter; however only the results for the developmental testing will be provided.

A. DEVELOPMENTAL TESTING

1. Test Plan

The developmental test plan describes the results used to validate that the port of OpenSSH to the XTS-400 is successful. A small network was created with three laptops, one is a Fedora Core 1 Linux system and the other two are Windows XP systems. The three clients are connected to the XTS-400 machine, "Holmes," though a switch. Holmes has a network interface card that has four interfaces. One interface is used to simulate the MLS-LAN with one client and one TPE. The other three interfaces are used to simulate the multiple single level LANs. The IP addresses of the three single level clients were to be configured to allow those clients to communicate with the corresponding network as required by the specific tests. A diagram of the test network is presented in Figure 1.

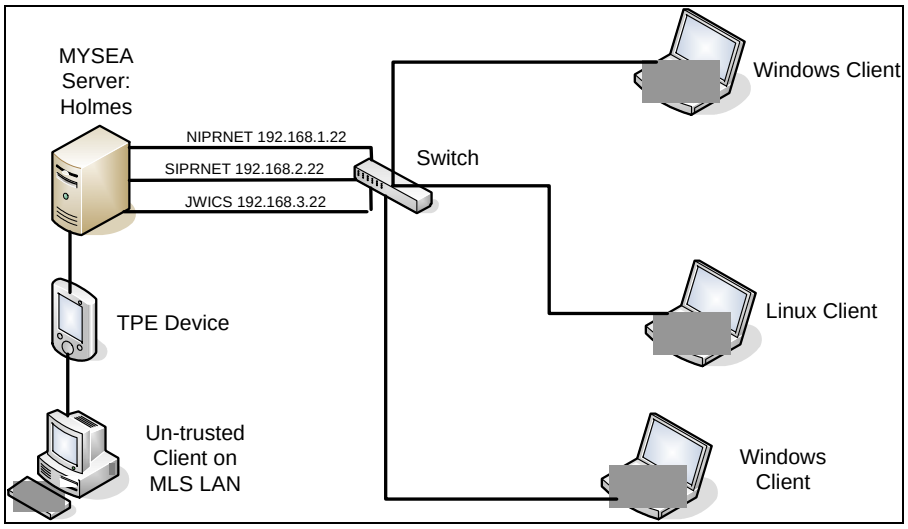

Figure 1. Developmental Testing Network Topology

a. MAC Policy Enforcement

The purpose of this test suite is to verify that the mandatory policies enforced by the XTS-400 are still enforced when a user is logged in through OpenSSH. The test plan is presented in Table 3. The Test Type identifies the kind of test being performed. The Session Level identifies the secrecy and integrity levels of the remote user. The Object Level identifies the secrecy and integrity levels of the object. The command used to test for read-access is *more*. This command only requires and uses read access; modification of the file is not required. The command used for writing is *vi* – the visual editor. *Vi* is a common editor that is available on most systems. *Vi* needs to be able to both modify and save the modifications. The integrity levels of OSS and Admin are defined as il3{all integrity compartments} and il7{all integrity compartments} respectively. In this suite of tests, a result of "pass" means that the operation was allowed, e.g., *more* was able to display the file, and *vi* was able to edit the file and save the changes. A result of "fail" means that the operation was not permitted. *Vi* can appear to make modifications, but if the modifications cannot be saved, then the write-access test fails.

The objects are text files with the following naming convention: test_sl#il#.txt where the two pound signs are replaced with the appropriate object level numbers, e.g., *test_sl1il3.txt* if the object level is sl1:il3. The permissions on the objects are read, write, and execute for owner, group and world. This ensures that the

discretionary policies do not interfere with the tests. Because the XTS-400 does not allow subjects to write to objects if the level of the parent directory is dominated by the level of the subject or object, directories had to be created to hold the objects at the specific sensitivity levels in order to allow tests for writing. For example, for objects with a level of sl1:il3, a directory with the same level had to be created to hold the objects.

Test Number	Test Type	Session Level	Object Level	Command	Expected Result
a1	Secrecy read -up	sl1:il3	sl2:il3	more	Fail
a2			sl3:il3	more	Fail
a3			sl4:il3	more	Fail
a4		sl2:il3	sl3:il3	more	Fail
a5			sl4:il3	more	Fail
a6		sl3:il3	sl4:il3	more	Fail
a7	Secrecy read-down	sl1:il3	sl0:il3	more	Pass
a8		sl2:il3	sl0:il3	more	Pass
a9			sl1:il3	more	Pass
a10		sl3:il3	sl0:il3	more	Pass
a11			sl1:il3	more	Pass
a12			sl2:il3	more	Pass
a13	Secrecy read-equal	sl1:il3	sl1:il3	more	Pass
a14		sl2:il3	sl2:il3	more	Pass
a15		sl3:il3	sl3:il3	more	Pass
a16	Secrecy write-up	sl1:il3	sl2:il3	vi	Fail
a17			sl3:il3	vi	Fail
a18			sl4:il3	vi	Fail
a19		sl2:il3	sl3:il3	vi	Fail
a20			sl4:il3	vi	Fail
a21		sl3:il3	sl4:il3	vi	Fail
a22	Secrecy write-down	sl1:il3	sl0:il3	vi	Fail
a23		sl2:il3	sl0:il3	vi	Fail
a24			sl1:il3	vi	Fail
a25		sl3:il3	sl0:il3	vi	Fail
a26			sl1:il3	vi	Fail
a27			sl2:il3	vi	Fail
a28	Secrecy write-equal	sl1:il3	sl1:il3	vi	Pass
a29		sl2:il3	sl2:il3	vi	Pass
a30		sl3:il3	sl3:il3	vi	Pass
a31	Integrity read-up	sl1:il3	sl1:OSS	more	Pass
a32	Integrity read-down	sl1:il3	sl1:il0	more	Fail
a33			sl1:il1	more	Fail
a34			sl1:il2	more	Fail
a35	Integrity write-up	sl1:il3	sl1:OSS	vi	Fail
a36	Integrity write-down	sl1:il3	sl1:il0	vi	Fail
a37			sl1:il1	vi	Fail
a38			sl1:il2	vi	Fail

Table 3. MAC Policy Enforcement Test

As mentioned, this suite of tests verifies that the MAC policies are enforced. Each MAC policy has two properties that must be maintained: a simple security or integrity property and a *-property. Table 4 presents definitions of the types of test used in this suite of tests. In this table, the Policy Type identifies whether the definition is for the mandatory secrecy or integrity policy. The Property column identifies which property of the mandatory policies is being tested. The Access Type specifies the type of access. The Definition gives the mathematical definition of the property where *S* represents the subject, *O* represents the object, *sl()* means the secrecy level, *il()* means the integrity level, '=' means equality, '>' means dominates, and '<' means is dominated by. The last column in the table states whether or not the action is allowed given the Property of the Policy type. The models that represent the security policies allow for secrecy write-up and integrity write-down. However, the XTS-400 does not allow these access types so the expected results for the XTS-400 should be no. The effected entries in the table are identified by an asterisk.

Policy Type	Property	Access Type	Definition	Allowed
Secrecy	simple security	Read-equal	sl(S) = sl(O)	Yes
		Read-down	sl(S) > sl(O)	Yes
		Read-up	sl(S) < sl(O)	No
	*-property	Write-equal	sl(S) = sl(O)	Yes
		Write-down	sl(S) > sl(O)	No
		Write-up	sl(S) < sl(O)	Yes*
Integrity	simple integrity	Read-equal	il(S) = il(O)	Yes
		Read-down	il(S) > il(O)	No
		Read-up	il(S) < il(O)	Yes
	*-property	Write-equal	il(S) = il(O)	Yes
		Write-down	il(S) > il(O)	Yes*
		Write-up	il(S) < il(O)	No

Table 4. MAC Policy Test Definitions

In the mandatory secrecy policy, the simple security property does not allow a subject to access an object if the secrecy level of the subject does not dominate the secrecy level of the object. Therefore, all secrecy read-up tests should fail. All

secrecy read-down and read-equal tests should pass. The *-property of the secrecy policy does not allow a subject to modify an object if the secrecy level of the subject dominates the secrecy level of the object. All secrecy write-down tests should fail. Theoretically, it is possible for a subject to modify an object if the secrecy level of the subject is dominated by the secrecy level of the object; however, the XTS-400 does not allow this action, so all secrecy write-up tests should fail as well. Only the secrecy write-equal tests should pass.

The simple integrity property of the mandatory integrity policy does not allow a subject to read an object if the integrity level of the subject dominates the integrity level of the object. All integrity read-down tests should fail and all integrity read-up and read-equal tests should pass. The *-property of the mandatory integrity policy does not allow a subject to modify an object if the integrity level of the subject is dominated by the integrity level of the object. All integrity write-up tests should fail. The formal model for the mandatory integrity policy allows for the theoretical write-down from a subject to an object, but this is not allowed on the XTS-400 and all integrity write-down tests should fail. Only the integrity write-equal tests should pass.

b. DAC Policy Enforcement

The purpose of this test suite is to verify that the discretionary access control policies enforced by the XTS-400 are still enforced when logged in through OpenSSH. The test plan is presented in Table 5. The username and group name used to run this test should remain constant, i.e., the same user login, “cherbig” with a default group of “other”, is used to test all cases. The object permissions identify the permissions of the object that can be seen when the *ls –l* command is issued. The object name identifies how the object was named to help keep track of the permissions. The naming convention is discussed later. The object owner identifies the username of the owner of the object and the object group identifies the group name of the owning group. The action identifies the type of access tested. For read-access, the command, *more*, will be used to attempt to read the file. The *vi* tool will be used to test for write-access. To test for execute-access, the file name will be entered at the shell prompt, i.e., the file will have to be a program.

The objects for read- and write-access tests are text files. The objects used for execute-access are simple c-programs that print a message to the screen. The naming convention for the text files are “test_<o or g or a><r or w or x>.txt”, where *o* means owner, *g* means group and *a* means all and *r* means read, *w* means write and *x* means execute. The “or” specified in the filename is not exclusive. For example, the file “test_orwx.txt” means that the object permissions are read, write and execute for the owner. The programs used for testing execute-access will have the same file name without the “.txt” extension. The session and object levels will remain fixed in order to prevent the MAC protection mechanisms from interfering with the tests.

Test Number	Object Permissions/ Name	Object Owner	Object Group	Action	Expected Results
b1	rwxrwxrwx test_ogarwx.txt	cherbig	other	read	Allowed
b2				write	Allowed
b3				execute	Allowed
b4	rwxrwx--- test_ogrwx.txt	cherbig	other	read	Allowed
b5				write	Allowed
b6				execute	Allowed
b7	rwx------ test_orwx.txt	cherbig	other	read	Allowed
b8				write	Allowed
b9				execute	Allowed
b10	rw------- test_orw.txt	cherbig	other	read	Allowed
b11				write	Allowed
b12				execute	Fail
b13	r-------- test_or.txt	cherbig	other	read	Allowed
b14				write	Fail
b15				execute	Fail
b16	--------- test_none.txt	cherbig	other	read	Fail
b17				write	Fail
b18				execute	Fail
b19	rwxrwxrwx dtest_ogarwx.txt	demo	other	read	Allowed
b20				write	Allowed
b21				execute	Allowed
b22	rwxrwx--- dtest_ogrwx.txt	demo	other	read	Allowed
b23				write	Allowed
b24				execute	Allowed
b25	rwxrw---- dtest_orwxgrw.txt	demo	other	read	Allowed
b26				write	Allowed
b27				execute	Fail
b28	rwxr----- dtest_orwxgr.txt	demo	other	read	Allowed
b29				write	Fail
b30				execute	Fail
b31	rwx------ dtest_orwx.txt	demo	other	read	Fail
b32				write	Fail
b33				execute	Fail
b34	---rwx--- atest_grwx.txt	demo	stop	read	Fail
b35				write	Fail
b36				execute	Fail

Table 5. DAC Policy Enforcement Test

c. TPE Testing with Files Created by OpenSSH

The purpose of these tests is to verify that when users login from the MLS-LAN through the TPEs, they can view the files created under OpenSSH. The test

plan is presented in Table 6. The TPE Login Level specifies the session level of the current test. The TPE allows users to negotiate and renegotiate their session level. This test will also test for MAC policy enforcement through the TPE. The Object Levels are the secrecy and integrity levels of the object. A result of "yes" means that the file was viewable through the web-browser and a result of "no" means that the file could not be displayed.

This test suite uses the MYSEA web-server's ability to navigate and display pages from the user's home directory. In order to do this, a special directory, "public_html", had to be created under the user's home directory. This directory must have the same levels as the user's home directory and the permissions must be read, write and execute for the owner, and read and execute for the group and world. Within the "public_html" directory, there must be three more directories, one for each network classification. The levels of these three directories must be set appropriately using *fsm*. This will allow OpenSSH to create and modify files as long as the files are within these directories. A diagram of the proposed file system structure is presented in Figure 2. In the diagram, the levels of the required directories are also listed.

The objects used in this test are text files which can be viewed through a web-browser.

Test Number	TPE Login Level	Object Levels	TPE Viewable Expected Results
c1	sl1:il3	sl1:il3	Yes
c2		sl2:il3	No
c3		sl3:il3	No
c4	sl2:il3	sl1:il3	Yes
c5		sl2:il3	Yes
c6		sl3:il3	No
c7	sl3:il3	sl1:il3	Yes
c8		sl2:il3	Yes
c9		sl3:il3	Yes

Table 6. TPE Viewing Capability Test for OpenSSH Created Files

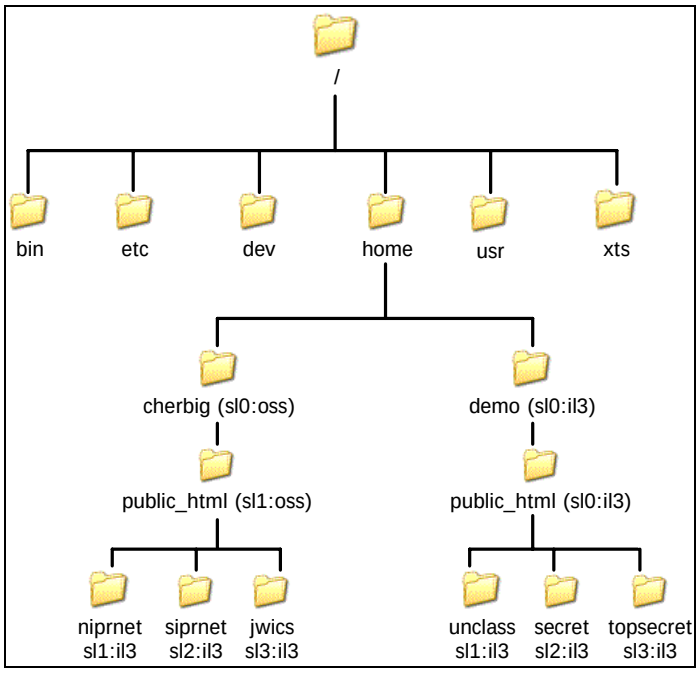

Figure 2. File System Structure for TPE Testing

d. TPE Testing with Files Modified by OpenSSH

The purpose of this test suite is to verify that changes made to pre-existing files through OpenSSH are viewable from the MLS-LAN through the TPEs. This test plan is very similar to the previous test. The test plan is presented in Table 7

Test objects were created at the console of the XTS-400 prior to the execution of the tests. The objects are within the three network-classified directories under the "public_html" subdirectory. This allows for a user logged in through OpenSSH to modify the files.

Test Number	TPE Login Level	Object Levels	TPE Viewable Expected Results
d1	sl1:il3	sl1:il3	Yes
d2		sl2:il3	No
d3		sl3:il3	No
d4	sl2:il3	sl1:il3	Yes
d5		sl2:il3	Yes
d6		sl3:il3	No
d7	sl3:il3	sl1:il3	Yes
d8		sl2:il3	Yes
d9		sl3:il3	Yes

Table 7. TPE Viewing Capability Test for Files Modified by OpenSSH

e. Single Level LAN – Simultaneous User Logins

The purpose of this test suite is to verify that multiple users on the same single level LAN can login. Success is determined by the user being presented with a shell and the command, *whoami*, returns the user's correct username. The test plan is presented in Table 8.

Test Number	LAN Level	User Name	Successful Logins
			Expected Results
e1	sl1:il3	cherbig	Pass
		demo	Pass
		testuser	Pass

Table 8. Single Level LAN – Simultaneous User Logins

f. Multiple Single Level LANs – Simultaneous User Logins

The purpose of this test suite is to verify that users can login from multiple networks of varying classifications. The test plan is presented in Table 9. Each user, specified by the username logs in at each network. Success is determined by the user being presented with a shell and the command, *whoami*, returns the correct username and the *level* command returns the correct session level.

Test Number	LAN Level	Username	Successful Login
			Expected Result
f1	sl1:il3	cherbig	Yes
	sl2:il3	demo	Yes
	sl3:il3	testuser	Yes

Table 9. Multiple Single Level LANs – Simultaneous User Logins

g. Public Key Authentication Tests

The purpose of these tests is to verify that the public key authentication mechanism works properly. The test plan is presented in Table 10. In this test suite, a valid username means that the user does exist within the system. An invalid user means that a username was supplied, but that user does not exist on the system. A correct private key means that the private key has a corresponding public key belonging to the user attempting to login. A wrong private key means that the private key used does not match the public key presented by the user.

Test Number	User Name	Private Key	Passphrase	Expected Results
g1	Valid	Correct	Correct	Succeed
g2	Valid	Correct	Wrong	Fail
g3	Valid	Wrong	Correct	Fail
g4	Invalid	Correct	Correct	Fail

Table 10. Public Key Authentication Test

h. Miscellaneous Tests

To test how OpenSSH reacts when the user's account is created incorrectly, a set of experiments was created. The test plan is presented in Table 11. The first case tests to see how OpenSSH reacts when a user's home directory is not present, i.e., it was never created. The second test, determines what happens when either the user or the Administrator revoke all permissions on the user's home directory. The third test determines what happens when a user logs in from a network with a lower security classification than that of his home directory.

Test Number	Case	Environment	Expected Results
h1	Home directory not present	Home directory not present	Cannot login, no key file
h2	permissions on home directory incorrect	--------- $HOME	Cannot login, cannot read key file
h3	level on home directory incorrect	level($HOME) = sl2:il3, session level = sl1:il3	Cannot login, cannot read key file

Table 11. Miscellaneous Tests

2. Test Validation Report

This section provides the results of the above tests as conducted on the developmental system for the port. The tables from the test plan have been replicated and a column with the test results has been added to each.

a. MAC Policy Enforcement Test Validation Results

For this suite of tests, the test results matched the expected results and it can be concluded that the MAC policies remain enforced when users are logged in through OpenSSH.

Test Number	Test Type	Session Level	Object Level	Command	Expected Result	Actual Result
a1	Secrecy read-up	sl1:il3	sl2:il3	more	Fail	Fail
a2			sl3:il3	more	Fail	Fail
a3			sl4:il3	more	Fail	Fail
a4		sl2:il3	sl3:il3	more	Fail	Fail
a5			sl4:il3	more	Fail	Fail
a6		sl3:il3	sl4:il3	more	Fail	Fail
a7	Secrecy read-down	sl1:il3	sl0:il3	more	Pass	Pass
a8		sl2:il3	sl0:il3	more	Pass	Pass
a9			sl1:il3	more	Pass	Pass
a10		sl3:il3	sl0:il3	more	Pass	Pass
a11			sl1:il3	more	Pass	Pass
a12			sl2:il3	more	Pass	Pass
a13	Secrecy read – equal	sl1:il3	sl1:il3	more	Pass	Pass
a14		sl2:il3	sl2:il3	more	Pass	Pass
a15		sl3:il3	sl3:il3	more	Pass	Pass
a16	Secrecy write-up	sl1:il3	sl2:il3	vi	Fail	Fail
a17			sl3:il3	vi	Fail	Fail
a18			sl4:il3	vi	Fail	Fail
a19		sl2:il3	sl3:il3	vi	Fail	Fail
a20			sl4:il3	vi	Fail	Fail
a21		sl3:il3	sl4:il3	vi	Fail	Fail
a22	Secrecy write-down	sl1:il3	sl0:il3	vi	Fail	Fail
a23		sl2:il3	sl0:il3	vi	Fail	Fail
a24			sl1:il3	vi	Fail	Fail
a25		sl3:il3	sl0:il3	vi	Fail	Fail
a26			sl1:il3	vi	Fail	Fail
a27			sl2:il3	vi	Fail	Fail
a28	Secrecy write-equal	sl1:il3	sl1:il3	vi	Pass	Fail
a29		sl2:il3	sl2:il3	vi	Pass	Fail
a30		sl3:il3	sl3:il3	vi	Pass	Fail
a31	Integrity read-up	sl1:il3	sl1:OSS	more	Pass	Pass
a32	Integrity read-down	sl1:il3	sl1:il0	more	Fail	Fail
a33			sl1:il1	more	Fail	Fail
a34			sl1:il2	more	Fail	Fail
a35	Integrity write-up	sl1:il3	sl1:OSS	vi	Fail	Fail
a36	Integrity write-down	sl1:il3	sl1:il0	vi	Fail	Fail
a37			sl1:il1	vi	Fail	Fail
a38			sl1:il2	vi	Fail	Fail

Table 12. MAC Policy Enforcement Test Validation Results

b. DAC Policy Enforcement Test Validation Results

This suite of tests demonstrated that the XTS-400 DAC policies are still enforced when users are logged in through OpenSSH

Test Number	Object Permissions/ Name	Object Owner	Object Group	Action	Expected Results	Actual Results
b1	rwxrwxrwx test_ogarwx.txt	cherbig	other	read	Allowed	Allowed
b2				write	Allowed	Allowed
b3				execute	Allowed	Allowed
b4	rwxrwx--- test_ogrwx.txt	cherbig	other	read	Allowed	Allowed
b5				write	Allowed	Allowed
b6				execute	Allowed	Allowed
b7	rwx------ test_orwx.txt	cherbig	other	read	Allowed	Allowed
b8				write	Allowed	Allowed
b9				execute	Allowed	Allowed
b10	rw------- test_orw.txt	cherbig	other	read	Allowed	Allowed
b11				write	Allowed	Allowed
b12				execute	Fail	Fail
b13	r-------- test_or.txt	cherbig	other	read	Allowed	Allowed
b14				write	Fail	Fail
b15				execute	Fail	Fail
b16	--------- test_none.txt	cherbig	other	read	Fail	Fail
b17				write	Fail	Fail
b18				execute	Fail	Fail
b19	rwxrwxrwx dtest_ogarwx.txt	demo	other	read	Allowed	Allowed
b20				write	Allowed	Allowed
b21				execute	Allowed	Allowed
b22	rwxrwx--- dtest_ogrwx.txt	demo	other	read	Allowed	Allowed
b23				write	Allowed	Allowed
b24				execute	Allowed	Allowed
b25	rwxrw---- dtest_orwxgrw.txt	demo	other	read	Allowed	Allowed
b26				write	Allowed	Allowed
b27				execute	Fail	Fail
b28	rwxr----- dtest_orwxgr.txt	demo	other	read	Allowed	Allowed
b29				write	Fail	Fail
b30				execute	Fail	Fail
b31	rwx------ dtest_orwx.txt	demo	other	read	Fail	Fail
b32				write	Fail	Fail
b33				execute	Fail	Fail
b34	---rwx--- atest_grwx.txt	demo	stop	read	Fail	Fail
b35				write	Fail	Fail
b36				execute	Fail	Fail

Table 13. DAC Policy Enforcement Test Validation Results

c. TPE Testing with File Created by OpenSSH Test Validation Results

This suite of tests validated the claim that files created through OpenSSH can be viewed from the MLS LAN through the TPEs.

Test Number	TPE Login Level	Object Levels	TPE Viewable Expected Results	Actual Results
c1	sl1:il3	sl1:il3	Yes	Yes
c2		sl2:il3	No	No
c3		sl3:il3	No	No
c4	sl2:il3	sl1:il3	Yes	Yes
c5		sl2:il3	Yes	Yes
c6		sl3:il3	No	No
c7	sl3:il3	sl1:il3	Yes	Yes
c8		sl2:il3	Yes	Yes
c9		sl3:il3	Yes	Yes

Table 14. TPE Viewing Capability for OpenSSH Created Files Test Validation Results

d. TPE Testing with Files Modified by OpenSSH Test Validation Results

This suite of tests validated that files created at the console of the XTS-400 and modified by a user logged in through OpenSSH can be viewed from the MLS LAN through the TPEs.

Test Number	TPE Login Level	Object Levels	TPE Viewable Expected Results	Actual Results
d1	sl1:il3	sl1:il3	Yes	Yes
d2		sl2:il3	No	No
d3		sl3:il3	No	No
d4	sl2:il3	sl1:il3	Yes	Yes
d5		sl2:il3	Yes	Yes
d6		sl3:il3	No	No
d7	sl3:il3	sl1:il3	Yes	Yes
d8		sl2:il3	Yes	Yes
d9		sl3:il3	Yes	Yes

Table 15. TPE Viewing Capability with OpenSSH Modified Files Test Validation Results

e. Single Level LAN – Simultaneous User Logins Test Validation Results

This test demonstrated that a single level OpenSSH daemon could handle multiple simultaneous user logins.

Test Number	LAN Level	User Name	Successful Logins Expected Results	Actual Results
e1	sl1:il3	cherbig	Pass	Pass
		demo	Pass	Pass
		testuser	Pass	Pass

Table 16. Single Level LAN – Simultaneous User Logins Test Validation Results

f. Multiple Single Level LANs – Simultaneous User Logins Test Validation Results

This test demonstrated that multiple users are able to login from multiple single level LANs simultaneously.

Test Number	LAN Level	Username	Successful Login Expected Result	Actual Results
f1	sl1:il3	cherbig	Yes	Yes
	sl2:il3	demo	Yes	Yes
	sl3:il3	testuser	Yes	Yes

Table 17. Multiple Single Level LANs – Simultaneous User Logins Test Validation Results

g. Public Key Authentication Test Validation Results

This suite of tests demonstrated that the public key authentication method works properly.

Test Number	User Name	Private Key	Passphrase	Expected Results	Actual Results
g1	Valid	Correct	Correct	Succeed	Succeed
g2	Valid	Correct	Wrong	Fail	Fail
g3	Valid	Wrong	Correct	Fail	Fail
g4	Invalid	Correct	Correct	Fail	Fail

Table 18. Public Key Authentication Test Validation Results

h. Miscellaneous Test Validation Results

This suite of tests demonstrated that OpenSSH will react as expected when user accounts are not configured properly.

Test Number	Case	Environment	Expected Results	Actual Results
h1	Home directory not present	home directory not present	Cannot login	Cannot Login
h2	permissions on home directory incorrect	--------- $HOME	Cannot login	Cannot Login
h3	level on home directory incorrect	level($HOME) = sl2:il3, session level = sl1:il3	Cannot login	Cannot Login

Table 19. Miscellaneous Test Validation Results

B. MLS TEST BED TESTING

1. Test Plan

The purpose of testing the OpenSSH port in the MLS testbed is to verify OpenSSH functionality in a more realistic environment. The only tests that need to be performed are the OpenSSH login and TPE tests because the XTS-400 in the testbed should also enforce the same policies as the XTS-400 used in the developmental testing so it is redundant to repeat the MAC and DAC tests. The network topology for the MLS testbed is taken from [IRV04] and is presented in Figure 3. The clients used to connect to the XTS-400 are the web servers for each single level LAN. The XTS-400 is the MYSEA MLS server in the diagram.

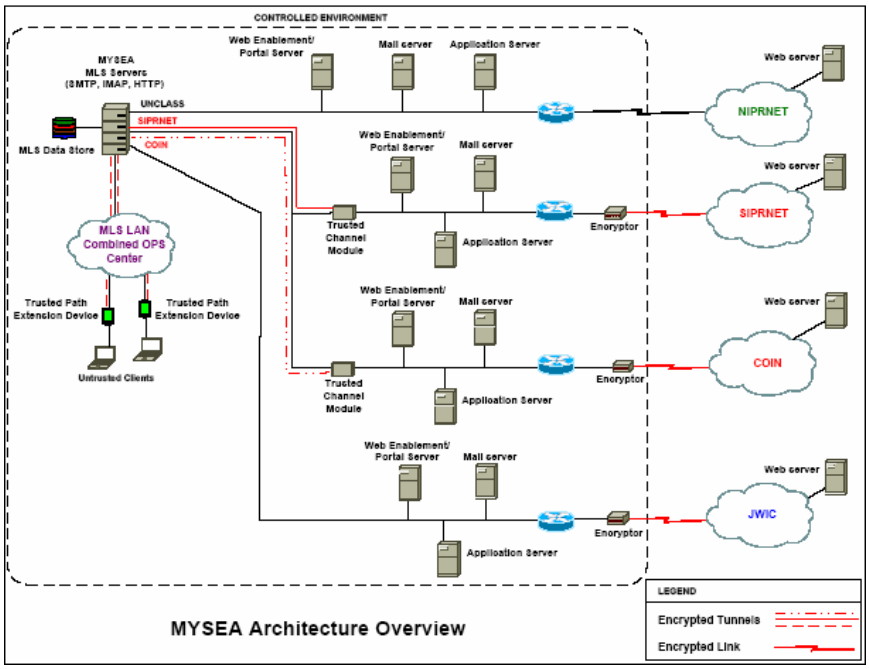

Figure 3. MLS Testbed Network Topology taken from [IRV04]

a. TPE Testing with Files Created by OpenSSH

The same test performed for the developmental testing is repeated for in the testbed environment. See the developmental test plan for more details on these tests. The test plan is presented in Table 20.

Test Number	TPE Login Level	Object Levels	TPE Viewable Expected Results
Ba1	sl1:il3	sl1:il3	Yes
Ba2		sl2:il3	No
Ba3		sl3:il3	No
Ba4	sl2:il3	sl1:il3	Yes
Ba5		sl2:il3	Yes
Ba6		sl3:il3	No
Ba7	sl3:il3	sl1:il3	Yes
Ba8		sl2:il3	Yes
Ba9		sl3:il3	Yes

Table 20. MLS Testbed – TPE Testing with Files Created Through OpenSSH

b. TPE Testing with Files Modified by OpenSSH

The same test performed for the developmental testing is repeated in the testbed environment. See the developmental test plan for more details on these tests. The test plan is presented in Table 21.

Test Number	TPE Login Level	Object Levels	TPE Viewable Expected Results
Bb1	sl1:il3	sl1:il3	Yes
Bb2		sl2:il3	No
Bb3		sl3:il3	No
Bb4	sl2:il3	sl1:il3	Yes
Bb5		sl2:il3	Yes
Bb6		sl3:il3	No
Bb7	sl3:il3	sl1:il3	Yes
Bb8		sl2:il3	Yes
Bb9		sl3:il3	Yes

Table 21. MLS Testbed – TPE Testing with Files Modified Through OpenSSH

c. Single Level LAN – Simultaneous User Logins

The same test performed for the developmental testing is repeated in the testbed environment. See the developmental test plan for more details on these tests. The test plan is presented in Table 22.

Test Number	LAN Level	User Name	Successful Logins
			Expected Results
Bc1	sl1:il3	cherbig	Pass
		demo	Pass
		testuser	Pass

Table 22. MLS Testbed – Single Level LAN – Simultaneous User Logins

d. Multiple Single Level LANs – Simultaneous User Logins

The same test performed for the developmental testing is repeated in the testbed environment. See the developmental test plan for more details on these tests. The test plan is presented in Table 23 .

Test Number	LAN Level	Username	Successful Login
			Expected Result
Bd1	sl1:il3	cherbig	Yes
	sl2:il3	demo	Yes
	sl3:il3	testuser	Yes

Table 23. MLS Testbed – Multiple Single Level LANs – Simultaneous User Logins

e. Public Key Authentication Tests

The same test performed for the developmental testing is repeated in the testbed environment. See the developmental test plan for more details on these tests. The test plan is presented in Table 24.

Test Number	User Name	Private Key	Passphrase	Expected Results
Be1	Valid	Correct	Correct	Succeed
Be2	Valid	Correct	Wrong	Fail
Be3	Valid	Wrong	Correct	Fail
Be4	Invalid	Correct	Correct	Fail

Table 24. MLS Testbed – Public Key Authentication Test

V. CONCLUSION

A. SUMMARY

This study successfully ported OpenSSH, without privilege separation enabled, to the XTS-400. Challenges caused by the functional differences between the XTS-400 and the Linux and OpenBSD platforms were encountered and solutions that did not require modification of the STOP source code were implemented. Remote login capabilities are now supported on the XTS-400 through OpenSSH and can be provided to single level LAN users through the MYSEA project's use of the XTS-400 as the MLS MYSEA server.

Developmental tests were conducted and the results of those tests were provided. These tests ensure that the security policies cannot be violated by users when they are logged in through OpenSSHA plan for validation of the results on the MLS testbed was developed, but due to time constraints, was not executed. The latter test plan is provided for future projects.

B. LESSONS LEARNED

Software porting with the XTS-400 as the target platform is difficult and time consuming; however, it can be achieved in most cases. For this study, more time should have been allocated to discovering the differences between the XTS-400 and Linux systems. This would have reduced the time spent tracking down errors. A greater knowledge of the C programming language would have reduced the time spent analyzing the OpenSSH source code and provided a greater understanding of that code. The porting process involved more time than anticipated

C. FUTURE WORK

The port of OpenSSH to the XTS-400 was successful, but the port resulted in a minor loss of functionality in OpenSSH. This section suggests future projects that could restore the lost functionality to the OpenSSH port.

One project is to implement password authentication with the XTS-400. To do this, more privileges will have to be granted to the OpenSSH daemon. The MYSEA libraries will help with this project, but the specific modifications to the OpenSSH source

code will have to be investigated to ensure that the privileges are acquired only when needed and then revoked when no longer in use. Password authentication would be very convenient for users because they would not have to carry a key file and the XTS-400 administrator would not have to worry about key generation and installation issues.

Another project is to implement support for proper group association. The *setgroups* and *initgroups* calls are not implemented and the "/etc/group" file is incorrect, so improper associations of users to groups could result in inadequate access permissions for users. This project would involve assessing the trusted group databases and implementing the functionality that *setgroups* and *initgroups* would normally implement. An analysis of where to acquire and revoke the privileges required to access the trusted databases would be required.

Another project would be to incorporate the remote shell capabilities of OpenSSH with the the MYSEA secure session server. Currently, the MYSEA server does not implement a remote shell for the MLS LAN user. The incorporation of the of the remote shell functionality would enable the MYSEA server to provide an interactive session to the user.

The entropy gathering daemon is used to provide randomness which is required for cryptography. Software-based pseudo-random number generators do not perform as well as hardware-based pseudo-random number generators. A future project would be to design and implement a pseudo-random number generator that produces sufficient randomness in terms of both quality and quantity.

A detailed vulnerability analysis should be performed for the ported OpenSSH. The XTS-400 documentation warns against network connectivity and explains the threats associated with networks [DIG03b].

APPENDIX A: SOFTWARE INSTALLATION

The purpose of this appendix is to describe the installation instructions for all of the software packages on the XTS-400 used in this project. The user should be familiar with the XTS-400 and the STOP operating system. This appendix has two main sections: support software and OpenSSH. The support software consists of the Entropy Gathering Daemon, the Zlib compression libraries and the OpenSSL encryption libraries. The Secure Attention Key (SAK) on the XTS-400 console is the "alt" and "Print Screen" keys pressed simultaneously.

A. SUPPORT SOFTWARE

For the Entropy Gathering Daemon, Zlib and OpenSSL installations, create a directory "src" in the "/usr" directory. The "src" directory must have the following mandatory levels: min:oss. Create another directory under "/usr" called "local". This directory should also be at min:oss. The MYSEA libraries will be needed in order to install OpenSSH. Installation of the MYSEA libraries should occur before any of these packages are installed.

When using the *cdtool*, to copy files from the CD-ROM drive, the levels of the files may have to be downgraded or upgraded. For now, assume that the level of the CD-ROM is min:oss. If it is not, then use *sda* to set the access level of the CD-ROM to the level of the current session.

1. Entropy Gathering Daemon

Source code for the entropy gathering daemon can be obtained from http://egd.sourceforge.net/ The user installing the daemon should be logged in as admin at a level of min:oss. It is suggested that the source be downloaded to the /usr/src directory. Untar the file by issuing the command: *tar –xvf egd.tar*. This will create a directory called "egd" and subdirectory under that directory called "egd-0.9". Navigate to the "egd-0.9" directory and issue the following commands:

perl Makefile.PL

make

make test

make install

This will install the egd.pl perl script in the /usr/bin directory. As user *admin*, enter the trusted environment by issuing the SAK.

Use *sl* to change your session level to min security and max integrity (administrator, all compartments).

Use *fsm* to copy the egd.pl script to the "/sys/daemon" directory.

Type fsm

Type copy for the request

Enter /usr/bin/egd.pl as the input path name

Enter /sys/daemon/egd.pl for the output path name

Type yes for create output file

A message is displayed that states that *n* bytes were copied.

While still in *fsm*, type change

Enter /sys/daemon/egd.pl for the pathname

Type no for modify access level

Enter network for the new owner

Enter network for the new group

Type yes for change discretionary access

Type rwx for owner

Hit enter for name of specific owner

Type rwx for group

Hit enter for specific group

Type rwx for others

Type no to display the object

Type no to hex dump the object

Type yes to okay to change

Next use *daemon_edit* to have *egd.pl* act as a daemon. Issue SAK.

Type *daemon_edit*

Type *add*.

Enter the “egd” as the name of the daemon.

For the command line type: *egd.pl*

For the arguments, use “/tmp/entropy”

Type enter for environment setting

Answer yes to starting the daemon at startup.

Answer no to high integrity.

Answer no to controls a device.

The security level for the daemon should be min.

The integrity level should be il3.

Run the daemon as user *network* and group *network*.

If seeing the current starting order for daemons is desired, type yes, otherwise say no.

To add the daemon to the end of the list, press enter otherwise enter a starting index.

Press enter for the delay at startup question.

Press enter for the delay at stop.

Type “entropy gathering daemon” for the description.

A message stating that the daemon was added should appear. To start the daemon, use the trusted *start_daemon* command. A list of available daemons will be presented. Choose *egd*. A message stating that the daemon has started should appear. To verify this, use the *proc_edit* command and type list. There should be an entry for egd.pl. It is possible that the daemon may not be able to keep up with all entropy

requests. It is better to start this daemon as soon as possible and leave it running continuously.

Each OpenSSH daemon will require its own entropy daemon. The entropy daemons will have to have the same session levels as its corresponding OpenSSH daemon. For instance, if the OpenSSH daemon will be run at sl1:il3, then an entropy daemon will have to run with those same levels. Repeat the above process three times starting at daemon_edit, specifying the session levels as those of the network interfaces (typically they will be sl1:il3, sl2:il3 and sl3:il3). You will also have to give each daemon a unique name, so for the daemon running at sl1:il3, the name should be egd-nipr. The daemon running at sl2:il3 should have the name egd-sipr. The daemon running at sl3:il3 should have the name egd-jwics.

2. Zlib compression libraries and tools

a. Installation Instructions

The zlib libraries should be installed by the admin user. The source can be downloaded from http://www.gzip.org/zlib/. Move the file "zlib.tar" to the XTS-400 and save in the /usr/src directory. Login as *admin* at sl0 and oss (il3, all compartments). Type tar -xvf zlib.tar. This will create a directory called "zlib". Change to the "zlib" directory. Run the following commands:

make test

make install

The make install will install the libz.* files in the "/usr/local/lib" directory and zlib.h in the "/usr/local/include" directory.

3. OpenSSL Encryption Libraries and Tools

a. Installation Instructions

The source for OpenSSL can be obtained from http://www.openssl.org. The current version is 0.9.7d. Download the source code or use source on the CD-ROM and load it onto the XTS-400 in the "/usr/src" directory. Login as *admin* at sl0 and oss. Un-pack the file by issuing the following command: *tar -zxvf openssl.tar.gz*. This will uncompress the files into a directory called "openssl-0.9.7d". Navigate to that directory and issue the following commands:

./config zlib

make

make test

make install

The configuration option, zlib, will allow OpenSSL to use the zlib libraries for compression. The *make* command will compile the source. The *make test* command will test the compilation and the encryption algorithms. The *make install* command will install the libraries in "/usr/local" directory. The man pages will not install properly, but they are not needed in order for openssl to function properly.

B. OPENSSH

Create a directory called "src" under the "/usr/local" directory. This directory should have the levels: min:oss, this directory and its contents will be downgraded later. The MYSEA software must be installed in the "/usr/local/mysea" directory.

Login as *admin* at min:oss. Copy the openssh.tar.gz file into the "/usr/local/src" directory.

Issue SAK

Type fsm

Type change

Enter /usr/local/src/openssh.tar.gz for the pathname

Type yes to modify the access level

Type min for security level

Type il3 for integrity level

Type yes for the question, "is the level correct?"

Hit enter for new owner name

Hit enter for new group name

Type no for modify discretionary access

Type no for display the object

Type yes for okay to change

While still in fsm,

Type change

Enter /usr/local/src for the pathname

Type yes to modify the access level

Enter min for security level

Enter il3 for the integrity level

Hit enter for new owner name

Hit enter for new group name

Type no for modify discretionary access

Type no for display the object

Type yes for okay to change

Uncompress and unpack the files by issuing the following command: tar-zxvf openssh.tar.gz. This will create a directory called “openssh-3.7.1p2”. Navigate to that directory and type the following commands:

./configure --prefix=/usr/local/src --with-prngd-socket=/tmp/entropy --with-default-path=:/xts/untrusted/bin:/bin:/usr/bin:/usr/X11R6/bin --disable-lastlog --disable-utmp --disable-utmpx --disable-wtmp --disable-wtmpx --with-ldflags=-L/usr/local/mysea/lib --with-libs=”-lut_oss –lut_xts –loss”

make

make install

The *make install* command will create directories within the “/usr/local/src” directory. The directories are: “bin”, “etc”, “libexec”, “sbin”, “share” and “man”. Verify that the file *sshd* is in the “/usr/local/src/sbin” directory. Navigate to the “/usr/local/src/etc” directory and copy the “sshd_config” file three times. Name each of

the three files a different name such as “sshd_config-nipr”, “sshd_config-sipr” and “sshd_config-jwics”. Modify each of these three files as described in Appendix C.

Invoke the Trusted Path with the SAK. Set session levels to min:max.

Run the tp_edit command.

Enter cd to change to the “/system” directory.

Type add to install the OpenSSH daemon in this directory.

For program name type *sshd*

For path enter “/usr/local/src/sbin/sshd”.

For maximum integrity enter admin.

For minimum integrity enter il0.

For assign privileges type yes and answer no to all privileges except for “**Set owner/group”**.

A message will be displayed stating that the program has been installed. Exit tp_edit.

Start daemon_edit.

Type add.

For the daemon name, specify three different daemon names for the three levels at which the OpenSSH daemons will run. Three suggested names are sshd-nipr, sshd-sipr, sshd-jwics.

For the command line type *sshd.*

For arguments type “-f /usr/local/src/etc/<sshd_config>”. Specify the configuration file that corresponds to the level of the network interface and the level of the daemon. For example, if the daemon is to run at sl1:il3, then the config file is “sshd_config-nipr”.

For the environment setting type TERM=/dev/console. This is to allow the daemon to start from the console, but it will detach from it.

Type no for start daemon at start up. This can be changed to yes, but make sure that the OpenSSH daemons start after the entropy daemons.

Enter yes for high integrity program.

Answer no for the daemon will control a device.

Enter the security level, either sl1, sl2 or sl3.

Type il3 for the integrity level for all three daemons.

For user and group names type network.

Type no for display start index.

Hit enter for end of list.

Enter 0 for delay in starting the daemon if answered no to start daemon at startup. If the question “start daemon at startup” was answered with yes, then enter a delay of 60 seconds. The entropy daemons need time to generate entropy.

Enter 0 for delay in stopping daemon. For the daemon description, type the classification of the daemon followed by “SSH Daemon”.

Issue SAK.

Set levels to min:max.

Type fsm.

Change the owner and group of the sshd, and the configuration and host keys files to the network user:

Type change

Enter “/system/sshd” for the path.

Do not modify the levels.

Enter “network” for the owner.

Enter “network” for the group.

Do not modify the discretionary access.

Type no for display.

Type yes for Okay to change.

Type change (Repeat this process for all of the configuration files)

Enter “/usr/local/src/etc/<config-file>

Do not modify the mandatory access levels.

Enter “network” for owner and group.

Do not modify the discretionary access.

Type no to display

Type yes for Okay to change.

Type change. Repeat this process for the three private keys: /usr/local/src/etc/ssh_host_rsa_key, /usr/local/src/etc/ssh_host_dsa_key, and /usr/local/src/etc/ssh_host_key.

Enter /usr/local/src/etc/<private key file>

Do not modify the mandatory access levels

Type “network” for the owner and group.

Do not modify the discretionary access.

To manually start the daemons, issue SAK.

Type *start_daemon* and enter the names of the daemons. To verify that the daemons have started, set the session level to max:max and then run *proc_edit*. Type list and all three daemons should be present. If ssh-rand-helper is also listed in proc_edit, then there is not enough entropy and the OpenSSH daemon is trying to generate more. Wait about 10 minutes or more before attempting to login. Because the daemon was installed using *tp_edit*, the command *stop_daemon* will not terminate the daemon. Using the “remove” function of *proc_edit* will not stop the daemon either. The only way to terminate the daemon is with a system reboot.

Edit the "/etc/passwd" to include the proper default group with each user. The fourth field is what is used to identify the groups. By default, these numbers will be 0. Change to the appropriate group ID for the user.

Issue SAK

Type sl

max is the security level

max is the integrity level

Issue SAK

Type ua_edit

Type display

Enter the user name of the user that will be allowed to login remotely via OpenSSH.

Look for the user's default group and remember it or write it down

Type exit

Issue SAK

Type sl

min is the security level

oss is the integrity level

Issue SAK and type run

Change directories to the /etc directory

Type vi group

Look for the user's default group and record the group ID number

Exit vi

Type vi passwd

Look for the user's entry and change the fourth field from 0 to the group ID of the user's default group

Each time a user is added, the "/etc/passwd" file will have to be updated. The *xtsmkpasswd* command, does not create a correct passwd file.

C. PUTTY INSTALLATION

This is the Windows SSH client. Do not attempt to install this on a Linux system or the XTS-400.

Login to the Windows machine as Administrator.

Copy the installer from the CD.

Double click on the installer and follow the on-screen instructions.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B: SOURCE CODE LISTING

The following files are the source code files that were modified in order to port OpenSSH to the XTS-400. All modifications are preceded with comments and the keyword, MYSEA.

A. DEFINES.H

```
/*
 * Copyright (c) 1999-2003 Damien Miller.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *     notice, this list of conditions and the following disclaimer in
the
 *         documentation and/or other materials provided with the
distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
 *  IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES
 *  OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED.
 * IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE OF
 * THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#ifndef _DEFINES_H
#define _DEFINES_H

/* $Id: defines.h,v 1.103 2003/09/16 01:52:19 dtucker Exp $ */


/* Constants */

#ifndef SHUT_RDWR
enum
{
  SHUT_RD = 0,          /* No more receptions.  */
  SHUT_WR,              /* No more transmissions.  */
```

```
  SHUT_RDWR             /* No more receptions or transmissions.  */
};
# define SHUT_RD   SHUT_RD
# define SHUT_WR   SHUT_WR
# define SHUT_RDWR SHUT_RDWR
#endif

#ifndef IPTOS_LOWDELAY
# define IPTOS_LOWDELAY          0x10
# define IPTOS_THROUGHPUT        0x08
# define IPTOS_RELIABILITY       0x04
# define IPTOS_LOWCOST           0x02
# define IPTOS_MINCOST           IPTOS_LOWCOST
#endif /* IPTOS_LOWDELAY */

#ifndef MAXPATHLEN
# ifdef PATH_MAX
#  define MAXPATHLEN PATH_MAX
# else /* PATH_MAX */
#  define MAXPATHLEN 64 /* Should be safe */
# endif /* PATH_MAX */
#endif /* MAXPATHLEN */

#ifndef STDIN_FILENO
# define STDIN_FILENO    0
#endif
#ifndef STDOUT_FILENO
# define STDOUT_FILENO   1
#endif
#ifndef STDERR_FILENO
# define STDERR_FILENO   2
#endif

#ifndef NGROUPS_MAX     /* Disable groupaccess if NGROUP_MAX is not set
*/
#ifdef NGROUPS
#define NGROUPS_MAX NGROUPS
#else
#define NGROUPS_MAX 0
#endif
#endif

#ifndef O_NONBLOCK      /* Non Blocking Open */
# define O_NONBLOCK      00004
#endif

#ifndef S_ISDIR
# define S_ISDIR(mode)  (((mode) & (_S_IFMT)) == (_S_IFDIR))
#endif /* S_ISDIR */

#ifndef S_ISREG
# define S_ISREG(mode)  (((mode) & (_S_IFMT)) == (_S_IFREG))
#endif /* S_ISREG */

#ifndef S_ISLNK
# define S_ISLNK(mode)  (((mode) & S_IFMT) == S_IFLNK)
#endif /* S_ISLNK */
```

```
#ifndef S_IXUSR
# define S_IXUSR              0000100     /*  execute/search  permission,
*/
# define S_IXGRP              0000010     /*  execute/search  permission,
*/
# define S_IXOTH              0000001     /*  execute/search  permission,
*/
# define _S_IWUSR             0000200     /* write permission, */
# define S_IWUSR              _S_IWUSR    /* write permission, owner */
# define S_IWGRP              0000020     /* write permission, group */
# define S_IWOTH              0000002     /* write permission, other */
# define S_IRUSR              0000400     /* read permission, owner */
# define S_IRGRP              0000040     /* read permission, group */
# define S_IROTH              0000004     /* read permission, other */
# define S_IRWXU              0000700     /* read, write, execute */
# define S_IRWXG              0000070     /* read, write, execute */
# define S_IRWXO              0000007     /* read, write, execute */
#endif /* S_IXUSR */

#if !defined(MAP_ANON) && defined(MAP_ANONYMOUS)
#define MAP_ANON MAP_ANONYMOUS
#endif

#ifndef MAP_FAILED
# define MAP_FAILED ((void *)-1)
#endif

/* *-*-nto-qnx doesn't define this constant in the system headers */
#ifdef MISSING_NFDBITS
# define    NFDBITS (8 * sizeof(unsigned long))
#endif

/*
SCO Open Server 3 has INADDR_LOOPBACK defined in rpc/rpc.h but
including rpc/rpc.h breaks Solaris 6
*/
#ifndef INADDR_LOOPBACK
#define INADDR_LOOPBACK ((u_long)0x7f000001)
#endif

/* Types */

/* If sys/types.h does not supply intXX_t, supply them ourselves */
/* (or die trying) */


#ifndef HAVE_U_INT
typedef unsigned int u_int;
#endif

#ifndef HAVE_INTXX_T
# if (SIZEOF_CHAR == 1)
typedef char int8_t;
# else
#  error "8 bit int type not found."
# endif
```

```
# if (SIZEOF_SHORT_INT == 2)
typedef short int int16_t;
# else
#  ifdef _UNICOS
#   if (SIZEOF_SHORT_INT == 4)
typedef short int16_t;
#   else
typedef long  int16_t;
#   endif
#  else
#   error "16 bit int type not found."
#  endif /* _UNICOS */
# endif
# if (SIZEOF_INT == 4)
typedef int int32_t;
# else
#  ifdef _UNICOS
typedef long  int32_t;
#  else
#   error "32 bit int type not found."
#  endif /* _UNICOS */
# endif
#endif

/* If sys/types.h does not supply u_intXX_t, supply them ourselves */
#ifndef HAVE_U_INTXX_T
# ifdef HAVE_UINTXX_T
typedef uint8_t u_int8_t;
typedef uint16_t u_int16_t;
typedef uint32_t u_int32_t;
# define HAVE_U_INTXX_T 1
# else
#  if (SIZEOF_CHAR == 1)
typedef unsigned char u_int8_t;
#  else
#   error "8 bit int type not found."
#  endif
#  if (SIZEOF_SHORT_INT == 2)
typedef unsigned short int u_int16_t;
#  else
#   ifdef _UNICOS
#    if (SIZEOF_SHORT_INT == 4)
typedef unsigned short u_int16_t;
#    else
typedef unsigned long  u_int16_t;
#    endif
#   else
#    error "16 bit int type not found."
#   endif
#  endif
#  if (SIZEOF_INT == 4)
typedef unsigned int u_int32_t;
#  else
#   ifdef _UNICOS
typedef unsigned long  u_int32_t;
#   else
#    error "32 bit int type not found."
```

```
#   endif
#  endif
# endif
#define __BIT_TYPES_DEFINED__
#endif

/* 64-bit types */
#ifndef HAVE_INT64_T
# if (SIZEOF_LONG_INT == 8)
typedef long int int64_t;
# else
#  if (SIZEOF_LONG_LONG_INT == 8)
typedef long long int int64_t;
#  endif
# endif
#endif
#ifndef HAVE_U_INT64_T
# if (SIZEOF_LONG_INT == 8)
typedef unsigned long int u_int64_t;
# else
#  if (SIZEOF_LONG_LONG_INT == 8)
typedef unsigned long long int u_int64_t;
#  endif
# endif
#endif

#ifndef HAVE_U_CHAR
typedef unsigned char u_char;
# define HAVE_U_CHAR
#endif /* HAVE_U_CHAR */

#ifndef SIZE_T_MAX
#define SIZE_T_MAX ULONG_MAX
#endif /* SIZE_T_MAX */

#ifndef HAVE_SIZE_T
typedef unsigned int size_t;
# define HAVE_SIZE_T
#endif /* HAVE_SIZE_T */

#ifndef HAVE_SSIZE_T
typedef int ssize_t;
# define HAVE_SSIZE_T
#endif /* HAVE_SSIZE_T */

#ifndef HAVE_CLOCK_T
typedef long clock_t;
# define HAVE_CLOCK_T
#endif /* HAVE_CLOCK_T */

#ifndef HAVE_SA_FAMILY_T
typedef int sa_family_t;
# define HAVE_SA_FAMILY_T
#endif /* HAVE_SA_FAMILY_T */

#ifndef HAVE_PID_T
typedef int pid_t;
```

```
# define HAVE_PID_T
#endif /* HAVE_PID_T */

#ifndef HAVE_SIG_ATOMIC_T
typedef int sig_atomic_t;
# define HAVE_SIG_ATOMIC_T
#endif /* HAVE_SIG_ATOMIC_T */

#ifndef HAVE_MODE_T
typedef int mode_t;
# define HAVE_MODE_T
#endif /* HAVE_MODE_T */

#if !defined(HAVE_SS_FAMILY_IN_SS) && defined(HAVE___SS_FAMILY_IN_SS)
# define ss_family __ss_family
#endif           /*           !defined(HAVE_SS_FAMILY_IN_SS)           &&
defined(HAVE_SA_FAMILY_IN_SS) */

#ifndef HAVE_SYS_UN_H
struct      sockaddr_un {
      short sun_family;       /* AF_UNIX */
      char  sun_path[108];          /* path name (gag) */
};
#endif /* HAVE_SYS_UN_H */

#if defined(BROKEN_SYS_TERMIO_H) && !defined(_STRUCT_WINSIZE)
#define _STRUCT_WINSIZE
struct winsize {
      unsigned short ws_row;          /* rows, in characters */
      unsigned short ws_col;          /* columns, in character */
      unsigned short ws_xpixel;       /* horizontal size, pixels */
      unsigned short ws_ypixel;       /* vertical size, pixels */
};
#endif

/* *-*-nto-qnx does not define this type in the system headers */
#ifdef MISSING_FD_MASK
 typedef unsigned long int    fd_mask;
#endif

/* Paths */

#ifndef _PATH_BSHELL
# define _PATH_BSHELL "/bin/sh"
#endif
#ifndef _PATH_CSHELL
# define _PATH_CSHELL "/bin/csh"
#endif
#ifndef _PATH_SHELLS
# define _PATH_SHELLS "/etc/shells"
#endif

#ifdef USER_PATH
# ifdef _PATH_STDPATH
#  undef _PATH_STDPATH
# endif
# define _PATH_STDPATH USER_PATH
```

```
#endif

#ifndef _PATH_STDPATH
# define _PATH_STDPATH "/usr/bin:/bin:/usr/sbin:/sbin"
#endif

#ifndef SUPERUSER_PATH
# define SUPERUSER_PATH _PATH_STDPATH
#endif

#ifndef _PATH_DEVNULL
# define _PATH_DEVNULL "/dev/null"
#endif

#ifndef MAIL_DIRECTORY
# define MAIL_DIRECTORY "/var/spool/mail"
#endif

#ifndef MAILDIR
# define MAILDIR MAIL_DIRECTORY
#endif

#if !defined(_PATH_MAILDIR) && defined(MAILDIR)
# define _PATH_MAILDIR MAILDIR
#endif /* !defined(_PATH_MAILDIR) && defined(MAILDIR) */

#ifndef _PATH_NOLOGIN
# define _PATH_NOLOGIN "/etc/nologin"
#endif

/* Define this to be the path of the xauth program. */
#ifdef XAUTH_PATH
#define _PATH_XAUTH XAUTH_PATH
#endif /* XAUTH_PATH */

/* derived from XF4/xc/lib/dps/Xlibnet.h */
#ifndef X_UNIX_PATH
#  ifdef __hpux
#    define X_UNIX_PATH "/var/spool/sockets/X11/%u"
#  else
#    define X_UNIX_PATH "/tmp/.X11-unix/X%u"
#  endif
#endif /* X_UNIX_PATH */
#define _PATH_UNIX_X X_UNIX_PATH

#ifndef _PATH_TTY
# define _PATH_TTY "/dev/tty"
#endif

/* Macros */

#if defined(HAVE_LOGIN_GETCAPBOOL) && defined(HAVE_LOGIN_CAP_H)
# define HAVE_LOGIN_CAP
#endif

#ifndef MAX
# define MAX(a,b) (((a)>(b))?(a):(b))
```

```
# define MIN(a,b) (((a)<(b))?(a):(b))
#endif

#ifndef roundup
# define roundup(x, y)   ((((x)+((y)-1))/(y))*(y))
#endif

#ifndef timersub
#define timersub(a, b, result)                                  \
   do {                                                         \
      (result)->tv_sec = (a)->tv_sec - (b)->tv_sec;             \
      (result)->tv_usec = (a)->tv_usec - (b)->tv_usec;          \
      if ((result)->tv_usec < 0) {                              \
       --(result)->tv_sec;                                      \
       (result)->tv_usec += 1000000;                            \
      }                                                         \
   } while (0)
#endif

#ifndef TIMEVAL_TO_TIMESPEC
#define     TIMEVAL_TO_TIMESPEC(tv, ts) {                       \
      (ts)->tv_sec = (tv)->tv_sec;                              \
      (ts)->tv_nsec = (tv)->tv_usec * 1000;                     \
}
#endif

#ifndef TIMESPEC_TO_TIMEVAL
#define     TIMESPEC_TO_TIMEVAL(tv, ts) {                       \
      (tv)->tv_sec = (ts)->tv_sec;                              \
      (tv)->tv_usec = (ts)->tv_nsec / 1000;                     \
}
#endif

#ifndef __P
# define __P(x) x
#endif

#if !defined(IN6_IS_ADDR_V4MAPPED)
# define IN6_IS_ADDR_V4MAPPED(a) \
      ((((u_int32_t *) (a))[0] == 0) && (((u_int32_t *) (a))[1] == 0)
&& \
       (((u_int32_t *) (a))[2] == htonl (0xffff)))
#endif /* !defined(IN6_IS_ADDR_V4MAPPED) */

#if !defined(__GNUC__) || (__GNUC__ < 2)
# define __attribute__(x)
#endif /* !defined(__GNUC__) || (__GNUC__ < 2) */

/* *-*-nto-qnx doesn't define this macro in the system headers */
#ifdef MISSING_HOWMANY
# define howmany(x,y)   (((x)+((y)-1))/(y))
#endif

#ifndef OSSH_ALIGNBYTES
#define OSSH_ALIGNBYTES (sizeof(int) - 1)
#endif
#ifndef __CMSG_ALIGN
```

```
#define     __CMSG_ALIGN(p)    (((u_int)(p)    +    OSSH_ALIGNBYTES)    &~
OSSH_ALIGNBYTES)
#endif

/* Length of the contents of a control message of length len */
#ifndef CMSG_LEN
#define     CMSG_LEN(len)     (__CMSG_ALIGN(sizeof(struct   cmsghdr))   +
(len))
#endif

/* Length of the space taken up by a padded control message of length
len */
#ifndef CMSG_SPACE
#define     CMSG_SPACE(len)   (__CMSG_ALIGN(sizeof(struct   cmsghdr))   +
__CMSG_ALIGN(len))
#endif

/* given pointer to struct cmsghdr, return pointer to data */
#ifndef CMSG_DATA
#define CMSG_DATA(cmsg) ((u_char *)(cmsg) + __CMSG_ALIGN(sizeof(struct
cmsghdr)))
#endif /* CMSG_DATA */

/*
 * RFC 2292 requires to check msg_controllen, in case that the kernel
returns
 * an empty list for some reasons.
 */
#ifndef CMSG_FIRSTHDR
#define CMSG_FIRSTHDR(mhdr) \
      ((mhdr)->msg_controllen >= sizeof(struct cmsghdr) ? \
       (struct cmsghdr *)(mhdr)->msg_control : \
       (struct cmsghdr *)NULL)
#endif /* CMSG_FIRSTHDR */


/* Function replacement / compatibility hacks */

#if   !defined(HAVE_GETADDRINFO)   &&   (defined(HAVE_OGETADDRINFO)   ||
defined(HAVE_NGETADDRINFO))
# define HAVE_GETADDRINFO
#endif

#ifndef HAVE_GETOPT_OPTRESET
# undef getopt
# undef opterr
# undef optind
# undef optopt
# undef optreset
# undef optarg
# define getopt(ac, av, o)  BSDgetopt(ac, av, o)
# define opterr             BSDopterr
# define optind             BSDoptind
# define optopt             BSDoptopt
# define optreset           BSDoptreset
# define optarg             BSDoptarg
#endif
```

```
/* In older versions of libpam, pam_strerror takes a single argument */
#ifdef HAVE_OLD_PAM
# define PAM_STRERROR(a,b) pam_strerror((b))
#else
# define PAM_STRERROR(a,b) pam_strerror((a),(b))
#endif

#ifdef PAM_SUN_CODEBASE
# define PAM_MSG_MEMBER(msg, n, member) ((*(msg))[(n)].member)
#else
# define PAM_MSG_MEMBER(msg, n, member) ((msg)[(n)]->member)
#endif

#if defined(BROKEN_GETADDRINFO) && defined(HAVE_GETADDRINFO)
# undef HAVE_GETADDRINFO
#endif
#if defined(BROKEN_GETADDRINFO) && defined(HAVE_FREEADDRINFO)
# undef HAVE_FREEADDRINFO
#endif
#if defined(BROKEN_GETADDRINFO) && defined(HAVE_GAI_STRERROR)
# undef HAVE_GAI_STRERROR
#endif

#if !defined(HAVE_MEMMOVE) && defined(HAVE_BCOPY)
# define memmove(s1, s2, n) bcopy((s2), (s1), (n))
#endif /* !defined(HAVE_MEMMOVE) && defined(HAVE_BCOPY) */

#if defined(HAVE_VHANGUP) && !defined(HAVE_DEV_PTMX)
#  define USE_VHANGUP
#endif /* defined(HAVE_VHANGUP) && !defined(HAVE_DEV_PTMX) */

#ifndef GETPGRP_VOID
# define getpgrp() getpgrp(0)
#endif

/* OPENSSL_free() is Free() in versions before OpenSSL 0.9.6 */
#if   !defined(OPENSSL_VERSION_NUMBER)   ||   (OPENSSL_VERSION_NUMBER   <
0x0090600f)
# define OPENSSL_free(x) Free(x)
#endif

#if !defined(HAVE___func__) && defined(HAVE___FUNCTION__)
#  define __func__ __FUNCTION__
#elif !defined(HAVE___func__)
#  define __func__ ""
#endif

#if defined(KRB5) && !defined(HEIMDAL)
#  define krb5_get_err_text(context,code) error_message(code)
#endif

/*
 * Define this to use pipes instead of socketpairs for communicating
with the
 * client program.  Socketpairs do not seem to work on all systems.
 *
```

```
 *  configure.ac  sets  this  for  a  few  OS's  which  are  known  to  have
problems
 * but you may need to set it yourself
 */
//MYSEA: Manually changed to use pipes
#define USE_PIPES 1

/**
 ** login recorder definitions
 **/

/* FIXME: put default paths back in */
#ifndef UTMP_FILE
#  ifdef _PATH_UTMP
#    define UTMP_FILE _PATH_UTMP
#  else
#    ifdef CONF_UTMP_FILE
#      define UTMP_FILE CONF_UTMP_FILE
#    endif
#  endif
#endif
#ifndef WTMP_FILE
#  ifdef _PATH_WTMP
#    define WTMP_FILE _PATH_WTMP
#  else
#    ifdef CONF_WTMP_FILE
#      define WTMP_FILE CONF_WTMP_FILE
#    endif
#  endif
#endif
/* pick up the user's location for lastlog if given */
#ifndef LASTLOG_FILE
#  ifdef _PATH_LASTLOG
#    define LASTLOG_FILE _PATH_LASTLOG
#  else
#    ifdef CONF_LASTLOG_FILE
#      define LASTLOG_FILE CONF_LASTLOG_FILE
#    endif
#  endif
#endif


/* The login() library function in libutil is first choice */
#if defined(HAVE_LOGIN) && !defined(DISABLE_LOGIN)
#  define USE_LOGIN

#else
/* Simply select your favourite login types. */
/* Can't do if-else because some systems use several... <sigh> */
#  if defined(UTMPX_FILE) && !defined(DISABLE_UTMPX)
#    define USE_UTMPX
#  endif
#  if defined(UTMP_FILE) && !defined(DISABLE_UTMP)
#    define USE_UTMP
#  endif
#  if defined(WTMPX_FILE) && !defined(DISABLE_WTMPX)
#    define USE_WTMPX
```

```
#  endif
#  if defined(WTMP_FILE) && !defined(DISABLE_WTMP)
#    define USE_WTMP
#  endif

#endif

/* I hope that the presence of LASTLOG_FILE is enough to detect this */
#if defined(LASTLOG_FILE) && !defined(DISABLE_LASTLOG)
#  define USE_LASTLOG
#endif

/** end of login recorder definitions */

#endif /* _DEFINES_H */
```

B. SESSION.C

```
/*
 * Copyright (c) 1995 Tatu Ylonen <ylo@cs.hut.fi>, Espoo, Finland
 *                    All rights reserved
 *
 * As far as I am concerned, the code I have written for this software
 * can be used freely for any purpose.  Any derived versions of this
 * software must be clearly marked as such, and if the derived work is
 * incompatible with the protocol description in the RFC file, it must
be
 * called by a name other than "ssh" or "Secure Shell".
 *
 * SSH2 support by Markus Friedl.
 * Copyright (c) 2000, 2001 Markus Friedl.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *     notice, this list of conditions and the following disclaimer in
the
 *         documentation  and/or  other  materials  provided  with  the
distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
 *  IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES
 *  OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED.
 * IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 *  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE OF
```

```
 * THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include "includes.h"
RCSID("$OpenBSD: session.c,v 1.164 2003/09/18 08:49:45 markus Exp $");

#include "ssh.h"
#include "ssh1.h"
#include "ssh2.h"
#include "xmalloc.h"
#include "sshpty.h"
#include "packet.h"
#include "buffer.h"
#include "mpaux.h"
#include "uidswap.h"
#include "compat.h"
#include "channels.h"
#include "bufaux.h"
#include "auth.h"
#include "auth-options.h"
#include "pathnames.h"
#include "log.h"
#include "servconf.h"
#include "sshlogin.h"
#include "serverloop.h"
#include "canohost.h"
#include "session.h"
#include "monitor_wrap.h"

#ifdef GSSAPI
#include "ssh-gss.h"
#endif

/* func */

Session *session_new(void);
void  session_set_fds(Session *, int, int, int);
void  session_pty_cleanup(void *);
void  session_proctitle(Session *);
int   session_setup_x11fwd(Session *);
void  do_exec_pty(Session *, const char *);
void  do_exec_no_pty(Session *, const char *);
void  do_exec(Session *, const char *);
void  do_login(Session *, const char *);
#ifdef LOGIN_NEEDS_UTMPX
static void do_pre_login(Session *s);
#endif
void  do_child(Session *, const char *);
void  do_motd(void);
int   check_quietlogin(Session *, const char *);

static void do_authenticated1(Authctxt *);
static void do_authenticated2(Authctxt *);

static int session_pty_req(Session *);

/* import */
```

```
extern ServerOptions options;
extern char *__progname;
extern int log_stderr;
extern int debug_flag;
extern u_int utmp_len;
extern int startup_pipe;
extern void destroy_sensitive_data(void);
extern Buffer loginmsg;

/* original command from peer. */
const char *original_command = NULL;

/* data */
#define MAX_SESSIONS 10
Session     sessions[MAX_SESSIONS];

#ifdef HAVE_LOGIN_CAP
login_cap_t *lc;
#endif

/* Name and directory of socket for authentication agent forwarding. */
static char *auth_sock_name = NULL;
static char *auth_sock_dir = NULL;

/* removes the agent forwarding socket */

static void
auth_sock_cleanup_proc(void *_pw)
{
      struct passwd *pw = _pw;

      if (auth_sock_name != NULL) {
            temporarily_use_uid(pw);
            unlink(auth_sock_name);
            rmdir(auth_sock_dir);
            auth_sock_name = NULL;
            restore_uid();
      }
}

static int
auth_input_request_forwarding(struct passwd * pw)
{
      Channel *nc;
      int sock;
      struct sockaddr_un sunaddr;

      if (auth_sock_name != NULL) {
            error("authentication forwarding requested twice.");
            return 0;
      }

      /* Temporarily drop privileged uid for mkdir/bind. */
      temporarily_use_uid(pw);

      /* Allocate a buffer for the socket name, and format the name. */
      auth_sock_name = xmalloc(MAXPATHLEN);
```

```
	auth_sock_dir = xmalloc(MAXPATHLEN);
	strlcpy(auth_sock_dir, "/tmp/ssh-XXXXXXXX", MAXPATHLEN);

	/* Create private directory for socket */
	if (mkdtemp(auth_sock_dir) == NULL) {
		packet_send_debug("Agent forwarding disabled: "
		    "mkdtemp() failed: %.100s", strerror(errno));
		restore_uid();
		xfree(auth_sock_name);
		xfree(auth_sock_dir);
		auth_sock_name = NULL;
		auth_sock_dir = NULL;
		return 0;
	}
	snprintf(auth_sock_name, MAXPATHLEN, "%s/agent.%ld",
		 auth_sock_dir, (long) getpid());

	/* delete agent socket on fatal() */
	fatal_add_cleanup(auth_sock_cleanup_proc, pw);

	/* Create the socket. */
	sock = socket(AF_UNIX, SOCK_STREAM, 0);
	if (sock < 0)
		packet_disconnect("socket: %.100s", strerror(errno));

	/* Bind it to the name. */
	memset(&sunaddr, 0, sizeof(sunaddr));
	sunaddr.sun_family = AF_UNIX;
	strlcpy(sunaddr.sun_path,                          auth_sock_name,
sizeof(sunaddr.sun_path));

	if (bind(sock, (struct sockaddr *) & sunaddr, sizeof(sunaddr)) <
0)
		packet_disconnect("bind: %.100s", strerror(errno));

	/* Restore the privileged uid. */
	restore_uid();

	/* Start listening on the socket. */
	if (listen(sock, 5) < 0)
		packet_disconnect("listen: %.100s", strerror(errno));

	/* Allocate a channel for the authentication agent socket. */
	nc = channel_new("auth socket",
	    SSH_CHANNEL_AUTH_SOCKET, sock, sock, -1,
	    CHAN_X11_WINDOW_DEFAULT, CHAN_X11_PACKET_DEFAULT,
	    0, "auth socket", 1);
	strlcpy(nc->path, auth_sock_name, sizeof(nc->path));
	return 1;
}


void
do_authenticated(Authctxt *authctxt)
{
	setproctitle("%s", authctxt->pw->pw_name);
```

```
	/*
	 * Cancel the alarm we set to limit the time taken for
	 * authentication.
	 */
	alarm(0);
	if (startup_pipe != -1) {
		close(startup_pipe);
		startup_pipe = -1;
	}

	/* setup the channel layer */
	if (!no_port_forwarding_flag && options.allow_tcp_forwarding)
		channel_permit_all_opens();

	if (compat20)
		do_authenticated2(authctxt);
	else
		do_authenticated1(authctxt);

	/* remove agent socket */
	if (auth_sock_name != NULL)
		auth_sock_cleanup_proc(authctxt->pw);
#ifdef KRB5
	if (options.kerberos_ticket_cleanup)
		krb5_cleanup_proc(authctxt);
#endif
}

/*
 * Prepares for an interactive session.  This is called after the user
has
 *  been  successfully  authenticated.   During  this  message  exchange,
pseudo
 *  terminals  are  allocated,  X11,  TCP/IP,  and  authentication  agent
forwardings
 * are requested, etc.
 */
static void
do_authenticated1(Authctxt *authctxt)
{
	Session *s;
	char *command;
	int success, type, screen_flag;
	int enable_compression_after_reply = 0;
	u_int proto_len, data_len, dlen, compression_level = 0;

	s = session_new();
	s->authctxt = authctxt;
	s->pw = authctxt->pw;

	/*
	 * We stay in this loop until the client requests to execute a
shell
	 * or a command.
	 */
	for (;;) {
		success = 0;
```

```
            /* Get a packet from the client. */
            type = packet_read();

            /* Process the packet. */
            switch (type) {
            case SSH_CMSG_REQUEST_COMPRESSION:
                  compression_level = packet_get_int();
                  packet_check_eom();
                  if (compression_level < 1 || compression_level > 9) {
                        packet_send_debug("Received illegal compression
level %d.",
                            compression_level);
                        break;
                  }
                  if (!options.compression) {
                        debug2("compression disabled");
                        break;
                  }
                  /* Enable compression after we have responded with
SUCCESS. */
                  enable_compression_after_reply = 1;
                  success = 1;
                  break;

            case SSH_CMSG_REQUEST_PTY:
                  success = session_pty_req(s);
                  break;

            case SSH_CMSG_X11_REQUEST_FORWARDING:
                  s->auth_proto = packet_get_string(&proto_len);
                  s->auth_data = packet_get_string(&data_len);

                  screen_flag = packet_get_protocol_flags() &
                      SSH_PROTOFLAG_SCREEN_NUMBER;
                  debug2("SSH_PROTOFLAG_SCREEN_NUMBER:              %d",
screen_flag);

                  if (packet_remaining() == 4) {
                        if (!screen_flag)
                              debug2("Buggy client: "
                                  "X11 screen flag missing");
                        s->screen = packet_get_int();
                  } else {
                        s->screen = 0;
                  }
                  packet_check_eom();
                  success = session_setup_x11fwd(s);
                  if (!success) {
                        xfree(s->auth_proto);
                        xfree(s->auth_data);
                        s->auth_proto = NULL;
                        s->auth_data = NULL;
                  }
                  break;

            case SSH_CMSG_AGENT_REQUEST_FORWARDING:
```

```
                if (no_agent_forwarding_flag || compat13) {
                        debug("Authentication agent forwarding not
permitted for this authentication.");
                        break;
                }
                debug("Received authentication agent forwarding
request.");
                success = auth_input_request_forwarding(s->pw);
                break;

        case SSH_CMSG_PORT_FORWARD_REQUEST:
                if (no_port_forwarding_flag) {
                        debug("Port forwarding not permitted for this
authentication.");
                        break;
                }
                if (!options.allow_tcp_forwarding) {
                        debug("Port forwarding not permitted.");
                        break;
                }
                debug("Received TCP/IP port forwarding request.");
                channel_input_port_forward_request(s->pw->pw_uid    ==
0, options.gateway_ports);
                success = 1;
                break;

        case SSH_CMSG_MAX_PACKET_SIZE:
                if (packet_set_maxsize(packet_get_int()) > 0)
                        success = 1;
                break;

        case SSH_CMSG_EXEC_SHELL:
        case SSH_CMSG_EXEC_CMD:
                if (type == SSH_CMSG_EXEC_CMD) {
                        command = packet_get_string(&dlen);
                        debug("Exec command '%.500s'", command);
                        do_exec(s, command);
                        xfree(command);
                } else {
                        do_exec(s, NULL);
                }
                packet_check_eom();
                debug("Calling Session Close");
                session_close(s);
                debug("Returned from Session Close called from do
Authenticated1");
                return;

        default:
                /*
                 * Any unknown messages in this phase are ignored,
                 * and a failure message is returned.
                 */
                logit("Unknown    packet    type    received    after
authentication: %d", type);

}
```

```
            packet_start(success       ?       SSH_SMSG_SUCCESS       :
SSH_SMSG_FAILURE);
            packet_send();
            packet_write_wait();

            /*  Enable  compression  now  that  we  have  replied  if
appropriate. */
            if (enable_compression_after_reply) {
                  enable_compression_after_reply = 0;
                  packet_start_compression(compression_level);
            }
      }
}

/*
 * This is called to fork and execute a command when we have no tty.
This
 * will call do_child from the child, and server_loop from the parent
after
 * setting up file descriptors and such.
 */
void
do_exec_no_pty(Session *s, const char *command)
{
      pid_t pid;

#ifdef USE_PIPES
      int pin[2], pout[2], perr[2];
      /* Allocate pipes for communicating with the program. */
      if (pipe(pin) < 0 || pipe(pout) < 0 || pipe(perr) < 0)
            packet_disconnect("Could not create pipes: %.100s",
                              strerror(errno));
#else /* USE_PIPES */
      int inout[2], err[2];
      /* Uses socket pairs to communicate with the program. */
      if (socketpair(AF_UNIX, SOCK_STREAM, 0, inout) < 0 ||
          socketpair(AF_UNIX, SOCK_STREAM, 0, err) < 0)
            packet_disconnect("Could not create socket pairs: %.100s",
                              strerror(errno));
#endif /* USE_PIPES */
      if (s == NULL)
            fatal("do_exec_no_pty: no session");

      session_proctitle(s);

#if defined(USE_PAM)
      if (options.use_pam) {
            do_pam_setcred(1);
            if (is_pam_password_change_required())
                  packet_disconnect("Password change required but no "
                      "TTY available");
      }
#endif /* USE_PAM */

      /* Fork the child. */
      if ((pid = fork()) == 0) {
            fatal_remove_all_cleanups();
```

```
            /* Child.  Reinitialize the log since the pid has changed.
*/
            log_init(__progname,                      options.log_level,
options.log_facility, log_stderr);

            /*
             * Create a new session and process group since the 4.4BSD
             * setlogin() affects the entire process group.
             */
            if (setsid() < 0)
                  error("setsid failed: %.100s", strerror(errno));

#ifdef USE_PIPES
            /*
             * Redirect stdin.  We close the parent side of the socket
             * pair, and make the child side the standard input.
             */
            close(pin[1]);
            if (dup2(pin[0], 0) < 0)
                  perror("dup2 stdin");
            close(pin[0]);

            /* Redirect stdout. */
            close(pout[0]);
            if (dup2(pout[1], 1) < 0)
                  perror("dup2 stdout");
            close(pout[1]);

            /* Redirect stderr. */
            close(perr[0]);
            if (dup2(perr[1], 2) < 0)
                  perror("dup2 stderr");
            close(perr[1]);
#else /* USE_PIPES */
            /*
             *  Redirect  stdin,  stdout,  and  stderr.   Stdin  and  stdout
will
             *  use  the  same  socket,  as  some  programs  (particularly
rdist)
             * seem to depend on it.
             */
            close(inout[1]);
            close(err[1]);
            if (dup2(inout[0], 0) < 0)    /* stdin */
                  perror("dup2 stdin");
            if (dup2(inout[0], 1) < 0)    /* stdout.  Note: same socket
as stdin. */
                  perror("dup2 stdout");
            if (dup2(err[0], 2) < 0)      /* stderr */
                  perror("dup2 stderr");
#endif /* USE_PIPES */

#ifdef _UNICOS
            cray_init_job(s->pw); /* set up cray jid and tmpdir */
#endif
```

```
            /* Do processing for the child (exec command etc). */
            do_child(s, command);
            /* NOTREACHED */
      }
#ifdef _UNICOS
      signal(WJSIGNAL, cray_job_termination_handler);
#endif /* _UNICOS */
#ifdef HAVE_CYGWIN
      if (is_winnt)
            cygwin_set_impersonation_token(INVALID_HANDLE_VALUE);
#endif
      if (pid < 0)
            packet_disconnect("fork failed: %.100s", strerror(errno));
      s->pid = pid;
      /* Set interactive/non-interactive mode. */
      packet_set_interactive(s->display != NULL);
#ifdef USE_PIPES
      /* We are the parent.  Close the child sides of the pipes. */
      close(pin[0]);
      close(pout[1]);
      close(perr[1]);

      if (compat20) {
            session_set_fds(s, pin[1], pout[0], s->is_subsystem ? -1 :
perr[0]);
      } else {
            /* Enter the interactive session. */
            server_loop(pid, pin[1], pout[0], perr[0]);
            /* server_loop has closed pin[1], pout[0], and perr[0]. */
      }
#else /* USE_PIPES */
      /* We are the parent.  Close the child sides of the socket pairs.
*/
      close(inout[0]);
      close(err[0]);

      /*
       * Enter the interactive session.  Note: server_loop must be able
to
       * handle the case that fdin and fdout are the same.
       */
      if (compat20) {
            session_set_fds(s, inout[1], inout[1], s->is_subsystem ? -1
: err[1]);
      } else {
            server_loop(pid, inout[1], inout[1], err[1]);
            /* server_loop has closed inout[1] and err[1]. */
      }
#endif /* USE_PIPES */
}

/*
 * This is called to fork and execute a command when we have a tty.
This
 * will call do_child from the child, and server_loop from the parent
after
 * setting up file descriptors, controlling tty, updating wtmp, utmp,
```

```
 * lastlog, and other such operations.
 */
void
do_exec_pty(Session *s, const char *command)
{
	int fdout, ptyfd, ttyfd, ptymaster;
	pid_t pid;
	if (s == NULL)
	{
		fatal("do_exec_pty: no session");
	}
	ptyfd = s->ptyfd;
	ttyfd = s->ttyfd;

#if defined(USE_PAM)
	if (options.use_pam) {
		do_pam_set_tty(s->tty);
		do_pam_setcred(1);
	}
#endif
	/* Fork the child. */
	if ((pid = fork()) == 0) {
		fatal_remove_all_cleanups();
		/*  Child.    Reinitialize  the  log  because  the  pid  has
changed. */
		log_init(__progname,                      options.log_level,
options.log_facility, log_stderr);
		/* Close the master side of the pseudo tty. */
		close(ptyfd);

		/* Make the pseudo tty our controlling tty. */
		pty_make_controlling_tty(&ttyfd, s->tty);

		/* Redirect stdin/stdout/stderr from the pseudo tty. */
		if (dup2(ttyfd, 0) < 0)
		{
			error("dup2 stdin: %s", strerror(errno));
		}
		if (dup2(ttyfd, 1) < 0)
		{
			error("dup2 stdout: %s", strerror(errno));
		}
		if (dup2(ttyfd, 2) < 0)
		{
			error("dup2 stderr: %s", strerror(errno));
		}

		/* Close the extra descriptor for the pseudo tty. */
		close(ttyfd);

		/* record login, etc. similar to login(1) */

#ifndef HAVE_OSF_SIA
		if (!(options.use_login && command == NULL)) {
#ifdef _UNICOS
			cray_init_job(s->pw); /*  set  up  cray  jid  and  tmpdir
*/
```

```
#endif /* _UNICOS */
                do_login(s, command);
          }
# ifdef LOGIN_NEEDS_UTMPX
          else
                do_pre_login(s);
# endif
#endif

          /* Do common processing for the child, such as execing the
command. */
          do_child(s, command);
          /* NOTREACHED */
     }
#ifdef _UNICOS
     signal(WJSIGNAL, cray_job_termination_handler);
#endif /* _UNICOS */
#ifdef HAVE_CYGWIN
     if (is_winnt)
          cygwin_set_impersonation_token(INVALID_HANDLE_VALUE);
#endif
     if (pid < 0)
          packet_disconnect("fork failed: %.100s", strerror(errno));
     s->pid = pid;

     /* Parent.  Close the slave side of the pseudo tty. */
     close(ttyfd);

     /*
      * Create another descriptor of the pty master side for use as
the
      * standard input.   We could use the original descriptor, but
this
      *   simplifies   code   in   server_loop.     The   descriptor   is
bidirectional.
      */
     fdout = dup(ptyfd);
     if (fdout < 0)
          packet_disconnect("dup       #1       failed:       %.100s",
strerror(errno));

     /* we keep a reference to the pty master */
     ptymaster = dup(ptyfd);
     if (ptymaster < 0)
          packet_disconnect("dup       #2       failed:       %.100s",
strerror(errno));
     s->ptymaster = ptymaster;

     /* Enter interactive session. */
     packet_set_interactive(1);
     if (compat20) {
          session_set_fds(s, ptyfd, fdout, -1);
     } else {
          server_loop(pid, ptyfd, fdout, -1);
          /* server_loop _has_ closed ptyfd and fdout. */
     }
}
```

```
#ifdef LOGIN_NEEDS_UTMPX
static void
do_pre_login(Session *s)
{
      socklen_t fromlen;
      struct sockaddr_storage from;
      pid_t pid = getpid();

      /*
       * Get IP address of client. If the connection is not a socket,
let
       * the address be 0.0.0.0.
       */
      memset(&from, 0, sizeof(from));
      fromlen = sizeof(from);
      if (packet_connection_is_on_socket()) {
            if (getpeername(packet_get_connection_in(),
                (struct sockaddr *) & from, &fromlen) < 0) {
                  debug("getpeername: %.100s", strerror(errno));
                  fatal_cleanup();
            }
      }

      record_utmp_only(pid, s->tty, s->pw->pw_name,
          get_remote_name_or_ip(utmp_len, options.use_dns),
          (struct sockaddr *)&from, fromlen);
}
#endif

/*
 * This is called to fork and execute a command.  If another command is
 * to be forced, execute that instead.
 */
void
do_exec(Session *s, const char *command)
{
      if (forced_command) {
            original_command = command;
            command = forced_command;
            debug("Forced command '%.900s'", command);
      }

#ifdef GSSAPI
      if (options.gss_authentication) {
            temporarily_use_uid(s->pw);
            ssh_gssapi_storecreds();
            restore_uid();
      }
#endif

      if (s->ttyfd != -1)
      {
            do_exec_pty(s, command);
      }
      else
      {
```

```
            do_exec_no_pty(s, command);
      }

      original_command = NULL;
}


/* administrative, login(1)-like work */
void
do_login(Session *s, const char *command)
{
      char *time_string;
      socklen_t fromlen;
      struct sockaddr_storage from;
      struct passwd * pw = s->pw;
      pid_t pid = getpid();
      /*
       * Get IP address of client. If the connection is not a socket,
let
       * the address be 0.0.0.0.
       */
      memset(&from, 0, sizeof(from));
      fromlen = sizeof(from);
      if (packet_connection_is_on_socket()) {
            if (getpeername(packet_get_connection_in(),
                (struct sockaddr *) & from, &fromlen) < 0) {
                  debug("getpeername: %.100s", strerror(errno));
                  fatal_cleanup();
            }
      }

      /* Record that there was a login on that tty from the remote
host. */

      if (!use_privsep)
            record_login(pid, s->tty, pw->pw_name, pw->pw_uid,
                get_remote_name_or_ip(utmp_len,
                options.use_dns),
                (struct sockaddr *)&from, fromlen);

#ifdef USE_PAM
      /*
       * If password change is needed, do it now.
       * This needs to occur before the ~/.hushlogin check.
       */
      if (options.use_pam && is_pam_password_change_required()) {
            print_pam_messages();
            do_pam_chauthtok();
            /* XXX - signal [net] parent to enable forwardings */
      }
#endif
      if (check_quietlogin(s, command))
      {
            return;
      }

#ifdef USE_PAM
```

```
	if (options.use_pam && !is_pam_password_change_required())
		print_pam_messages();
#endif /* USE_PAM */

	/* display post-login message */
	if (buffer_len(&loginmsg) > 0) {
		buffer_append(&loginmsg, "\0", 1);
		printf("%s\n", (char *)buffer_ptr(&loginmsg));
	}
	buffer_free(&loginmsg);

#ifndef NO_SSH_LASTLOG
	if (options.print_lastlog && s->last_login_time != 0) {
		time_string = ctime(&s->last_login_time);
		if (strchr(time_string, '\n'))
			*strchr(time_string, '\n') = 0;
		if (strcmp(s->hostname, "") == 0)
			printf("Last login: %s\r\n", time_string);
		else
			printf("Last login: %s from %s\r\n", time_string,
			    s->hostname);
	}
#endif /* NO_SSH_LASTLOG */
	do_motd();
}

/*
 * Display the message of the day.
 */
void
do_motd(void)
{
	FILE *f;
	char buf[256];

	if (options.print_motd) {
#ifdef HAVE_LOGIN_CAP
		f = fopen(login_getcapstr(lc, "welcome", "/etc/motd",
		    "/etc/motd"), "r");
#else
		f = fopen("/etc/motd", "r");
#endif
		if (f) {
			while (fgets(buf, sizeof(buf), f))
				fputs(buf, stdout);
			fclose(f);
		}
	}
}


/*
 * Check for quiet login, either .hushlogin or command given.
 */
int
check_quietlogin(Session *s, const char *command)
{
```

```
	char buf[256];
	struct passwd *pw = s->pw;
	struct stat st;

	/* Return 1 if .hushlogin exists or a command given. */
	if (command != NULL)
		return 1;
	snprintf(buf, sizeof(buf), "%.200s/.hushlogin", pw->pw_dir);
#ifdef HAVE_LOGIN_CAP
	if (login_getcapbool(lc, "hushlogin", 0) || stat(buf, &st) >= 0)
		return 1;
#else
	if (stat(buf, &st) >= 0)
		return 1;
#endif
	return 0;
}

/*
 * Sets the value of the given variable in the environment.  If the
variable
 * already exists, its value is overriden.
 */
void
child_set_env(char ***envp, u_int *envsizep, const char *name,
	const char *value)
{
	char **env;
	u_int envsize;
	u_int i, namelen;

	/*
	 * If we're passed an uninitialized list, allocate a single null
	 * entry before continuing.
	 */
	if (*envp == NULL && *envsizep == 0) {
		*envp = xmalloc(sizeof(char *));
		*envp[0] = NULL;
		*envsizep = 1;
	}

	/*
	 * Find the slot where the value should be stored.  If the
variable
	 * already exists, we reuse the slot; otherwise we append a new
slot
	 * at the end of the array, expanding if necessary.
	 */
	env = *envp;
	namelen = strlen(name);
	for (i = 0; env[i]; i++)
		if (strncmp(env[i], name, namelen) == 0 && env[i][namelen]
== '=')
			break;
	if (env[i]) {
		/* Reuse the slot. */
		xfree(env[i]);
```

```
	} else {
		/* New variable.  Expand if necessary. */
		envsize = *envsizep;
		if (i >= envsize - 1) {
			if (envsize >= 1000)
				fatal("child_set_env: too many env vars");
			envsize += 50;
			env = (*envp) = xrealloc(env, envsize * sizeof(char
*));
			*envsizep = envsize;
		}
		/* Need to set the NULL pointer at end of array beyond the
new slot. */
		env[i + 1] = NULL;
	}

	/* Allocate space and format the variable in the appropriate
slot. */
	env[i] = xmalloc(strlen(name) + 1 + strlen(value) + 1);
	snprintf(env[i], strlen(name) + 1 + strlen(value) + 1, "%s=%s",
name, value);
}

/*
 * Reads environment variables from the given file and adds/overrides
them
 * into the environment.   If the file does not exist, this does
nothing.
 * Otherwise, it must consist of empty lines, comments (line starts
with '#')
 * and assignments of the form name=value.  No other forms are allowed.
 */
static void
read_environment_file(char ***env, u_int *envsize,
	const char *filename)
{
	FILE *f;
	char buf[4096];
	char *cp, *value;
	u_int lineno = 0;

	f = fopen(filename, "r");
	if (!f)
		return;

	while (fgets(buf, sizeof(buf), f)) {
		if (++lineno > 1000)
			fatal("Too  many  lines  in  environment  file  %s",
filename);
		for (cp = buf; *cp == ' ' || *cp == '\t'; cp++)
			;
		if (!*cp || *cp == '#' || *cp == '\n')
			continue;
		if (strchr(cp, '\n'))
			*strchr(cp, '\n') = '\0';
		value = strchr(cp, '=');
		if (value == NULL) {
```

```
                fprintf(stderr, "Bad line %u in %.100s\n", lineno,
                    filename);
                continue;
            }
            /*
             * Replace the equals sign by nul, and advance value to
             * the value string.
             */
            *value = '\0';
            value++;
            child_set_env(env, envsize, cp, value);
      }
      fclose(f);
}

#ifdef HAVE_ETC_DEFAULT_LOGIN
/*
 * Return named variable from specified environment, or NULL if not
present.
 */
static char *
child_get_env(char **env, const char *name)
{
      int i;
      size_t len;

      len = strlen(name);
      for (i=0; env[i] != NULL; i++)
            if (strncmp(name, env[i], len) == 0 && env[i][len] == '=')
                  return(env[i] + len + 1);
      return NULL;
}

/*
 * Read /etc/default/login.
 * We pick up the PATH (or SUPATH for root) and UMASK.
 */
static void
read_etc_default_login(char ***env, u_int *envsize, uid_t uid)
{
      char **tmpenv = NULL, *var;
      u_int i, tmpenvsize = 0;
      mode_t mask;

      /*
       *  We  don't  want  to  copy  the  whole  file  to  the  child's
environment,
       * so we use a temporary environment and copy the variables we're
       * interested in.
       */
      read_environment_file(&tmpenv,                        &tmpenvsize,
"/etc/default/login");

      if (tmpenv == NULL)
            return;

      if (uid == 0)
```

```
            var = child_get_env(tmpenv, "SUPATH");
      else
            var = child_get_env(tmpenv, "PATH");
      if (var != NULL)
            child_set_env(env, envsize, "PATH", var);

      if ((var = child_get_env(tmpenv, "UMASK")) != NULL)
            if (sscanf(var, "%5lo", &mask) == 1)
                  umask(mask);

      for (i = 0; tmpenv[i] != NULL; i++)
            xfree(tmpenv[i]);
      xfree(tmpenv);
}
#endif /* HAVE_ETC_DEFAULT_LOGIN */

void copy_environment(char **source, char ***env, u_int *envsize)
{
      char *var_name, *var_val;
      int i;

      if (source == NULL)
            return;

      for(i = 0; source[i] != NULL; i++) {
            var_name = xstrdup(source[i]);
            if ((var_val = strstr(var_name, "=")) == NULL) {
                  xfree(var_name);
                  continue;
            }
            *var_val++ = '\0';

            debug3("Copy environment: %s=%s", var_name, var_val);
            child_set_env(env, envsize, var_name, var_val);

            xfree(var_name);
      }
}

static char **
do_setup_env(Session *s, const char *shell)
{
      char buf[256];
      u_int i, envsize;
      char **env, *laddr, *path = NULL;
      struct passwd *pw = s->pw;

      /* Initialize the environment. */
      envsize = 100;
      env = xmalloc(envsize * sizeof(char *));
      env[0] = NULL;

#ifdef HAVE_CYGWIN
      /*
       * The Windows environment contains some setting which are
       * important for a running system. They must not be dropped.
       */
```

```
      copy_environment(environ, &env, &envsize);
#endif

#ifdef GSSAPI
      /* Allow any GSSAPI methods that we've used to alter
       * the childs environment as they see fit
       */
      ssh_gssapi_do_child(&env, &envsize);
#endif

      if (!options.use_login) {
            /* Set basic environment. */
            child_set_env(&env, &envsize, "USER", pw->pw_name);
            child_set_env(&env, &envsize, "LOGNAME", pw->pw_name);
#ifdef _AIX
            child_set_env(&env, &envsize, "LOGIN", pw->pw_name);
#endif
            child_set_env(&env, &envsize, "HOME", pw->pw_dir);
#ifdef HAVE_LOGIN_CAP
            if (setusercontext(lc, pw, pw->pw_uid, LOGIN_SETPATH) < 0)
                  child_set_env(&env, &envsize, "PATH", _PATH_STDPATH);
            else
                  child_set_env(&env,         &envsize,         "PATH",
getenv("PATH"));
#else /* HAVE_LOGIN_CAP */
# ifndef HAVE_CYGWIN
            /*
             * There's no standard path on Windows. The path contains
             * important components pointing to the system directories,
             * needed for loading shared libraries. So the path better
             * remains intact here.
             */
#  ifdef HAVE_ETC_DEFAULT_LOGIN
            read_etc_default_login(&env, &envsize, pw->pw_uid);
            path = child_get_env(env, "PATH");
#  endif /* HAVE_ETC_DEFAULT_LOGIN */
            if (path == NULL || *path == '\0') {
                  child_set_env(&env, &envsize, "PATH",
                      s->pw->pw_uid == 0 ?
                        SUPERUSER_PATH : _PATH_STDPATH);
            }
# endif /* HAVE_CYGWIN */
#endif /* HAVE_LOGIN_CAP */

            snprintf(buf, sizeof buf, "%.200s/%.50s",
                   _PATH_MAILDIR, pw->pw_name);
            child_set_env(&env, &envsize, "MAIL", buf);

            /* Normal systems set SHELL by default. */
            child_set_env(&env, &envsize, "SHELL", shell);
      }
      if (getenv("TZ"))
            child_set_env(&env, &envsize, "TZ", getenv("TZ"));

      /* Set custom environment options from RSA authentication. */
      if (!options.use_login) {
            while (custom_environment) {
```

```
                struct envstring *ce = custom_environment;
                char *str = ce->s;

                for (i = 0; str[i] != '=' && str[i]; i++)
                      ;
                if (str[i] == '=') {
                      str[i] = 0;
                      child_set_env(&env,  &envsize,  str,  str  +  i  +
1);
                }
                custom_environment = ce->next;
                xfree(ce->s);
                xfree(ce);
          }
      }

      /* SSH_CLIENT deprecated */
      snprintf(buf, sizeof buf, "%.50s %d %d",
          get_remote_ipaddr(), get_remote_port(), get_local_port());
      child_set_env(&env, &envsize, "SSH_CLIENT", buf);

      laddr = get_local_ipaddr(packet_get_connection_in());
      snprintf(buf, sizeof buf, "%.50s %d %.50s %d",
          get_remote_ipaddr(),         get_remote_port(),         laddr,
get_local_port());
      xfree(laddr);
      child_set_env(&env, &envsize, "SSH_CONNECTION", buf);

      if (s->ttyfd != -1)
            child_set_env(&env, &envsize, "SSH_TTY", s->tty);
      if (s->term)
            child_set_env(&env, &envsize, "TERM", s->term);
      if (s->display)
            child_set_env(&env, &envsize, "DISPLAY", s->display);
      if (original_command)
            child_set_env(&env, &envsize, "SSH_ORIGINAL_COMMAND",
                original_command);

#ifdef _UNICOS
      if (cray_tmpdir[0] != '\0')
            child_set_env(&env, &envsize, "TMPDIR", cray_tmpdir);
#endif /* _UNICOS */

#ifdef _AIX
      {
            char *cp;

            if ((cp = getenv("AUTHSTATE")) != NULL)
                  child_set_env(&env, &envsize, "AUTHSTATE", cp);
            if ((cp = getenv("KRB5CCNAME")) != NULL)
                  child_set_env(&env, &envsize, "KRB5CCNAME", cp);
            read_environment_file(&env, &envsize, "/etc/environment");
      }
#endif
#ifdef KRB5
      if (s->authctxt->krb5_ticket_file)
            child_set_env(&env, &envsize, "KRB5CCNAME",
```

```
                s->authctxt->krb5_ticket_file);
#endif
#ifdef USE_PAM
      /*
       * Pull in any environment variables that may have
       * been set by PAM.
       */
      if (options.use_pam) {
            char **p = fetch_pam_environment();

            copy_environment(p, &env, &envsize);
            free_pam_environment(p);
      }
#endif /* USE_PAM */

      if (auth_sock_name != NULL)
            child_set_env(&env, &envsize, SSH_AUTHSOCKET_ENV_NAME,
                auth_sock_name);

      /* read $HOME/.ssh/environment. */
      if (options.permit_user_env && !options.use_login) {
            snprintf(buf, sizeof buf, "%.200s/.ssh/environment",
                strcmp(pw->pw_dir, "/") ? pw->pw_dir : "");
            read_environment_file(&env, &envsize, buf);
      }
      if (debug_flag) {
            /* dump the environment */
            fprintf(stderr, "Environment:\n");
            for (i = 0; env[i]; i++)
                  fprintf(stderr, "  %.200s\n", env[i]);
      }
      return env;
}

/*
 * Run $HOME/.ssh/rc, /etc/ssh/sshrc, or xauth (whichever is found
 * first in this order).
 */
static void
do_rc_files(Session *s, const char *shell)
{
      FILE *f = NULL;
      char cmd[1024];
      int do_xauth;
      struct stat st;

      do_xauth =
          s->display != NULL && s->auth_proto != NULL && s->auth_data
!= NULL;

      /* ignore _PATH_SSH_USER_RC for subsystems */
      if (!s->is_subsystem && (stat(_PATH_SSH_USER_RC, &st) >= 0)) {
            snprintf(cmd, sizeof cmd, "%s -c '%s %s'",
                shell, _PATH_BSHELL, _PATH_SSH_USER_RC);
            if (debug_flag)
                  fprintf(stderr, "Running %s\n", cmd);
            f = popen(cmd, "w");
```

```
            if (f) {
                  if (do_xauth)
                        fprintf(f, "%s %s\n", s->auth_proto,
                            s->auth_data);
                  pclose(f);
            } else
                  fprintf(stderr, "Could not run %s\n",
                      _PATH_SSH_USER_RC);
      } else if (stat(_PATH_SSH_SYSTEM_RC, &st) >= 0) {
            if (debug_flag)
                  fprintf(stderr, "Running %s %s\n", _PATH_BSHELL,
                      _PATH_SSH_SYSTEM_RC);
            f = popen(_PATH_BSHELL " " _PATH_SSH_SYSTEM_RC, "w");
            if (f) {
                  if (do_xauth)
                        fprintf(f, "%s %s\n", s->auth_proto,
                            s->auth_data);
                  pclose(f);
            } else
                  fprintf(stderr, "Could not run %s\n",
                      _PATH_SSH_SYSTEM_RC);
      } else if (do_xauth && options.xauth_location != NULL) {
            /* Add authority data to .Xauthority if appropriate. */
            if (debug_flag) {
                  fprintf(stderr,
                      "Running %.500s remove %.100s\n",
                      options.xauth_location, s->auth_display);
                  fprintf(stderr,
                      "%.500s add %.100s %.100s %.100s\n",
                      options.xauth_location, s->auth_display,
                      s->auth_proto, s->auth_data);
            }
            snprintf(cmd, sizeof cmd, "%s -q -",
                options.xauth_location);
            f = popen(cmd, "w");
            if (f) {
                  fprintf(f, "remove %s\n",
                      s->auth_display);
                  fprintf(f, "add %s %s %s\n",
                      s->auth_display, s->auth_proto,
                      s->auth_data);
                  pclose(f);
            } else {
                  fprintf(stderr, "Could not run %s\n",
                      cmd);
            }
      }
}

static void
do_nologin(struct passwd *pw)
{
      FILE *f = NULL;
      char buf[1024];

#ifdef HAVE_LOGIN_CAP
      if (!login_getcapbool(lc, "ignorenologin", 0) && pw->pw_uid)
```

```
            f = fopen(login_getcapstr(lc, "nologin", _PATH_NOLOGIN,
                _PATH_NOLOGIN), "r");
#else
      if (pw->pw_uid)
            f = fopen(_PATH_NOLOGIN, "r");
#endif
      if (f) {
            /* /etc/nologin exists.  Print its contents and exit. */
            logit("User %.100s not allowed because %s exists",
                pw->pw_name, _PATH_NOLOGIN);
            while (fgets(buf, sizeof(buf), f))
                  fputs(buf, stderr);
            fclose(f);
            fflush(NULL);
            exit(254);
      }
}

/* Set login name, uid, gid, and groups. */
void
do_setusercontext(struct passwd *pw)
{
      //MYSEA: Change these numbers to the UID of the user the daemon
      //will run as.
#ifndef HAVE_CYGWIN
      if (getuid() == 3 || geteuid() == 3)
#endif /* HAVE_CYGWIN */
      {

#ifdef HAVE_SETPCRED
            if (setpcred(pw->pw_name, (char **)NULL) == -1)
            {
                  fatal("Failed to set process credentials");
            }
#endif /* HAVE_SETPCRED */
#ifdef HAVE_LOGIN_CAP
# ifdef __bsdi__
            setpgid(0, 0);
# endif
            if (setusercontext(lc, pw, pw->pw_uid,
                (LOGIN_SETALL & ~LOGIN_SETPATH)) < 0) {
                  perror("unable to set user context");
                  exit(1);
            }
#else
# if defined(HAVE_GETLUID) && defined(HAVE_SETLUID)
            /* Sets login uid for accounting */
            if (getluid() == -1 && setluid(pw->pw_uid) == -1)
            {
                  error("setluid: %s", strerror(errno));
            }
# endif /* defined(HAVE_GETLUID) && defined(HAVE_SETLUID) */

            if (setlogin(pw->pw_name) < 0)
            {
                  error("setlogin failed: %s", strerror(errno));
            }
```

```
            if (setgid(pw->pw_gid) < 0) {
                  perror("setgid");
                  exit(1);
            }
            //MYSEA: initgroups is not implemented on the XTS-400,
            //for now, it has been commented out.
            /* Initialize the group list. */
            //if (initgroups(pw->pw_name, pw->pw_gid) < 0) {
            //    perror("initgroups");
            //    exit(1);
            //}
            endgrent();
# ifdef USE_PAM
            /*
             *  PAM  credentials  may  take  the  form  of  supplementary
groups.
             *  These  will  have  been  wiped  by  the  above  initgroups()
call.
             * Reestablish them here.
             */
            if (options.use_pam) {
                  do_pam_session();
                  do_pam_setcred(0);
            }
# endif /* USE_PAM */
#   if   defined(WITH_IRIX_PROJECT)   ||   defined(WITH_IRIX_JOBS)   ||
defined(WITH_IRIX_ARRAY)
            irix_setusercontext(pw);
#   endif  /*  defined(WITH_IRIX_PROJECT)  ||  defined(WITH_IRIX_JOBS)  ||
defined(WITH_IRIX_ARRAY) */
# ifdef _AIX
            aix_usrinfo(pw);
# endif /* _AIX */
            /* Permanently switch to the desired uid. */
            permanently_set_uid(pw);
#endif
      }

#ifdef HAVE_CYGWIN
      if (is_winnt)
#endif
      if (getuid() != pw->pw_uid || geteuid() != pw->pw_uid)
            fatal("Failed to set uids to %u.", (u_int) pw->pw_uid);
}

static void
launch_login(struct passwd *pw, const char *hostname)
{
      /* Launch login(1). */

      execl(LOGIN_PROGRAM, "login", "-h", hostname,
#ifdef xxxLOGIN_NEEDS_TERM
                (s->term ? s->term : "unknown"),
#endif /* LOGIN_NEEDS_TERM */
#ifdef LOGIN_NO_ENDOPT
           "-p", "-f", pw->pw_name, (char *)NULL);
#else
```

```
          "-p", "-f", "--", pw->pw_name, (char *)NULL);
#endif

      /* Login couldn't be executed, die. */

      perror("login");
      exit(1);
}

/*
 * Performs common processing for the child, such as setting up the
 *  environment,  closing  extra  file  descriptors,  setting  the  user  and
group
 * ids, and executing the command or shell.
 */
void
do_child(Session *s, const char *command)
{
      extern char **environ;
      char **env;
      char *argv[10];
      const char *shell, *shell0, *hostname = NULL;
      struct passwd *pw = s->pw;
      u_int i;
      /* remove hostkey from the child's memory */
      destroy_sensitive_data();
      /* login(1) is only called if we execute the login shell */
      if (options.use_login && command != NULL)
            options.use_login = 0;

#ifdef _UNICOS
      cray_setup(pw->pw_uid, pw->pw_name, command);
#endif /* _UNICOS */

      /*
       * Login(1) does this as well, and it needs uid 0 for the "-h"
       * switch, so we let login(1) to this for us.
       */
      if (!options.use_login) {
#ifdef HAVE_OSF_SIA
            session_setup_sia(pw, s->ttyfd == -1 ? NULL : s->tty);
            if (!check_quietlogin(s, command))
                  do_motd();
#else /* HAVE_OSF_SIA */
            do_nologin(pw);
            do_setusercontext(pw);
#endif /* HAVE_OSF_SIA */
      }

      /*
       * Get the shell from the password data.  An empty shell field is
       * legal, and means /bin/sh.
       */
      shell = (pw->pw_shell[0] == '\0') ? _PATH_BSHELL : pw->pw_shell;

      /*
       * Make sure $SHELL points to the shell from the password file,
```

```
	 * even if shell is overridden from login.conf
	 */
	env = do_setup_env(s, shell);

#ifdef HAVE_LOGIN_CAP
	shell = login_getcapstr(lc, "shell", (char *)shell, (char
*)shell);
#endif

	/* we have to stash the hostname before we close our socket. */
	if (options.use_login)
		hostname = get_remote_name_or_ip(utmp_len,
		    options.use_dns);
	/*
	 * Close the connection descriptors; note that this is the child,
and
	 * the server will still have the socket open, and it is
important
	 * that we do not shutdown it.  Note that the descriptors cannot
be
	 * closed before building the environment, as we call
	 * get_remote_ipaddr there.
	 */
	if (packet_get_connection_in() == packet_get_connection_out())
		close(packet_get_connection_in());
	else {
		close(packet_get_connection_in());
		close(packet_get_connection_out());
	}
	/*
	 * Close all descriptors related to channels.  They will still
remain
	 * open in the parent.
	 */
	/* XXX better use close-on-exec? -markus */
	channel_close_all();

	/*
	 * Close any extra file descriptors.  Note that there may still
be
	 * descriptors left by system functions.  They will be closed
later.
	 */
	endpwent();

	/*
	 * Close any extra open file descriptors so that we don\'t have
them
	 * hanging around in clients.  Note that we want to do this after
	 * initgroups, because at least on Solaris 2.3 it leaves file
	 * descriptors open.
	 */
	for (i = 3; i < 64; i++)
		close(i);

	/*
	 * Must take new environment into use so that .ssh/rc,
```

```
	 * /etc/ssh/sshrc and xauth are run in the proper environment.
	 */
	environ = env;

	/* Change current directory to the user\'s home directory. */
	if (chdir(pw->pw_dir) < 0) {
		fprintf(stderr, "Could not chdir to home directory %s: %s\n",
		    pw->pw_dir, strerror(errno));
#ifdef HAVE_LOGIN_CAP
		if (login_getcapbool(lc, "requirehome", 0))
			exit(1);
#endif
	}

	if (!options.use_login)
		do_rc_files(s, shell);

	/* restore SIGPIPE for child */
	signal(SIGPIPE, SIG_DFL);

	if (options.use_login) {
		launch_login(pw, hostname);
		/* NEVERREACHED */
	}

	/* Get the last component of the shell name. */
	if ((shell0 = strrchr(shell, '/')) != NULL)
		shell0++;
	else
		shell0 = shell;

	/*
	 * If we have no command, execute the shell.  In this case, the shell
	 * name to be passed in argv[0] is preceded by '-' to indicate that
	 * this is a login shell.
	 */
	if (!command) {
		char argv0[256];

		/* Start the shell.  Set initial character to '-'. */
		argv0[0] = '-';

		if (strlcpy(argv0 + 1, shell0, sizeof(argv0) - 1)
		    >= sizeof(argv0) - 1) {
			errno = EINVAL;
			perror(shell);
			exit(1);
		}

		/* Execute the shell. */
		argv[0] = argv0;
		argv[1] = NULL;
		execve(shell, argv, env);
		/* Executing the shell failed. */
```

```
            perror(shell);
            exit(1);
      }
      /*
       * Execute the command using the user's shell.  This uses the -c
       * option to execute the command.
       */
      argv[0] = (char *) shell0;
      argv[1] = "-c";
      argv[2] = (char *) command;
      argv[3] = NULL;
      execve(shell, argv, env);
      perror(shell);
      exit(1);
}

Session *
session_new(void)
{
      int i;
      static int did_init = 0;
      if (!did_init) {
            debug("session_new: init");
            for (i = 0; i < MAX_SESSIONS; i++) {
                  sessions[i].used = 0;
            }
            did_init = 1;
      }
      for (i = 0; i < MAX_SESSIONS; i++) {
            Session *s = &sessions[i];
            if (! s->used) {
                  memset(s, 0, sizeof(*s));
                  s->chanid = -1;
                  s->ptyfd = -1;
                  s->ttyfd = -1;
                  s->used = 1;
                  s->self = i;
                  debug("session_new: session %d", i);
                  return s;
            }
      }
      return NULL;
}

static void
session_dump(void)
{
      int i;
      for (i = 0; i < MAX_SESSIONS; i++) {
            Session *s = &sessions[i];
            debug("dump: used %d session %d %p channel %d pid %ld",
                s->used,
                s->self,
                s,
                s->chanid,
                (long)s->pid);
      }
```

```
}

int
session_open(Authctxt *authctxt, int chanid)
{
	Session *s = session_new();
	debug("session_open: channel %d", chanid);
	if (s == NULL) {
		error("no more sessions");
		return 0;
	}
	s->authctxt = authctxt;
	s->pw = authctxt->pw;
	if (s->pw == NULL)
		fatal("no user for session %d", s->self);
	debug("session_open: session %d: link with channel %d", s->self,
chanid);
	s->chanid = chanid;
	return 1;
}

Session *
session_by_tty(char *tty)
{
	int i;
	for (i = 0; i < MAX_SESSIONS; i++) {
		Session *s = &sessions[i];
		if (s->used && s->ttyfd != -1 && strcmp(s->tty, tty) == 0)
{
			debug("session_by_tty: session %d tty %s", i, tty);
			return s;
		}
	}
	debug("session_by_tty: unknown tty %.100s", tty);
	session_dump();
	return NULL;
}

static Session *
session_by_channel(int id)
{
	int i;
	for (i = 0; i < MAX_SESSIONS; i++) {
		Session *s = &sessions[i];
		if (s->used && s->chanid == id) {
			debug("session_by_channel: session %d channel %d", i,
id);
			return s;
		}
	}
	debug("session_by_channel: unknown channel %d", id);
	session_dump();
	return NULL;
}

static Session *
session_by_pid(pid_t pid)
```

```
{
      int i;
      debug("session_by_pid: pid %ld", (long)pid);
      for (i = 0; i < MAX_SESSIONS; i++) {
            Session *s = &sessions[i];
            if (s->used && s->pid == pid)
                  return s;
      }
      error("session_by_pid: unknown pid %ld", (long)pid);
      session_dump();
      return NULL;
}

static int
session_window_change_req(Session *s)
{
      s->col = packet_get_int();
      s->row = packet_get_int();
      s->xpixel = packet_get_int();
      s->ypixel = packet_get_int();
      packet_check_eom();
      pty_change_window_size(s->ptyfd,  s->row,  s->col,  s->xpixel,  s-
>ypixel);
      return 1;
}

static int
session_pty_req(Session *s)
{
      u_int len;
      int n_bytes;
      if (no_pty_flag) {
            debug("Allocating   a   pty   not   permitted   for   this
authentication.");
            return 0;
      }
      if (s->ttyfd != -1) {
            packet_disconnect("Protocol  error:  you  already  have  a
pty.");
            return 0;
      }
      /* Get the time and hostname when the user last logged in. */

      if (options.print_lastlog) {
            s->hostname[0] = '\0';
            s->last_login_time = get_last_login_time(s->pw->pw_uid,
                s->pw->pw_name, s->hostname, sizeof(s->hostname));
      }

      s->term = packet_get_string(&len);

      if (compat20) {
            s->col = packet_get_int();
            s->row = packet_get_int();
      } else {
            s->row = packet_get_int();
            s->col = packet_get_int();
```

```
	}
	s->xpixel = packet_get_int();
	s->ypixel = packet_get_int();

	if (strcmp(s->term, "") == 0) {
		xfree(s->term);
		s->term = NULL;
	}

	/* Allocate a pty and open it. */
	debug("Allocating pty.");
	if (!PRIVSEP(pty_allocate(&s->ptyfd, &s->ttyfd, s->tty, sizeof(s-
>tty)))) {
		if (s->term)
			xfree(s->term);
		s->term = NULL;
		s->ptyfd = -1;
		s->ttyfd = -1;
		error("session_pty_req: session %d alloc failed", s->self);
		return 0;
	}
	debug("session_pty_req: session %d alloc %s", s->self, s->tty);
	/* for SSH1 the tty modes length is not given */
	if (!compat20)
		n_bytes = packet_remaining();
	tty_parse_modes(s->ttyfd, &n_bytes);

	/*
	 * Add a cleanup function to clear the utmp entry and record
logout
	 * time in case we call fatal() (e.g., the connection gets
closed).
	 */
	fatal_add_cleanup(session_pty_cleanup, (void *)s);
	if (!use_privsep)
		pty_setowner(s->pw, s->tty);

	/* Set window size from the packet. */
	pty_change_window_size(s->ptyfd, s->row, s->col, s->xpixel, s-
>ypixel);

	packet_check_eom();
	session_proctitle(s);
	return 1;
}

static int
session_subsystem_req(Session *s)
{
	struct stat st;
	u_int len;
	int success = 0;
	char *cmd, *subsys = packet_get_string(&len);
	int i;

	packet_check_eom();
	logit("subsystem request for %.100s", subsys);
```

```
	for (i = 0; i < options.num_subsystems; i++) {
		if (strcmp(subsys, options.subsystem_name[i]) == 0) {
			cmd = options.subsystem_command[i];
			if (stat(cmd, &st) < 0) {
				error("subsystem: cannot stat %s: %s", cmd,
				    strerror(errno));
				break;
			}
			debug("subsystem: exec() %s", cmd);
			s->is_subsystem = 1;
			do_exec(s, cmd);
			success = 1;
			break;
		}
	}

	if (!success)
		logit("subsystem request for %.100s failed, subsystem not
found",
		    subsys);

	xfree(subsys);
	return success;
}

static int
session_x11_req(Session *s)
{
	int success;

	s->single_connection = packet_get_char();
	s->auth_proto = packet_get_string(NULL);
	s->auth_data = packet_get_string(NULL);
	s->screen = packet_get_int();
	packet_check_eom();

	success = session_setup_x11fwd(s);
	if (!success) {
		xfree(s->auth_proto);
		xfree(s->auth_data);
		s->auth_proto = NULL;
		s->auth_data = NULL;
	}
	return success;
}

static int
session_shell_req(Session *s)
{
	packet_check_eom();
	do_exec(s, NULL);
	return 1;
}

static int
session_exec_req(Session *s)
```

```
{
      u_int len;
      char *command = packet_get_string(&len);
      packet_check_eom();
      do_exec(s, command);
      xfree(command);
      return 1;
}

static int
session_break_req(Session *s)
{
      u_int break_length;

      break_length = packet_get_int();    /* ignored */
      packet_check_eom();

      if (s->ttyfd == -1 ||
          tcsendbreak(s->ttyfd, 0) < 0)
            return 0;
      return 1;
}

static int
session_auth_agent_req(Session *s)
{
      static int called = 0;
      packet_check_eom();
      if (no_agent_forwarding_flag) {
            debug("session_auth_agent_req: no_agent_forwarding_flag");
            return 0;
      }
      if (called) {
            return 0;
      } else {
            called = 1;
            return auth_input_request_forwarding(s->pw);
      }
}

int
session_input_channel_req(Channel *c, const char *rtype)
{
      int success = 0;
      Session *s;
      if ((s = session_by_channel(c->self)) == NULL) {
            logit("session_input_channel_req:   no   session   %d   req
%.100s",
                c->self, rtype);
            return 0;
      }
      debug("session_input_channel_req:  session  %d  req  %s",  s->self,
rtype);
      /*
       * a session is in LARVAL state until a shell, a command
       * or a subsystem is executed
       */
```

```
      if (c->type == SSH_CHANNEL_LARVAL) {
            if (strcmp(rtype, "shell") == 0) {
                  success = session_shell_req(s);
            } else if (strcmp(rtype, "exec") == 0) {
                  success = session_exec_req(s);
            } else if (strcmp(rtype, "pty-req") == 0) {
                  success =  session_pty_req(s);
            } else if (strcmp(rtype, "x11-req") == 0) {
                  success = session_x11_req(s);
            } else if (strcmp(rtype, "auth-agent-req@openssh.com") ==
0) {
                  success = session_auth_agent_req(s);
            } else if (strcmp(rtype, "subsystem") == 0) {
                  success = session_subsystem_req(s);
            } else if (strcmp(rtype, "break") == 0) {
                  success = session_break_req(s);
            }
      }
      if (strcmp(rtype, "window-change") == 0) {
            success = session_window_change_req(s);
      }
      return success;
}

void
session_set_fds(Session *s, int fdin, int fdout, int fderr)
{
      if (!compat20)
            fatal("session_set_fds: called for proto != 2.0");
      /*
       * now that have a child and a pipe to the child,
       * we can activate our channel and register the fd's
       */
      if (s->chanid == -1)
            fatal("no channel for session %d", s->self);
      channel_set_fds(s->chanid,
          fdout, fdin, fderr,
          fderr == -1 ? CHAN_EXTENDED_IGNORE : CHAN_EXTENDED_READ,
          1,
          CHAN_SES_WINDOW_DEFAULT);
}

/*
 *  Function  to  perform  pty  cleanup.  Also  called  if  we  get  aborted
abnormally
 * (e.g., due to a dropped connection).
 */
void
session_pty_cleanup2(void *session)
{
      Session *s = session;

      if (s == NULL) {
            error("session_pty_cleanup: no session");
            return;
      }
      if (s->ttyfd == -1)
```

```
            return;

      debug("session_pty_cleanup: session %d release %s", s->self, s-
>tty);

      /* Record that the user has logged out. */
      if (s->pid != 0)
            record_logout(s->pid, s->tty, s->pw->pw_name);

      /* Release the pseudo-tty. */
      debug("Going to Release PTY\n");
      if (getuid() == 0)
            pty_release(s->tty);
      debug("Released the PTY\n");

      /*
       * Close the server side of the socket pairs.  We must do this
after
       * the pty cleanup, so that another process doesn't get this pty
       * while we're still cleaning up.
       */
      if (close(s->ptymaster) < 0)
            error("close(s->ptymaster/%d):      %s",      s->ptymaster,
strerror(errno));

      /* unlink pty from session */
      s->ttyfd = -1;
      debug("end of pty_cleanup function");
}

void
session_pty_cleanup(void *session)
{
      PRIVSEP(session_pty_cleanup2(session));
}

static char *
sig2name(int sig)
{
#define SSH_SIG(x) if (sig == SIG ## x) return #x
      SSH_SIG(ABRT);
      SSH_SIG(ALRM);
      SSH_SIG(FPE);
      SSH_SIG(HUP);
      SSH_SIG(ILL);
      SSH_SIG(INT);
      SSH_SIG(KILL);
      SSH_SIG(PIPE);
      SSH_SIG(QUIT);
      SSH_SIG(SEGV);
      SSH_SIG(TERM);
      SSH_SIG(USR1);
      SSH_SIG(USR2);
#undef      SSH_SIG
      return "SIG@openssh.com";
}
```

```
static void
session_exit_message(Session *s, int status)
{
      Channel *c;

      if ((c = channel_lookup(s->chanid)) == NULL)
            fatal("session_exit_message: session %d: no channel %d",
                s->self, s->chanid);
      debug("session_exit_message: session %d channel %d pid %ld",
          s->self, s->chanid, (long)s->pid);

      if (WIFEXITED(status)) {
            channel_request_start(s->chanid, "exit-status", 0);
            packet_put_int(WEXITSTATUS(status));
            packet_send();
      } else if (WIFSIGNALED(status)) {
            channel_request_start(s->chanid, "exit-signal", 0);
            packet_put_cstring(sig2name(WTERMSIG(status)));
#ifdef WCOREDUMP
            packet_put_char(WCOREDUMP(status));
#else /* WCOREDUMP */
            packet_put_char(0);
#endif /* WCOREDUMP */
            packet_put_cstring("");
            packet_put_cstring("");
            packet_send();
      } else {
            /* Some weird exit cause.  Just exit. */
            packet_disconnect("wait returned status %04x.", status);
      }

      /* disconnect channel */
      debug("session_exit_message: release channel %d", s->chanid);
      channel_cancel_cleanup(s->chanid);
      /*
       * emulate a write failure with 'chan_write_failed', nobody will
be
       * interested in data we write.
       *  Note  that  we  must  not  call  'chan_read_failed',  since  there
could
       * be some more data waiting in the pipe.
       */
      if (c->ostate != CHAN_OUTPUT_CLOSED)
            chan_write_failed(c);
      s->chanid = -1;
}

void
session_close(Session *s)
{
      debug("session_close:  session  %d  pid  %ld",  s->self,  (long)s-
>pid);
      debug("Starting Session close");
      if (s->ttyfd != -1) {
            fatal_remove_cleanup(session_pty_cleanup, (void *)s);
            session_pty_cleanup(s);
      }
```

```
	if (s->term)
		xfree(s->term);
	if (s->display)
		xfree(s->display);
	if (s->auth_display)
		xfree(s->auth_display);
	if (s->auth_data)
		xfree(s->auth_data);
	if (s->auth_proto)
		xfree(s->auth_proto);
	s->used = 0;
	session_proctitle(s);
	debug("ending session close");
}

void
session_close_by_pid(pid_t pid, int status)
{
	Session *s = session_by_pid(pid);
	if (s == NULL) {
		debug("session_close_by_pid: no session for pid %ld",
		    (long)pid);
		return;
	}
	if (s->chanid != -1)
		session_exit_message(s, status);
	session_close(s);
}

/*
 * this is called when a channel dies before
 * the session 'child' itself dies
 */
void
session_close_by_channel(int id, void *arg)
{
	Session *s = session_by_channel(id);
	if (s == NULL) {
		debug("session_close_by_channel:  no  session  for  id  %d",
id);
		return;
	}
	debug("session_close_by_channel: channel %d child %ld",
	    id, (long)s->pid);
	if (s->pid != 0) {
		debug("session_close_by_channel:  channel  %d:  has  child",
id);
		/*
		 * delay detach of session, but release pty, since
		 * the fd's to the child are already closed
		 */
		if (s->ttyfd != -1) {
			fatal_remove_cleanup(session_pty_cleanup, (void *)s);
			session_pty_cleanup(s);
		}
		return;
	}
```

```
	/* detach by removing callback */
	channel_cancel_cleanup(s->chanid);
	s->chanid = -1;
	session_close(s);
}

void
session_destroy_all(void (*closefunc)(Session *))
{
	int i;
	for (i = 0; i < MAX_SESSIONS; i++) {
		Session *s = &sessions[i];
		if (s->used) {
			if (closefunc != NULL)
				closefunc(s);
			else
				session_close(s);
		}
	}
}

static char *
session_tty_list(void)
{
	static char buf[1024];
	int i;
	char *cp;

	buf[0] = '\0';
	for (i = 0; i < MAX_SESSIONS; i++) {
		Session *s = &sessions[i];
		if (s->used && s->ttyfd != -1) {

			if (strncmp(s->tty, "/dev/", 5) != 0) {
				cp = strrchr(s->tty, '/');
				cp = (cp == NULL) ? s->tty : cp + 1;
			} else
				cp = s->tty + 5;

			if (buf[0] != '\0')
				strlcat(buf, ",", sizeof buf);
			strlcat(buf, cp, sizeof buf);
		}
	}
	if (buf[0] == '\0')
		strlcpy(buf, "notty", sizeof buf);
	return buf;
}

void
session_proctitle(Session *s)
{
	if (s->pw == NULL)
		error("no user for session %d", s->self);
	else
		setproctitle("%s@%s", s->pw->pw_name, session_tty_list());
}
```

```
int
session_setup_x11fwd(Session *s)
{
      struct stat st;
      char display[512], auth_display[512];
      char hostname[MAXHOSTNAMELEN];

      if (no_x11_forwarding_flag) {
            packet_send_debug("X11   forwarding   disabled   in   user
configuration file.");
            return 0;
      }
      if (!options.x11_forwarding) {
            debug("X11  forwarding  disabled  in  server  configuration
file.");
            return 0;
      }
      if (!options.xauth_location ||
          (stat(options.xauth_location, &st) == -1)) {
            packet_send_debug("No  xauth  program;  cannot  forward  with
spoofing.");
            return 0;
      }
      if (options.use_login) {
            packet_send_debug("X11 forwarding disabled; "
                "not compatible with UseLogin=yes.");
            return 0;
      }
      if (s->display != NULL) {
            debug("X11 display already set.");
            return 0;
      }
      if (x11_create_display_inet(options.x11_display_offset,
          options.x11_use_localhost, s->single_connection,
          &s->display_number) == -1) {
            debug("x11_create_display_inet failed.");
            return 0;
      }

      /* Set up a suitable value for the DISPLAY variable. */
      if (gethostname(hostname, sizeof(hostname)) < 0)
            fatal("gethostname: %.100s", strerror(errno));
      /*
       * auth_display must be used as the displayname when the
       * authorization entry is added with xauth(1).  This will be
       * different than the DISPLAY string for localhost displays.
       */
      if (options.x11_use_localhost) {
            snprintf(display, sizeof display, "localhost:%u.%u",
                s->display_number, s->screen);
            snprintf(auth_display, sizeof auth_display, "unix:%u.%u",
                s->display_number, s->screen);
            s->display = xstrdup(display);
            s->auth_display = xstrdup(auth_display);
      } else {
#ifdef IPADDR_IN_DISPLAY
```

```
            struct hostent *he;
            struct in_addr my_addr;

            he = gethostbyname(hostname);
            if (he == NULL) {
                  error("Can't get IP address for X11 DISPLAY.");
                  packet_send_debug("Can't  get  IP  address  for  X11
DISPLAY.");
                  return 0;
            }
            memcpy(&my_addr,      he->h_addr_list[0],      sizeof(struct
in_addr));
            snprintf(display,     sizeof     display,     "%.50s:%u.%u",
inet_ntoa(my_addr),
                s->display_number, s->screen);
#else
            snprintf(display, sizeof display, "%.400s:%u.%u", hostname,
                s->display_number, s->screen);
#endif
            s->display = xstrdup(display);
            s->auth_display = xstrdup(display);
      }

      return 1;
}

static void
do_authenticated2(Authctxt *authctxt)
{
      server_loop2(authctxt);
#if defined(GSSAPI)
      if (options.gss_cleanup_creds)
            ssh_gssapi_cleanup_creds(NULL);
#endif
}
```

C. SSHD.C

```
/*
 * Author: Tatu Ylonen <ylo@cs.hut.fi>
 * Copyright (c) 1995 Tatu Ylonen <ylo@cs.hut.fi>, Espoo, Finland
 *                    All rights reserved
 * This program is the ssh daemon.  It listens for connections from
clients,
 * and performs authentication, executes use commands or shell, and
forwards
 * information to/from the application to the user client over an
encrypted
 * connection.  This can also handle forwarding of X11, TCP/IP, and
 * authentication agent connections.
 *
 * As far as I am concerned, the code I have written for this software
 * can be used freely for any purpose.  Any derived versions of this
 * software must be clearly marked as such, and if the derived work is
 * incompatible with the protocol description in the RFC file, it must
be
 * called by a name other than "ssh" or "Secure Shell".
 *
```

```
 * SSH2 implementation:
 * Privilege Separation:
 *
 * Copyright (c) 2000, 2001, 2002 Markus Friedl.  All rights reserved.
 * Copyright (c) 2002 Niels Provos.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *     notice, this list of conditions and the following disclaimer in
the
 *         documentation and/or other materials provided with the
distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
 *  IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES
 *  OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED.
 * IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE OF
 * THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include "includes.h"
RCSID("$OpenBSD: sshd.c,v 1.276 2003/08/28 12:54:34 markus Exp $");

#include <openssl/dh.h>
#include <openssl/bn.h>
#include <openssl/md5.h>
#include <openssl/rand.h>
#ifdef HAVE_SECUREWARE
#include <sys/security.h>
#include <prot.h>
#endif

#include "ssh.h"
#include "ssh1.h"
#include "ssh2.h"
#include "xmalloc.h"
#include "rsa.h"
#include "sshpty.h"
#include "packet.h"
#include "mpaux.h"
#include "log.h"
#include "servconf.h"
```

```
#include "uidswap.h"
#include "compat.h"
#include "buffer.h"
#include "cipher.h"
#include "kex.h"
#include "key.h"
#include "dh.h"
#include "myproposal.h"
#include "authfile.h"
#include "pathnames.h"
#include "atomicio.h"
#include "canohost.h"
#include "auth.h"
#include "misc.h"
#include "dispatch.h"
#include "channels.h"
#include "session.h"
#include "monitor_mm.h"
#include "monitor.h"
#include "monitor_wrap.h"
#include "monitor_fdpass.h"

#ifdef LIBWRAP
#include <tcpd.h>
#include <syslog.h>
int allow_severity = LOG_INFO;
int deny_severity = LOG_WARNING;
#endif /* LIBWRAP */

#ifndef O_NOCTTY
#define O_NOCTTY  0
#endif

#ifdef HAVE___PROGNAME
extern char *__progname;
#else
char *__progname;
#endif

/* Server configuration options. */
ServerOptions options;

/* Name of the server configuration file. */
char *config_file_name = _PATH_SERVER_CONFIG_FILE;

/*
 *  Flag indicating whether IPv4 or IPv6.   This can be set on the
command line.
 * Default value is AF_UNSPEC means both IPv4 and IPv6.
 */
int IPv4or6 = AF_UNSPEC;

/*
 * Debug mode flag.  This can be set on the command line.  If debug
 * mode is enabled, extra debugging output will be sent to the system
 *  log,  the  daemon  will  not  go  to  background,  and  will  exit  after
processing
```

```
 * the first connection.
 */
int debug_flag = 0;

/* Flag indicating that the daemon should only test the configuration
and keys. */
int test_flag = 0;

/* Flag indicating that the daemon is being started from inetd. */
int inetd_flag = 0;

/* Flag indicating that sshd should not detach and become a daemon. */
int no_daemon_flag = 0;

/* debug goes to stderr unless inetd_flag is set */
int log_stderr = 0;

/* Saved arguments to main(). */
char **saved_argv;
int saved_argc;

/*
 * The sockets that the server is listening; this is used in the SIGHUP
 * signal handler.
 */
#define     MAX_LISTEN_SOCKS  16
int listen_socks[MAX_LISTEN_SOCKS];
int num_listen_socks = 0;

/*
 * the client's version string, passed by sshd2 in compat mode. if !=
NULL,
 * sshd will skip the version-number exchange
 */
char *client_version_string = NULL;
char *server_version_string = NULL;

/* for rekeying XXX fixme */
Kex *xxx_kex;

/*
 * Any really sensitive data in the application is contained in this
 * structure. The idea is that this structure could be locked into
memory so
 * that the pages do not get written into swap.  However, there are
some
 * problems. The private key contains BIGNUMs, and we do not (in
principle)
 * have access to the internals of them, and locking just the structure
is
 * not very useful.  Currently, memory locking is not implemented.
 */
struct {
      Key   *server_key;            /* ephemeral server key */
      Key   *ssh1_host_key;         /* ssh1 host key */
      Key   **host_keys;            /* all private host keys */
      int   have_ssh1_key;
```

```
	int	have_ssh2_key;
	u_char	ssh1_cookie[SSH_SESSION_KEY_LENGTH];
} sensitive_data;

/*
 * Flag indicating whether the RSA server key needs to be regenerated.
 *  Is  set  in  the  SIGALRM  handler  and  cleared  when  the  key  is
regenerated.
 */
static volatile sig_atomic_t key_do_regen = 0;

/* This is set to true when a signal is received. */
static volatile sig_atomic_t received_sighup = 0;
static volatile sig_atomic_t received_sigterm = 0;

/* session identifier, used by RSA-auth */
u_char session_id[16];

/* same for ssh2 */
u_char *session_id2 = NULL;
u_int session_id2_len = 0;

/* record remote hostname or ip */
u_int utmp_len = MAXHOSTNAMELEN;

/* options.max_startup sized array of fd ints */
int *startup_pipes = NULL;
int startup_pipe;	/* in child */

/* variables used for privilege separation */
int use_privsep;
struct monitor *pmonitor;

/* message to be displayed after login */
Buffer loginmsg;

/* Prototypes for various functions defined later in this file. */
void destroy_sensitive_data(void);
void demote_sensitive_data(void);

static void do_ssh1_kex(void);
static void do_ssh2_kex(void);

/*MYSEA: Implement daemon function as daemonize*/
int daemonize(int nochdir, int noclose);

/*
 * Close all listening sockets
 */
static void
close_listen_socks(void)
{
	int i;

	for (i = 0; i < num_listen_socks; i++)
		close(listen_socks[i]);
	num_listen_socks = -1;
```

```
}

static void
close_startup_pipes(void)
{
      int i;

      if (startup_pipes)
            for (i = 0; i < options.max_startups; i++)
                  if (startup_pipes[i] != -1)
                        close(startup_pipes[i]);
}

/*
 * Signal handler for SIGHUP.   Sshd execs itself when it receives
SIGHUP;
 * the effect is to reread the configuration file (and to regenerate
 * the server key).
 */
static void
sighup_handler(int sig)
{
      int save_errno = errno;

      received_sighup = 1;
      signal(SIGHUP, sighup_handler);
      errno = save_errno;
}

/*
 * Called from the main program after receiving SIGHUP.
 * Restarts the server.
 */
static void
sighup_restart(void)
{
      logit("Received SIGHUP; restarting.");
      close_listen_socks();
      close_startup_pipes();
      execv(saved_argv[0], saved_argv);
      logit("RESTART    FAILED:    av[0]='%.100s',    error:    %.100s.",
saved_argv[0],
          strerror(errno));
      exit(1);
}

/*
 * Generic signal handler for terminating signals in the master daemon.
 */
static void
sigterm_handler(int sig)
{
      received_sigterm = sig;
}

/*
```

```
 * SIGCHLD handler.  This is called whenever a child dies.  This will
then
 * reap any zombies left by exited children.
 */
static void
main_sigchld_handler(int sig)
{
      int save_errno = errno;
      pid_t pid;
      int status;

      while ((pid = waitpid(-1, &status, WNOHANG)) > 0 ||
          (pid < 0 && errno == EINTR))
            ;

      signal(SIGCHLD, main_sigchld_handler);
      errno = save_errno;
}

/*
 *  Signal  handler  for  the  alarm  after  the  login  grace  period  has
expired.
 */
static void
grace_alarm_handler(int sig)
{
      /* XXX no idea how fix this signal handler */

      /* Log error and exit. */
      fatal("Timeout      before      authentication      for      %s",
get_remote_ipaddr());
}

/*
 * Signal handler for the key regeneration alarm.  Note that this
 * alarm only occurs in the daemon waiting for connections, and it does
not
 * do anything with the private key or random state before forking.
 * Thus there should be no concurrency control/asynchronous execution
 * problems.
 */
static void
generate_ephemeral_server_key(void)
{
      u_int32_t rnd = 0;
      int i;

      verbose("Generating %s%d bit RSA key.",
          sensitive_data.server_key     ?     "new     "     :     "",
options.server_key_bits);
      if (sensitive_data.server_key != NULL)
            key_free(sensitive_data.server_key);
      sensitive_data.server_key = key_generate(KEY_RSA1,
          options.server_key_bits);
      verbose("RSA key generation complete.");

      for (i = 0; i < SSH_SESSION_KEY_LENGTH; i++) {
```

```
            if (i % 4 == 0)
                  rnd = arc4random();
            sensitive_data.ssh1_cookie[i] = rnd & 0xff;
            rnd >>= 8;
      }
      arc4random_stir();
}

static void
key_regeneration_alarm(int sig)
{
      int save_errno = errno;

      signal(SIGALRM, SIG_DFL);
      errno = save_errno;
      key_do_regen = 1;
}

static void
sshd_exchange_identification(int sock_in, int sock_out)
{
      int i, mismatch;
      int remote_major, remote_minor;
      int major, minor;
      char *s;
      char buf[256];                /*   Must   not   be   larger   than
remote_version. */
      char remote_version[256];     /* Must be at least as big as buf.
*/

      if ((options.protocol & SSH_PROTO_1) &&
          (options.protocol & SSH_PROTO_2)) {
            major = PROTOCOL_MAJOR_1;
            minor = 99;
      } else if (options.protocol & SSH_PROTO_2) {
            major = PROTOCOL_MAJOR_2;
            minor = PROTOCOL_MINOR_2;
      } else {
            major = PROTOCOL_MAJOR_1;
            minor = PROTOCOL_MINOR_1;
      }
      snprintf(buf,  sizeof  buf,  "SSH-%d.%d-%.100s\n",  major,  minor,
SSH_VERSION);
      server_version_string = xstrdup(buf);

      /* Send our protocol version identification. */
      if (atomicio(vwrite, sock_out, server_version_string,
          strlen(server_version_string))
          != strlen(server_version_string)) {
            logit("Could    not    write    ident    string    to    %s",
get_remote_ipaddr());
            fatal_cleanup();
      }

      /* Read other sides version identification. */
      memset(buf, 0, sizeof(buf));
      for (i = 0; i < sizeof(buf) - 1; i++) {
```

```
            if (atomicio(read, sock_in, &buf[i], 1) != 1) {
                  logit("Did  not  receive  identification  string  from
%s",
                      get_remote_ipaddr());
                  fatal_cleanup();
            }
            if (buf[i] == '\r') {
                  buf[i] = 0;
                  /* Kludge for F-Secure Macintosh < 1.0.2 */
                  if (i == 12 &&
                      strncmp(buf, "SSH-1.5-W1.0", 12) == 0)
                        break;
                  continue;
            }
            if (buf[i] == '\n') {
                  buf[i] = 0;
                  break;
            }
      }
      buf[sizeof(buf) - 1] = 0;
      client_version_string = xstrdup(buf);

      /*
       * Check that the versions match.  In future this might accept
       * several versions and set appropriate flags to handle them.
       */
      if (sscanf(client_version_string, "SSH-%d.%d-%[^\n]\n",
          &remote_major, &remote_minor, remote_version) != 3) {
            s = "Protocol mismatch.\n";
            (void) atomicio(vwrite, sock_out, s, strlen(s));
            close(sock_in);
            close(sock_out);
            logit("Bad  protocol  version  identification  '%.100s'  from
%s",
                client_version_string, get_remote_ipaddr());
            fatal_cleanup();
      }
      debug("Client  protocol  version  %d.%d;  client  software  version
%.100s",
          remote_major, remote_minor, remote_version);

      compat_datafellows(remote_version);

      if (datafellows & SSH_BUG_PROBE) {
            logit("probed from %s with %s.  Don't panic.",
                get_remote_ipaddr(), client_version_string);
            fatal_cleanup();
      }

      if (datafellows & SSH_BUG_SCANNER) {
            logit("scanned from %s with %s.  Don't panic.",
                get_remote_ipaddr(), client_version_string);
            fatal_cleanup();
      }

      mismatch = 0;
      switch (remote_major) {
```

```
	case 1:
		if (remote_minor == 99) {
			if (options.protocol & SSH_PROTO_2)
				enable_compat20();
			else
				mismatch = 1;
			break;
		}
		if (!(options.protocol & SSH_PROTO_1)) {
			mismatch = 1;
			break;
		}
		if (remote_minor < 3) {
			packet_disconnect("Your ssh version is too old and "
			    "is no longer supported.  Please install a newer
version.");
		} else if (remote_minor == 3) {
			/* note that this disables agent-forwarding */
			enable_compat13();
		}
		break;
	case 2:
		if (options.protocol & SSH_PROTO_2) {
			enable_compat20();
			break;
		}
		/* FALLTHROUGH */
	default:
		mismatch = 1;
		break;
	}
	chop(server_version_string);
	debug("Local version string %.200s", server_version_string);

	if (mismatch) {
		s = "Protocol major versions differ.\n";
		(void) atomicio(vwrite, sock_out, s, strlen(s));
		close(sock_in);
		close(sock_out);
		logit("Protocol major versions differ for %s: %.200s vs.
%.200s",
		    get_remote_ipaddr(),
		    server_version_string, client_version_string);
		fatal_cleanup();
	}
}

/* Destroy the host and server keys.  They will no longer be needed. */
void
destroy_sensitive_data(void)
{
	int i;

	if (sensitive_data.server_key) {
		key_free(sensitive_data.server_key);
		sensitive_data.server_key = NULL;
	}
```

```
	for (i = 0; i < options.num_host_key_files; i++) {
		if (sensitive_data.host_keys[i]) {
			key_free(sensitive_data.host_keys[i]);
			sensitive_data.host_keys[i] = NULL;
		}
	}
	sensitive_data.ssh1_host_key = NULL;
	memset(sensitive_data.ssh1_cookie, 0, SSH_SESSION_KEY_LENGTH);
}

/* Demote private to public keys for network child */
void
demote_sensitive_data(void)
{
	Key *tmp;
	int i;

	if (sensitive_data.server_key) {
		tmp = key_demote(sensitive_data.server_key);
		key_free(sensitive_data.server_key);
		sensitive_data.server_key = tmp;
	}

	for (i = 0; i < options.num_host_key_files; i++) {
		if (sensitive_data.host_keys[i]) {
			tmp = key_demote(sensitive_data.host_keys[i]);
			key_free(sensitive_data.host_keys[i]);
			sensitive_data.host_keys[i] = tmp;
			if (tmp->type == KEY_RSA1)
				sensitive_data.ssh1_host_key = tmp;
		}
	}

	/* We do not clear ssh1_host key and cookie.  XXX - Okay Niels?
*/
}

static void
privsep_preauth_child(void)
{
	u_int32_t rnd[256];
	gid_t gidset[1];
	struct passwd *pw;
	int i;

	/*  Enable  challenge-response  authentication  for  privilege
separation */
	privsep_challenge_enable();

	for (i = 0; i < 256; i++)
		rnd[i] = arc4random();
	RAND_seed(rnd, sizeof(rnd));

	/* Demote the private keys to public keys. */
	demote_sensitive_data();

	if ((pw = getpwnam(SSH_PRIVSEP_USER)) == NULL)
```

```
            fatal("Privilege separation user %s does not exist",
                SSH_PRIVSEP_USER);
      memset(pw->pw_passwd, 0, strlen(pw->pw_passwd));
      endpwent();

      /* Change our root directory */
      /*MYSEA: chroot is not supported so comment out for now*/
      //if (chroot(_PATH_PRIVSEP_CHROOT_DIR) == -1)
      //    fatal("chroot(\"%s\"): %s", _PATH_PRIVSEP_CHROOT_DIR,
      //        strerror(errno));
      /*MYSEA:change directory to var run empty*/
      if (chdir("/var/empty") == -1)
            fatal("chdir(\"/var/empty\"): %s", strerror(errno));

      /* Drop our privileges */
      debug3("privsep user:group %u:%u", (u_int)pw->pw_uid,
          (u_int)pw->pw_gid);
#if 0
      /* XXX not ready, to heavy after chroot */
      do_setusercontext(pw);
#else
      gidset[0] = pw->pw_gid;
      /*MYSEA: setgroups is not implemented*/
      /*if (setgroups(1, gidset) < 0)
            fatal("setgroups: %.100s", strerror(errno));*/
      permanently_set_uid(pw);
#endif
}

static Authctxt *
privsep_preauth(void)
{
      Authctxt *authctxt = NULL;
      int status;
      pid_t pid;

      /* Set up unprivileged child process to deal with network data */
      pmonitor = monitor_init();
      /* Store a pointer to the kex for later rekeying */
      pmonitor->m_pkex = &xxx_kex;

      pid = fork();
      if (pid == -1) {
            fatal("fork of unprivileged child failed");
      } else if (pid != 0) {
            fatal_remove_cleanup((void   (*)   (void   *))   packet_close,
NULL);

            debug2("Network child is on pid %ld", (long)pid);
            /*MYSEA:   Need   to   close   both   of   the   child's   file
descriptors*/
            close(pmonitor->m_recvfd);
            //close(pmonitor->m_childrecvfd);
            //close(pmonitor->m_childsendfd);
            authctxt = monitor_child_preauth(pmonitor);
            /*MYSEA:now close both of the parent's file descriptors*/
            close(pmonitor->m_sendfd);
```

```
            //close(pmonitor->m_parentrecvfd);
            //close(pmonitor->m_parentsendfd);

            /* Sync memory */
            monitor_sync(pmonitor);

            /* Wait for the child's exit status */
            while (waitpid(pid, &status, 0) < 0)
                  if (errno != EINTR)
                        break;

            /* Reinstall, since the child has finished */
            fatal_add_cleanup((void (*) (void *)) packet_close, NULL);

            return (authctxt);
      } else {
            /* child */
            /*MYSEA: close the parent side of the file descriptors*/
            close(pmonitor->m_sendfd);
            //close(pmonitor->m_parentrecvfd);
            //close(pmonitor->m_parentsendfd);

            /* Demote the child */
            /*MYSEA: no root user, 3 is network user*/
            if (getuid() == 3 || geteuid() == 3)
                  privsep_preauth_child();
            setproctitle("%s", "[net]");
      }
      return (NULL);
}

static void
privsep_postauth(Authctxt *authctxt)
{
      extern Authctxt *x_authctxt;

      /* XXX - Remote port forwarding */
      x_authctxt = authctxt;

#ifdef DISABLE_FD_PASSING
      if (1) {
#else
      /*MYSEA:network user is 3*/
      if (authctxt->pw->pw_uid == 3 || options.use_login) {
#endif
            /* File descriptor passing is broken or root login */
            monitor_apply_keystate(pmonitor);
            use_privsep = 0;
            return;
      }

      /* Authentication complete */
      alarm(0);
      if (startup_pipe != -1) {
            close(startup_pipe);
            startup_pipe = -1;
      }
```

```
	/* New socket pair */
	monitor_reinit(pmonitor);

	pmonitor->m_pid = fork();
	if (pmonitor->m_pid == -1)
		fatal("fork of unprivileged child failed");
	else if (pmonitor->m_pid != 0) {
		fatal_remove_cleanup((void (*) (void *)) packet_close,
NULL);

		debug2("User child is on pid %ld", (long)pmonitor->m_pid);
		/*MYSEA: close child side of descriptors*/
		close(pmonitor->m_recvfd);
		//close(pmonitor->m_childrecvfd);
		//close(pmonitor->m_childsendfd);
		monitor_child_postauth(pmonitor);

		/* NEVERREACHED */
		exit(0);
	}
	/*MYSEA: Child side, close the parent side of descriptors*/
	close(pmonitor->m_sendfd);
	//close(pmonitor->m_parentrecvfd);
	//close(pmonitor->m_parentsendfd);

	/* Demote the private keys to public keys. */
	demote_sensitive_data();

	/* Drop privileges */
	do_setusercontext(authctxt->pw);

	/* It is safe now to apply the key state */
	monitor_apply_keystate(pmonitor);
}

static char *
list_hostkey_types(void)
{
	Buffer b;
	char *p;
	int i;

	buffer_init(&b);
	for (i = 0; i < options.num_host_key_files; i++) {
		Key *key = sensitive_data.host_keys[i];
		if (key == NULL)
			continue;
		switch (key->type) {
		case KEY_RSA:
		case KEY_DSA:
			if (buffer_len(&b) > 0)
				buffer_append(&b, ",", 1);
			p = key_ssh_name(key);
			buffer_append(&b, p, strlen(p));
			break;
		}
```

```
	}
	buffer_append(&b, "\0", 1);
	p = xstrdup(buffer_ptr(&b));
	buffer_free(&b);
	debug("list_hostkey_types: %s", p);
	return p;
}

Key *
get_hostkey_by_type(int type)
{
	int i;

	for (i = 0; i < options.num_host_key_files; i++) {
		Key *key = sensitive_data.host_keys[i];
		if (key != NULL && key->type == type)
			return key;
	}
	return NULL;
}

Key *
get_hostkey_by_index(int ind)
{
	if (ind < 0 || ind >= options.num_host_key_files)
		return (NULL);
	return (sensitive_data.host_keys[ind]);
}

int
get_hostkey_index(Key *key)
{
	int i;

	for (i = 0; i < options.num_host_key_files; i++) {
		if (key == sensitive_data.host_keys[i])
			return (i);
	}
	return (-1);
}

/*
 * returns 1 if connection should be dropped, 0 otherwise.
 * dropping starts at connection #max_startups_begin with a probability
 * of (max_startups_rate/100). the probability increases linearly until
 * all connections are dropped for startups > max_startups
 */
static int
drop_connection(int startups)
{
	double p, r;

	if (startups < options.max_startups_begin)
		return 0;
	if (startups >= options.max_startups)
		return 1;
	if (options.max_startups_rate == 100)
```

```
            return 1;

      p  = 100 - options.max_startups_rate;
      p *= startups - options.max_startups_begin;
      p        /=        (double)        (options.max_startups        -
options.max_startups_begin);
      p += options.max_startups_rate;
      p /= 100.0;
      r = arc4random() / (double) UINT_MAX;

      debug("drop_connection: p %g, r %g", p, r);
      return (r < p) ? 1 : 0;
}

static void
usage(void)
{
      fprintf(stderr, "sshd version %s\n", SSH_VERSION);
      fprintf(stderr, "Usage: %s [options]\n", __progname);
      fprintf(stderr, "Options:\n");
      fprintf(stderr, "  -f file    Configuration file (default %s)\n",
_PATH_SERVER_CONFIG_FILE);
      fprintf(stderr, "  -d          Debugging mode (multiple -d means
more debugging)\n");
      fprintf(stderr, "  -i         Started from inetd\n");
      fprintf(stderr, "  -D         Do not fork into daemon mode\n");
      fprintf(stderr, "  -t           Only test configuration file and
keys\n");
      fprintf(stderr, "  -q         Quiet (no logging)\n");
      fprintf(stderr, "   -p port       Listen on the specified port
(default: 22)\n");
      fprintf(stderr, "   -k seconds Regenerate server key every this
many seconds (default: 3600)\n");
      fprintf(stderr, "   -g seconds Grace period for authentication
(default: 600)\n");
      fprintf(stderr, "   -b bits     Size of server RSA key (default:
768 bits)\n");
      fprintf(stderr, "   -h file     File from which to read host key
(default: %s)\n",
          _PATH_HOST_KEY_FILE);
      fprintf(stderr, "   -u len      Maximum hostname length for utmp
recording\n");
      fprintf(stderr, "  -4         Use IPv4 only\n");
      fprintf(stderr, "  -6         Use IPv6 only\n");
      fprintf(stderr, "   -o option   Process the option as if it was
read from a configuration file.\n");
      exit(1);
}

/*
 * Main program for the daemon.
 */
int
main(int ac, char **av)
{
      extern char *optarg;
      extern int optind;
```

```
      int opt, sock_in = 0, sock_out = 0, newsock, j, i, fdsetsz, on =
1;
      pid_t pid;
      socklen_t fromlen;
      fd_set *fdset;
      struct sockaddr_storage from;
      const char *remote_ip;
      int remote_port;
      FILE *f;
      struct addrinfo *ai;
      char ntop[NI_MAXHOST], strport[NI_MAXSERV];
      int listen_sock, maxfd;
      int startup_p[2];
      int startups = 0;
      Authctxt *authctxt;
      Key *key;
      int ret, key_used = 0;
      /*MYSEA: Need to Create File Descriptors so that descriptors will
start numbering above 2*/
      int tf1, tf2, tf3;

#ifdef HAVE_SECUREWARE
      (void)set_auth_parameters(ac, av);
#endif
      __progname = ssh_get_progname(av[0]);
      init_rng();
      /*MYSEA: OPen files so that descriptors are above 2*/
      tf1 = open(_PATH_DEVNULL,O_RDWR, 0);
      tf2 = open(_PATH_DEVNULL,O_RDWR, 0);
      tf3 = open(_PATH_DEVNULL,O_RDWR, 0);


      /* Save argv. Duplicate so setproctitle emulation doesn't clobber
it */
      saved_argc = ac;
      saved_argv = xmalloc(sizeof(*saved_argv) * (ac + 1));
      for (i = 0; i < ac; i++)
            saved_argv[i] = xstrdup(av[i]);
      saved_argv[i] = NULL;

#ifndef HAVE_SETPROCTITLE
      /* Prepare for later setproctitle emulation */
      compat_init_setproctitle(ac, av);
      av = saved_argv;
#endif

      /* Initialize configuration options to their default values. */
      initialize_server_options(&options);

      /* Parse command-line arguments. */
      while ((opt = getopt(ac, av, "f:p:b:k:h:g:u:o:dDeiqtQ46")) != -1)
{
            switch (opt) {
            case '4':
                  IPv4or6 = AF_INET;
                  break;
            case '6':
```

```
                IPv4or6 = AF_INET6;
                break;
        case 'f':
                config_file_name = optarg;
                break;
        case 'd':
                if (debug_flag == 0) {
                        debug_flag = 1;
                        options.log_level = SYSLOG_LEVEL_DEBUG1;
                } else if (options.log_level < SYSLOG_LEVEL_DEBUG3)
                        options.log_level++;
                break;
        case 'D':
                no_daemon_flag = 1;
                break;
        case 'e':
                log_stderr = 1;
                break;
        case 'i':
                inetd_flag = 1;
                break;
        case 'Q':
                /* ignored */
                break;
        case 'q':
                options.log_level = SYSLOG_LEVEL_QUIET;
                break;
        case 'b':
                options.server_key_bits = atoi(optarg);
                break;
        case 'p':
                options.ports_from_cmdline = 1;
                if (options.num_ports >= MAX_PORTS) {
                        fprintf(stderr, "too many ports.\n");
                        exit(1);
                }
                options.ports[options.num_ports++] = a2port(optarg);
                if (options.ports[options.num_ports-1] == 0) {
                        fprintf(stderr, "Bad port number.\n");
                        exit(1);
                }
                break;
        case 'g':
                if ((options.login_grace_time = convtime(optarg)) ==
-1) {
                        fprintf(stderr, "Invalid login grace time.\n");
                        exit(1);
                }
                break;
        case 'k':
                if          ((options.key_regeneration_time          =
convtime(optarg)) == -1) {
                        fprintf(stderr,   "Invalid   key   regeneration
interval.\n");
                        exit(1);
                }
                break;
```

```
            case 'h':
                  if (options.num_host_key_files >= MAX_HOSTKEYS) {
                        fprintf(stderr, "too many host keys.\n");
                        exit(1);
                  }
                  options.host_key_files[options.num_host_key_files++]
= optarg;
                  break;
            case 't':
                  test_flag = 1;
                  break;
            case 'u':
                  utmp_len = atoi(optarg);
                  if (utmp_len > MAXHOSTNAMELEN) {
                        fprintf(stderr, "Invalid utmp length.\n");
                        exit(1);
                  }
                  break;
            case 'o':
                  if (process_server_config_line(&options, optarg,
                      "command-line", 0) != 0)
                        exit(1);
                  break;
            case '?':
            default:
                  usage();
                  break;
            }
      }
      SSLeay_add_all_algorithms();
      channel_set_af(IPv4or6);
      /*
       * Force logging to stderr until we have loaded the private host
       * key (unless started from inetd)
       */
      log_init(__progname,
          options.log_level == SYSLOG_LEVEL_NOT_SET ?
          SYSLOG_LEVEL_INFO : options.log_level,
          options.log_facility == SYSLOG_FACILITY_NOT_SET ?
          SYSLOG_FACILITY_AUTH : options.log_facility,
          log_stderr || !inetd_flag);

#ifdef _UNICOS
      /* Cray can define user privs drop all prives now!
       * Not needed on PRIV_SU systems!
       */
      drop_cray_privs();
#endif

      seed_rng();

      /* Read server configuration options from the configuration file.
*/
      read_server_config(&options, config_file_name);

      /* Fill in default values for those options not explicitly set.
*/
```

```
	fill_default_server_options(&options);

	/* Check that there are no remaining arguments. */
	if (optind < ac) {
		fprintf(stderr, "Extra argument %s.\n", av[optind]);
		exit(1);
	}

	debug("sshd version %.100s", SSH_VERSION);

	/* load private host keys */
	sensitive_data.host_keys = xmalloc(options.num_host_key_files *
	    sizeof(Key *));
	for (i = 0; i < options.num_host_key_files; i++)
		sensitive_data.host_keys[i] = NULL;
	sensitive_data.server_key = NULL;
	sensitive_data.ssh1_host_key = NULL;
	sensitive_data.have_ssh1_key = 0;
	sensitive_data.have_ssh2_key = 0;

	for (i = 0; i < options.num_host_key_files; i++) {
		key   =   key_load_private(options.host_key_files[i],   "",
NULL);
		sensitive_data.host_keys[i] = key;
		if (key == NULL) {
			error("Could not load host key: %s",
			    options.host_key_files[i]);
			sensitive_data.host_keys[i] = NULL;
			continue;
		}
		switch (key->type) {
		case KEY_RSA1:
			sensitive_data.ssh1_host_key = key;
			sensitive_data.have_ssh1_key = 1;
			break;
		case KEY_RSA:
		case KEY_DSA:
			sensitive_data.have_ssh2_key = 1;
			break;
		}
		debug("private host key: #%d type %d %s", i, key->type,
		    key_type(key));
	}
	if	((options.protocol	&	SSH_PROTO_1)	&&
!sensitive_data.have_ssh1_key) {
		logit("Disabling protocol version 1. Could not load host
key");
		options.protocol &= ~SSH_PROTO_1;
	}
	if	((options.protocol	&	SSH_PROTO_2)	&&
!sensitive_data.have_ssh2_key) {
		logit("Disabling protocol version 2. Could not load host
key");
		options.protocol &= ~SSH_PROTO_2;
	}
	if (!(options.protocol & (SSH_PROTO_1|SSH_PROTO_2))) {
		logit("sshd: no hostkeys available -- exiting.");
```

```
            exit(1);
      }

      /* Check certain values for sanity. */
      if (options.protocol & SSH_PROTO_1) {
            if (options.server_key_bits < 512 ||
                options.server_key_bits > 32768) {
                  fprintf(stderr, "Bad server key size.\n");
                  exit(1);
            }
            /*
             *  Check  that  server  and  host  key  lengths  differ
sufficiently. This
             * is necessary to make double encryption work with rsaref.
Oh, I
             * hate software patents. I dont know if this can go? Niels
             */
            if (options.server_key_bits >
                BN_num_bits(sensitive_data.ssh1_host_key->rsa->n) -
                SSH_KEY_BITS_RESERVED && options.server_key_bits <
                BN_num_bits(sensitive_data.ssh1_host_key->rsa->n) +
                SSH_KEY_BITS_RESERVED) {
                  options.server_key_bits =
                      BN_num_bits(sensitive_data.ssh1_host_key->rsa->n)
+
                      SSH_KEY_BITS_RESERVED;
                  debug("Forcing  server  key  to  %d  bits  to  make  it
differ from host key.",
                      options.server_key_bits);
            }
      }

      if (use_privsep) {
            struct passwd *pw;
            struct stat st;

            if ((pw = getpwnam(SSH_PRIVSEP_USER)) == NULL)
                  fatal("Privilege separation user %s does not exist",
                      SSH_PRIVSEP_USER);
            if ((stat(_PATH_PRIVSEP_CHROOT_DIR, &st) == -1) ||
                (S_ISDIR(st.st_mode) == 0))
                  fatal("Missing privilege separation directory: %s",
                      _PATH_PRIVSEP_CHROOT_DIR);

#ifdef HAVE_CYGWIN
            if (check_ntsec(_PATH_PRIVSEP_CHROOT_DIR) &&
                (st.st_uid != getuid () ||
                (st.st_mode & (S_IWGRP|S_IWOTH)) != 0))
#else       /* MYSEA:Change test to 3 for network user on XTS400*/
            if (st.st_uid != 3 || (st.st_mode & (S_IWGRP|S_IWOTH)) !=
0)
#endif
                  fatal("%s must be owned by root and not group or "
                      "world-writable.", _PATH_PRIVSEP_CHROOT_DIR);
      }

      /* Configuration looks good, so exit if in test mode. */
```

```
      if (test_flag)
            exit(0);

      /*
       * Clear out any supplemental groups we may have inherited.  This
       * prevents inadvertent creation of files with bad modes (in the
       * portable version at least, it's certainly possible for PAM
       * to create a file, and we can't control the code in every
       * module which might be used).
       */
      //MYSEA: setgroups is not implemented
      //if (setgroups(0, NULL) < 0)
      //    debug("setgroups() failed: %.200s", strerror(errno));

      /*  Initialize  the  log  (it  is  reinitialized  below  in  case  we
forked). */
      if (debug_flag && !inetd_flag)
            log_stderr = 1;
      log_init(__progname,   options.log_level,   options.log_facility,
log_stderr);

      /*
       *  If  not  in  debugging  mode,  and  not  started  from  inetd,
disconnect
       * from the controlling terminal, and fork.  The original process
       * exits.
       */
      if (!(debug_flag || inetd_flag || no_daemon_flag)) {
#ifdef TIOCNOTTY
            int fd;
#endif /* TIOCNOTTY */
            //MYSEA: Use daemonize function instead of daemon
            //They are the same thing
            if (daemonize(0, 0) < 0)
                  fatal("daemon() failed: %.200s", strerror(errno));

            /* Disconnect from the controlling tty. */
#ifdef TIOCNOTTY
            fd = open(_PATH_TTY, O_RDWR | O_NOCTTY);
            if (fd >= 0) {
                  (void) ioctl(fd, TIOCNOTTY, NULL);
                  close(fd);
            }
#endif /* TIOCNOTTY */
      }
      /* Reinitialize the log (because of the fork above). */
      log_init(__progname,   options.log_level,   options.log_facility,
log_stderr);

      /* Initialize the random number generator. */
      arc4random_stir();

      /* Chdir to the root directory so that the current disk can be
         unmounted if desired. */
      chdir("/");

      /* ignore SIGPIPE */
```

```
      signal(SIGPIPE, SIG_IGN);

      /* Start listening for a socket, unless started from inetd. */
      if (inetd_flag) {
            int s1;
            s1 = dup(0);      /* Make sure descriptors 0, 1, and 2 are
in use. */
            dup(s1);
            sock_in = dup(0);
            sock_out = dup(1);
            startup_pipe = -1;
            /*
             * We intentionally do not close the descriptors 0, 1, and
2
             * as our code for setting the descriptors won\'t work if
             * ttyfd happens to be one of those.
             */
            debug("inetd  sockets  after  dupping:  %d,  %d",  sock_in,
sock_out);
            if (options.protocol & SSH_PROTO_1)
                  generate_ephemeral_server_key();
      } else {
            for (ai = options.listen_addrs; ai; ai = ai->ai_next) {
                  if  (ai->ai_family  !=  AF_INET  &&  ai->ai_family  !=
AF_INET6)
                        continue;
                  if (num_listen_socks >= MAX_LISTEN_SOCKS)
                        fatal("Too many listen sockets. "
                            "Enlarge MAX_LISTEN_SOCKS");
                  if (getnameinfo(ai->ai_addr, ai->ai_addrlen,
                      ntop, sizeof(ntop), strport, sizeof(strport),
                      NI_NUMERICHOST|NI_NUMERICSERV) != 0) {
                        error("getnameinfo failed");
                        continue;
                  }
                  /* Create socket for listening. */
                  listen_sock = socket(ai->ai_family, ai->ai_socktype,
                      ai->ai_protocol);
                  if (listen_sock < 0) {
                        /* kernel may not support ipv6 */
                        verbose("socket: %.100s", strerror(errno));
                        continue;
                  }
                  if (fcntl(listen_sock, F_SETFL, O_NONBLOCK) < 0) {
                        error("listen_sock       O_NONBLOCK:       %s",
strerror(errno));
                        close(listen_sock);
                        continue;
                  }
                  /*
                   * Set socket options.
                   * Allow local port reuse in TIME_WAIT.
                   */
                  if (setsockopt(listen_sock, SOL_SOCKET, SO_REUSEADDR,
                      &on, sizeof(on)) == -1)
                        error("setsockopt       SO_REUSEADDR:       %s",
strerror(errno));
```

```
                debug("Bind to port %s on %s.", strport, ntop);

                /* Bind the socket to the desired port. */
                if (bind(listen_sock, ai->ai_addr, ai->ai_addrlen) <
0) {
                        if (!ai->ai_next)
                            error("Bind to port %s on %s failed:
%.200s.",
                                      strport, ntop, strerror(errno));
                        close(listen_sock);
                        continue;
                }
                listen_socks[num_listen_socks] = listen_sock;
                num_listen_socks++;

                /* Start listening on the port. */
                logit("Server listening on %s port %s.", ntop,
strport);
                if (listen(listen_sock, 5) < 0)
                        fatal("listen: %.100s", strerror(errno));

        }
        freeaddrinfo(options.listen_addrs);

        if (!num_listen_socks)
                fatal("Cannot bind any address.");

        if (options.protocol & SSH_PROTO_1)
                generate_ephemeral_server_key();

        /*
         * Arrange to restart on SIGHUP.  The handler needs
         * listen_sock.
         */
        signal(SIGHUP, sighup_handler);

        signal(SIGTERM, sigterm_handler);
        signal(SIGQUIT, sigterm_handler);

        /* Arrange SIGCHLD to be caught. */
        signal(SIGCHLD, main_sigchld_handler);

        /* Write out the pid file after the sigterm handler is
setup */
        if (!debug_flag) {
                /*
                 * Record our pid in /var/run/sshd.pid to make it
                 * easier to kill the correct sshd.  We don't want to
                 * do this before the bind above because the bind
will
                 * fail if there already is a daemon, and this will
                 * overwrite any old pid in the file.
                 */
                f = fopen(options.pid_file, "wb");
                if (f == NULL) {
                        error("Couldn't create pid file \"%s\": %s",
```

```
                        options.pid_file, strerror(errno));
                } else {
                        fprintf(f, "%ld\n", (long) getpid());
                        fclose(f);
                }
        }

        /* setup fd set for listen */
        fdset = NULL;
        maxfd = 0;
        for (i = 0; i < num_listen_socks; i++)
                if (listen_socks[i] > maxfd)
                        maxfd = listen_socks[i];
        /* pipes connected to unauthenticated childs */
        startup_pipes     =     xmalloc(options.max_startups     *
sizeof(int));
        for (i = 0; i < options.max_startups; i++)
                startup_pipes[i] = -1;

        /*
         * Stay listening for connections until the system crashes
or
         * the daemon is killed with a signal.
         */
        for (;;) {
                if (received_sighup)
                        sighup_restart();
                if (fdset != NULL)
                        xfree(fdset);
                fdsetsz     =     howmany(maxfd+1,     NFDBITS)     *
sizeof(fd_mask);
                fdset = (fd_set *)xmalloc(fdsetsz);
                memset(fdset, 0, fdsetsz);

                for (i = 0; i < num_listen_socks; i++)
                        FD_SET(listen_socks[i], fdset);
                for (i = 0; i < options.max_startups; i++)
                        if (startup_pipes[i] != -1)
                                FD_SET(startup_pipes[i], fdset);

                /* Wait in select until there is a connection. */
                ret = select(maxfd+1, fdset, NULL, NULL, NULL);
                if (ret < 0 && errno != EINTR)
                        error("select: %.100s", strerror(errno));
                if (received_sigterm) {
                        logit("Received signal %d; terminating.",
                            (int) received_sigterm);
                        close_listen_socks();
                        unlink(options.pid_file);
                        exit(255);
                }
                if (key_used && key_do_regen) {
                        generate_ephemeral_server_key();
                        key_used = 0;
                        key_do_regen = 0;
                }
                if (ret < 0)
```

```
                        continue;

                for (i = 0; i < options.max_startups; i++)
                        if (startup_pipes[i] != -1 &&
                            FD_ISSET(startup_pipes[i], fdset)) {
                                /*
                                 * the read end of the pipe is ready
                                 * if the child has closed the pipe
                                 * after successful authentication
                                 * or if the child has died
                                 */
                                close(startup_pipes[i]);
                                startup_pipes[i] = -1;
                                startups--;
                        }
                for (i = 0; i < num_listen_socks; i++) {
                        if (!FD_ISSET(listen_socks[i], fdset))
                                continue;
                        fromlen = sizeof(from);
                        newsock    =    accept(listen_socks[i],    (struct
sockaddr *)&from,
                            &fromlen);
                        if (newsock < 0) {
                                if   (errno   !=   EINTR   &&   errno   !=
EWOULDBLOCK)
                                        error("accept:              %.100s",
strerror(errno));
                                continue;
                        }
                        if (fcntl(newsock, F_SETFL, 0) < 0) {
                                error("newsock    del    O_NONBLOCK:    %s",
strerror(errno));
                                close(newsock);
                                continue;
                        }
                        if (drop_connection(startups) == 1) {
                                debug("drop connection #%d", startups);
                                close(newsock);
                                continue;
                        }
                        if (pipe(startup_p) == -1) {
                                close(newsock);
                                continue;
                        }

                        for (j = 0; j < options.max_startups; j++)
                                if (startup_pipes[j] == -1) {
                                        startup_pipes[j] = startup_p[0];
                                        if (maxfd < startup_p[0])
                                                maxfd = startup_p[0];
                                        startups++;
                                        break;
                                }

                        /*
                         * Got connection.  Fork a child to handle it,
unless
```

```
                         * we are in debugging mode.
                         */
                        if (debug_flag) {
                                /*
                                 *   In   debugging   mode.     Close   the
listening
                                 * socket, and start processing the
                                 * connection without forking.
                                 */
                                debug("Server will not fork when running
in debugging mode.");
                                close_listen_socks();
                                sock_in = newsock;
                                sock_out = newsock;
                                startup_pipe = -1;
                                pid = getpid();
                                break;
                        } else {
                                /*
                                 *  Normal production daemon.   Fork, and
have
                                 * the child process the connection. The
                                 * parent continues listening.
                                 */
                                if ((pid = fork()) == 0) {
                                        /*
                                         * Child.   Close the listening and
max_startup
                                         *   sockets.     Start   using   the
accepted socket.
                                         *  Reinitialize  logging  (since  our
pid has
                                         *  changed).   We  break  out  of  the
loop to handle
                                         * the connection.
                                         */
                                        startup_pipe = startup_p[1];
                                        close_startup_pipes();
                                        close_listen_socks();
                                        sock_in = newsock;
                                        sock_out = newsock;
                                        log_init(__progname,
options.log_level, options.log_facility, log_stderr);
                                        break;
                                }
                        }

                        /* Parent.  Stay in the loop. */
                        if (pid < 0)
                                error("fork: %.100s", strerror(errno));
                        else
                                debug("Forked child %ld.", (long)pid);

                        close(startup_p[1]);

                        /*  Mark  that  the  key  has  been  used  (it  was
"given" to the child). */
```

```
                        if ((options.protocol & SSH_PROTO_1) &&
                            key_used == 0) {
                              /*   Schedule   server   key   regeneration
alarm. */
                              signal(SIGALRM, key_regeneration_alarm);
                              alarm(options.key_regeneration_time);
                              key_used = 1;
                        }

                        arc4random_stir();

                        /*  Close  the  new  socket  (the  child  is  now
taking care of it). */
                        close(newsock);
                  }
                  /* child process check (or debug mode) */
                  if (num_listen_socks < 0)
                        break;
            }
      }

      /* This is the child processing a new connection. */

      /*
       * Create a new session and process group since the 4.4BSD
       * setlogin() affects the entire process group.  We don't
       * want the child to be able to affect the parent.
       */
#if !defined(SSHD_ACQUIRES_CTTY)
      /*
       * If setsid is called, on some platforms sshd will later acquire
a
       * controlling terminal which will result in "could not set
       * controlling tty" errors.
       */
      if (!debug_flag && !inetd_flag && setsid() < 0)
            error("setsid: %.100s", strerror(errno));
#endif

      /*
       *  Disable  the  key  regeneration  alarm.   We  will  not  regenerate
the
       * key since we are no longer in a position to give it to anyone.
We
       * will not restart on SIGHUP since it no longer makes sense.
       */
      alarm(0);
      signal(SIGALRM, SIG_DFL);
      signal(SIGHUP, SIG_DFL);
      signal(SIGTERM, SIG_DFL);
      signal(SIGQUIT, SIG_DFL);
      signal(SIGCHLD, SIG_DFL);
      signal(SIGINT, SIG_DFL);

      /* Set keepalives if requested. */
      if (options.keepalives &&
          setsockopt(sock_in, SOL_SOCKET, SO_KEEPALIVE, &on,
```

```
           sizeof(on)) < 0)
             error("setsockopt SO_KEEPALIVE: %.100s", strerror(errno));

       /*
        * Register our connection.  This turns encryption off because we
do
        * not have a key.
        */
       packet_set_connection(sock_in, sock_out);

       remote_port = get_remote_port();
       remote_ip = get_remote_ipaddr();

#ifdef LIBWRAP
       /* Check whether logins are denied from this host. */
       {
             struct request_info req;

             request_init(&req, RQ_DAEMON, __progname, RQ_FILE, sock_in,
0);
             fromhost(&req);

             if (!hosts_access(&req)) {
                   debug("Connection refused by tcp wrapper");
                   refuse(&req);
                   /* NOTREACHED */
                   fatal("libwrap refuse returns");
             }
       }
#endif /* LIBWRAP */

       /* Log the connection. */
       verbose("Connection    from    %.500s    port    %d",    remote_ip,
remote_port);

       /*
        * We don\'t want to listen forever unless the other side
        *  successfully  authenticates  itself.   So  we  set  up  an  alarm
which is
        * cleared after successful authentication.  A limit of zero
        *  indicates  no  limit.  Note  that  we  don\'t  set  the  alarm  in
debugging
        * mode; it is just annoying to have the server exit just when
you
        * are about to discover the bug.
        */
       signal(SIGALRM, grace_alarm_handler);
       if (!debug_flag)
             alarm(options.login_grace_time);

       sshd_exchange_identification(sock_in, sock_out);

       packet_set_nonblocking();

         /* prepare buffers to collect authentication messages */
       buffer_init(&loginmsg);
```

```
      if (use_privsep)
            if ((authctxt = privsep_preauth()) != NULL)
                  goto authenticated;

      /* perform the key exchange */
      /* authenticate user and start session */
      if (compat20) {
            do_ssh2_kex();
            authctxt = do_authentication2();
      } else {
            do_ssh1_kex();
            authctxt = do_authentication();
      }
      /*
       *  If  we  use  privilege  separation,  the  unprivileged  child
transfers
       * the current keystate and exits
       */
      if (use_privsep) {
            mm_send_keystate(pmonitor);
            exit(0);
      }

 authenticated:
      /*
       * In privilege separation, we fork another child and prepare
       * file descriptor passing.
       */
      if (use_privsep) {
            privsep_postauth(authctxt);
            /* the monitor process [priv] will not return */
            if (!compat20)
                  destroy_sensitive_data();
      }

      /* Perform session preparation. */
      do_authenticated(authctxt);

      /* The connection has been terminated. */
      verbose("Closing connection to %.100s", remote_ip);

#ifdef USE_PAM
      if (options.use_pam)
            finish_pam();
#endif /* USE_PAM */

      packet_close();

      if (use_privsep)
            mm_terminate();
      close(tf1);
      close(tf2);
      close(tf3);
      exit(0);
}

/*
```

```
 * Decrypt session_key_int using our private server key and private
host key
 * (key with larger modulus first).
 */
int
ssh1_session_key(BIGNUM *session_key_int)
{
      int rsafail = 0;

      if                      (BN_cmp(sensitive_data.server_key->rsa->n,
sensitive_data.ssh1_host_key->rsa->n) > 0) {
            /* Server key has bigger modulus. */
            if (BN_num_bits(sensitive_data.server_key->rsa->n) <
                BN_num_bits(sensitive_data.ssh1_host_key->rsa->n)      +
SSH_KEY_BITS_RESERVED) {
                  fatal("do_connection: %s: server_key %d < host_key %d
+ SSH_KEY_BITS_RESERVED %d",
                        get_remote_ipaddr(),
                        BN_num_bits(sensitive_data.server_key->rsa->n),
                        BN_num_bits(sensitive_data.ssh1_host_key->rsa-
>n),
                        SSH_KEY_BITS_RESERVED);
            }
            if (rsa_private_decrypt(session_key_int, session_key_int,
                sensitive_data.server_key->rsa) <= 0)
                  rsafail++;
            if (rsa_private_decrypt(session_key_int, session_key_int,
                sensitive_data.ssh1_host_key->rsa) <= 0)
                  rsafail++;
      } else {
            /* Host key has bigger modulus (or they are equal). */
            if (BN_num_bits(sensitive_data.ssh1_host_key->rsa->n) <
                BN_num_bits(sensitive_data.server_key->rsa->n)         +
SSH_KEY_BITS_RESERVED) {
                  fatal("do_connection: %s: host_key %d < server_key %d
+ SSH_KEY_BITS_RESERVED %d",
                        get_remote_ipaddr(),
                        BN_num_bits(sensitive_data.ssh1_host_key->rsa-
>n),
                        BN_num_bits(sensitive_data.server_key->rsa->n),
                        SSH_KEY_BITS_RESERVED);
            }
            if (rsa_private_decrypt(session_key_int, session_key_int,
                sensitive_data.ssh1_host_key->rsa) < 0)
                  rsafail++;
            if (rsa_private_decrypt(session_key_int, session_key_int,
                sensitive_data.server_key->rsa) < 0)
                  rsafail++;
      }
      return (rsafail);
}
/*
 * SSH1 key exchange
 */
static void
do_ssh1_kex(void)
{
```

```
	int i, len;
	int rsafail = 0;
	BIGNUM *session_key_int;
	u_char session_key[SSH_SESSION_KEY_LENGTH];
	u_char cookie[8];
	u_int cipher_type, auth_mask, protocol_flags;
	u_int32_t rnd = 0;

	/*
	 * Generate check bytes that the client must send back in the
user
	 * packet in order for it to be accepted; this is used to defy ip
	 * spoofing attacks.  Note that this only works against somebody
	 * doing IP spoofing from a remote machine; any machine on the
local
	 * network can still see outgoing packets and catch the random
	 * cookie.  This only affects rhosts authentication, and this is
one
	 * of the reasons why it is inherently insecure.
	 */
	for (i = 0; i < 8; i++) {
		if (i % 4 == 0)
			rnd = arc4random();
		cookie[i] = rnd & 0xff;
		rnd >>= 8;
	}

	/*
	 * Send our public key.  We include in the packet 64 bits of
random
	 * data that must be matched in the reply in order to prevent IP
	 * spoofing.
	 */
	packet_start(SSH_SMSG_PUBLIC_KEY);
	for (i = 0; i < 8; i++)
		packet_put_char(cookie[i]);

	/* Store our public server RSA key. */
	packet_put_int(BN_num_bits(sensitive_data.server_key->rsa->n));
	packet_put_bignum(sensitive_data.server_key->rsa->e);
	packet_put_bignum(sensitive_data.server_key->rsa->n);

	/* Store our public host RSA key. */
	packet_put_int(BN_num_bits(sensitive_data.ssh1_host_key->rsa-
>n));
	packet_put_bignum(sensitive_data.ssh1_host_key->rsa->e);
	packet_put_bignum(sensitive_data.ssh1_host_key->rsa->n);

	/* Put protocol flags. */
	packet_put_int(SSH_PROTOFLAG_HOST_IN_FWD_OPEN);

	/* Declare which ciphers we support. */
	packet_put_int(cipher_mask_ssh1(0));

	/* Declare supported authentication types. */
	auth_mask = 0;
	if (options.rhosts_rsa_authentication)
```

```
            auth_mask |= 1 << SSH_AUTH_RHOSTS_RSA;
      if (options.rsa_authentication)
            auth_mask |= 1 << SSH_AUTH_RSA;
      if (options.challenge_response_authentication == 1)
            auth_mask |= 1 << SSH_AUTH_TIS;
      if (options.password_authentication)
            auth_mask |= 1 << SSH_AUTH_PASSWORD;
      packet_put_int(auth_mask);

      /* Send the packet and wait for it to be sent. */
      packet_send();
      packet_write_wait();

      debug("Sent %d bit server key and %d bit host key.",
          BN_num_bits(sensitive_data.server_key->rsa->n),
          BN_num_bits(sensitive_data.ssh1_host_key->rsa->n));

      /* Read clients reply (cipher type and session key). */
      packet_read_expect(SSH_CMSG_SESSION_KEY);

      /* Get cipher type and check whether we accept this. */
      cipher_type = packet_get_char();

      if (!(cipher_mask_ssh1(0) & (1 << cipher_type)))
            packet_disconnect("Warning:   client   selects   unsupported
cipher.");

      /* Get check bytes from the packet.  These must match those we
         sent earlier with the public key packet. */
      for (i = 0; i < 8; i++)
            if (cookie[i] != packet_get_char())
                  packet_disconnect("IP  Spoofing  check  bytes  do  not
match.");

      debug("Encryption type: %.200s", cipher_name(cipher_type));

      /* Get the encrypted integer. */
      if ((session_key_int = BN_new()) == NULL)
            fatal("do_ssh1_kex: BN_new failed");
      packet_get_bignum(session_key_int);

      protocol_flags = packet_get_int();
      packet_set_protocol_flags(protocol_flags);
      packet_check_eom();

      /* Decrypt session_key_int using host/server keys */
      rsafail = PRIVSEP(ssh1_session_key(session_key_int));

      /*
       * Extract session key from the decrypted integer.  The key is in
the
       * least significant 256 bits of the integer; the first byte of
the
       * key is in the highest bits.
       */
      if (!rsafail) {
            BN_mask_bits(session_key_int, sizeof(session_key) * 8);
```

```
            len = BN_num_bytes(session_key_int);
            if (len < 0 || len > sizeof(session_key)) {
                  error("do_connection: bad session key len from %s: "
                      "session_key_int %d > sizeof(session_key) %lu",
                      get_remote_ipaddr(),                              len,
(u_long)sizeof(session_key));
                  rsafail++;
            } else {
                  memset(session_key, 0, sizeof(session_key));
                  BN_bn2bin(session_key_int,
                      session_key + sizeof(session_key) - len);

                  compute_session_id(session_id, cookie,
                      sensitive_data.ssh1_host_key->rsa->n,
                      sensitive_data.server_key->rsa->n);
                  /*
                   * Xor the first 16 bytes of the session key with the
                   * session id.
                   */
                  for (i = 0; i < 16; i++)
                        session_key[i] ^= session_id[i];
            }
      }
      if (rsafail) {
            int bytes = BN_num_bytes(session_key_int);
            u_char *buf = xmalloc(bytes);
            MD5_CTX md;

            logit("do_connection: generating a fake encryption key");
            BN_bn2bin(session_key_int, buf);
            MD5_Init(&md);
            MD5_Update(&md, buf, bytes);
            MD5_Update(&md,                   sensitive_data.ssh1_cookie,
SSH_SESSION_KEY_LENGTH);
            MD5_Final(session_key, &md);
            MD5_Init(&md);
            MD5_Update(&md, session_key, 16);
            MD5_Update(&md, buf, bytes);
            MD5_Update(&md,                   sensitive_data.ssh1_cookie,
SSH_SESSION_KEY_LENGTH);
            MD5_Final(session_key + 16, &md);
            memset(buf, 0, bytes);
            xfree(buf);
            for (i = 0; i < 16; i++)
                  session_id[i] = session_key[i] ^ session_key[i + 16];
      }
      /* Destroy the private and public keys. No longer. */
      destroy_sensitive_data();

      if (use_privsep)
            mm_ssh1_session_id(session_id);

      /* Destroy the decrypted integer.  It is no longer needed. */
      BN_clear_free(session_key_int);

      /* Set the session key.  From this on all communications will be
encrypted. */
```

```
	packet_set_encryption_key(session_key,	SSH_SESSION_KEY_LENGTH,
cipher_type);

	/* Destroy our copy of the session key.  It is no longer needed.
*/
	memset(session_key, 0, sizeof(session_key));

	debug("Received session key; encryption turned on.");

	/* Send an acknowledgment packet.  Note that this packet is sent
encrypted. */
	packet_start(SSH_SMSG_SUCCESS);
	packet_send();
	packet_write_wait();
}

/*
 * SSH2 key exchange: diffie-hellman-group1-sha1
 */
static void
do_ssh2_kex(void)
{
	Kex *kex;

	if (options.ciphers != NULL) {
		myproposal[PROPOSAL_ENC_ALGS_CTOS] =
		myproposal[PROPOSAL_ENC_ALGS_STOC] = options.ciphers;
	}
	myproposal[PROPOSAL_ENC_ALGS_CTOS] =
	    compat_cipher_proposal(myproposal[PROPOSAL_ENC_ALGS_CTOS]);
	myproposal[PROPOSAL_ENC_ALGS_STOC] =
	    compat_cipher_proposal(myproposal[PROPOSAL_ENC_ALGS_STOC]);

	if (options.macs != NULL) {
		myproposal[PROPOSAL_MAC_ALGS_CTOS] =
		myproposal[PROPOSAL_MAC_ALGS_STOC] = options.macs;
	}
	if (!options.compression) {
		myproposal[PROPOSAL_COMP_ALGS_CTOS] =
		myproposal[PROPOSAL_COMP_ALGS_STOC] = "none";
	}
	myproposal[PROPOSAL_SERVER_HOST_KEY_ALGS] = list_hostkey_types();

	/* start key exchange */
	kex = kex_setup(myproposal);
	kex->kex[KEX_DH_GRP1_SHA1] = kexdh_server;
	kex->kex[KEX_DH_GEX_SHA1] = kexgex_server;
	kex->server = 1;
	kex->client_version_string=client_version_string;
	kex->server_version_string=server_version_string;
	kex->load_host_key=&get_hostkey_by_type;
	kex->host_key_index=&get_hostkey_index;

	xxx_kex = kex;

	dispatch_run(DISPATCH_BLOCK, &kex->done, kex);
```

```
	session_id2 = kex->session_id;
	session_id2_len = kex->session_id_len;

#ifdef DEBUG_KEXDH
	/* send 1st encrypted/maced/compressed message */
	packet_start(SSH2_MSG_IGNORE);
	packet_put_cstring("markus");
	packet_send();
	packet_write_wait();
#endif
	debug("KEX done");
}

/*MYSEA: Definition of daemonize function*/
int daemonize(int nochdir, int noclose)
{
	int fd;
	switch (fork())
	{
	case -1:
		return (-1);
	case 0:
		break;
	default:
		_exit(0);
	}
	if (setsid() == -1)
		return (-1);
	if (!nochdir)
		(void)chdir("/");
	if (!noclose && (fd = open(_PATH_DEVNULL, O_RDWR, 0)) != -1)
	{
		(void)dup2(fd, STDIN_FILENO);
		(void)dup2(fd, STDOUT_FILENO);
		(void)dup2(fd, STDERR_FILENO);
		if (fd > 2)
			(void)close(fd);
	}
	return (0);
}
```

D. UIDSWAP.C

```
/*
 * Author: Tatu Ylonen <ylo@cs.hut.fi>
 * Copyright (c) 1995 Tatu Ylonen <ylo@cs.hut.fi>, Espoo, Finland
 *                    All rights reserved
 * Code for uid-swapping.
 *
 * As far as I am concerned, the code I have written for this software
 * can be used freely for any purpose.  Any derived versions of this
 * software must be clearly marked as such, and if the derived work is
 * incompatible with the protocol description in the RFC file, it must
be
 * called by a name other than "ssh" or "Secure Shell".
 */
```

```
#include "includes.h"
RCSID("$OpenBSD: uidswap.c,v 1.24 2003/05/29 16:58:45 deraadt Exp $");

#include "log.h"
#include "uidswap.h"
/*MYSEA: Need to include extra headers to make the daemon privileged to
change the user and group ids of the processes*/
#include </usr/local/mysea/include/priv_util.h>
#include </usr/local/mysea/include/util.h>
#include <xts/types.h>

/*
 * Note: all these functions must work in all of the following cases:
 *    1. euid=0, ruid=0
 *    2. euid=0, ruid!=0
 *    3. euid!=0, ruid!=0
 * Additionally, they must work regardless of whether the system has
 * POSIX saved uids or not.
 */

#if defined(_POSIX_SAVED_IDS) && !defined(BROKEN_SAVED_UIDS)
/* Lets assume that posix saved ids also work with seteuid, even though
that
   is not part of the posix specification. */
#define SAVED_IDS_WORK_WITH_SETEUID
/* Saved effective uid. */
static uid_t      saved_euid = 0;
static gid_t      saved_egid = 0;
#endif

/* Saved effective uid. */
static int  privileged = 0;
static int  temporarily_use_uid_effective = 0;
static gid_t      saved_egroups[NGROUPS_MAX], user_groups[NGROUPS_MAX];
static int  saved_egroupslen = -1, user_groupslen = -1;

/*
 * Temporarily changes to the given uid.  If the effective user
 * id is not root, this does nothing.  This call cannot be nested.
 */
void
temporarily_use_uid(struct passwd *pw)
{
      /*MYSEA: variable needed for privileged code*/
      xts_privilege_t old_priv;
      /* Save the current euid, and egroups. */
#ifdef SAVED_IDS_WORK_WITH_SETEUID
      saved_euid = geteuid();
      saved_egid = getegid();
      debug("temporarily_use_uid: %u/%u (e=%u/%u)",
          (u_int)pw->pw_uid, (u_int)pw->pw_gid,
          (u_int)saved_euid, (u_int)saved_egid);

      /*MYSEA: Change this to be the user the program will run as*/
      if (saved_euid != 3) {
            privileged = 0;
            return;
```

```
	}
#else
	if (geteuid() != 3) {
		privileged = 0;
		return;
	}
#endif /* SAVED_IDS_WORK_WITH_SETEUID */
	privileged = 1;
	temporarily_use_uid_effective = 1;
	saved_egroupslen = getgroups(NGROUPS_MAX, saved_egroups);
	if (saved_egroupslen < 0)
	{
		fatal("getgroups: %.100s", strerror(errno));
	}
	/* set and save the user's groups */
	if (user_groupslen == -1) {
		//MYSEA: initgroups is not implemented on the XTS-400
		//Comment out for now
		//if (initgroups(pw->pw_name, pw->pw_gid) < 0)
		//{
		//    fatal("initgroups: %s: %.100s", pw->pw_name,
		//        strerror(errno));
		//}
		user_groupslen = getgroups(NGROUPS_MAX, user_groups);
		if (user_groupslen < 0)
		{
			fatal("getgroups: %.100s", strerror(errno));
		}
	}
	/* Set the effective uid to the given (unprivileged) uid. */
	/*MYSEA: PRIVILEGED CODE*/
	old_priv = enable_uid_priv();
	//MYSEA: setgroups is not implemented on the XTS-400
	//Comment out for now
	//if (setgroups(user_groupslen, user_groups) < 0)
	//    fatal("setgroups: %.100s", strerror(errno));
#ifndef SAVED_IDS_WORK_WITH_SETEUID
	/* Propagate the privileged gid to all of our gids. */
	if (setgid(getegid()) < 0)
	{
		debug("setgid    %u:    %.100s",    (u_int)    getegid(),
strerror(errno));
	}
	/* Propagate the privileged uid to all of our uids. */
	if (setuid(geteuid()) < 0)
	{
		debug("setuid    %u:    %.100s",    (u_int)    geteuid(),
strerror(errno));
	}
#endif /* SAVED_IDS_WORK_WITH_SETEUID */
	if (setegid(pw->pw_gid) < 0)
	{
		fatal("setegid %u: %.100s", (u_int)pw->pw_gid,
		    strerror(errno));
	}
	if (seteuid(pw->pw_uid) == -1)
	{
```

```
            fatal("seteuid %u: %.100s\n",(u_int)pw->pw_uid,
                  strerror(errno));
      }
      /*MYSEA: Release the Privileges*/
      set_priv(old_priv);
}

/*
 * Restores to the original (privileged) uid.
 */
void
restore_uid(void)
{
      /*MYSEA: Need to add privileges here too in order to change the
user and group ids*/
      xts_privilege_t old_priv;
      /* it's a no-op unless privileged */
      if (!privileged) {
            debug("restore_uid: (unprivileged)\n");
            return;
      }
      if (!temporarily_use_uid_effective)
      {
            fatal("restore_uid: temporarily_use_uid not effective");
      }
      /*MYSEA: Enable the Privileges*/
      old_priv = enable_uid_priv();
#ifdef SAVED_IDS_WORK_WITH_SETEUID
      debug("restore_uid:           %u/%u",           (u_int)saved_euid,
(u_int)saved_egid);
      /* Set the effective uid back to the saved privileged uid. */
      if (seteuid(saved_euid) < 0)
      {
            fatal("seteuid     %u:     %.100s",     (u_int)saved_euid,
strerror(errno));
      }
      if (setegid(saved_egid) < 0)
      {
            fatal("setegid     %u:     %.100s",     (u_int)saved_egid,
strerror(errno));
      }
#else /* SAVED_IDS_WORK_WITH_SETEUID */
      /*
       * We are unable to restore the real uid to its unprivileged
value.
       * Propagate the real uid (usually more privileged) to effective
uid
       * as well.
       */
      setuid(getuid());
      setgid(getgid());
#endif /* SAVED_IDS_WORK_WITH_SETEUID */
      //MYSEA: Setgroups is not implmented on the XTS-400, for now
comment out
      //if (setgroups(saved_egroupslen, saved_egroups) < 0)
      //    fatal("setgroups: %.100s", strerror(errno));
      temporarily_use_uid_effective = 0;
```

```
	/*MYSEA: Drop the Privileges*/
	set_priv(old_priv);
}

/*
 * Permanently sets all uids to the given uid.  This cannot be
 * called while temporarily_use_uid is effective.
 */
void
permanently_set_uid(struct passwd *pw)
{
	/*MYSEA: Privilege Code here as well*/
	xts_privilege_t old_priv;
	uid_t old_uid = getuid();
	gid_t old_gid = getgid();

	if (temporarily_use_uid_effective)
	{
		fatal("permanently_set_uid:		temporarily_use_uid
effective");
	}
	debug("permanently_set_uid: %u/%u", (u_int)pw->pw_uid,
	    (u_int)pw->pw_gid);

	/*MYSEA: Enable Privileges*/
	old_priv = enable_uid_priv();
#if defined(HAVE_SETRESGID)
	if (setresgid(pw->pw_gid, pw->pw_gid, pw->pw_gid) < 0)
	{
		fatal("setresgid	%u:	%.100s",	(u_int)pw->pw_gid,
strerror(errno));
	}
#elif defined(HAVE_SETREGID) && !defined(BROKEN_SETREGID)
	if (setregid(pw->pw_gid, pw->pw_gid) < 0)
	{
		fatal("setregid	%u:	%.100s",	(u_int)pw->pw_gid,
strerror(errno));
	}
#else
	if (setegid(pw->pw_gid) < 0)
	{
		fatal("setegid	%u:	%.100s",	(u_int)pw->pw_gid,
strerror(errno));
	}
	if (setgid(pw->pw_gid) < 0)
	{
		fatal("setgid	%u:	%.100s",	(u_int)pw->pw_gid,
strerror(errno));
	}
#endif

#if defined(HAVE_SETRESUID)
	if (setresuid(pw->pw_uid, pw->pw_uid, pw->pw_uid) < 0)
	{
		fatal("setresuid	%u:	%.100s",	(u_int)pw->pw_uid,
strerror(errno));
	}
```

```
#elif defined(HAVE_SETREUID) && !defined(BROKEN_SETREUID)
      if (setreuid(pw->pw_uid, pw->pw_uid) < 0)
      {
            fatal("setreuid     %u:     %.100s",     (u_int)pw->pw_uid,
strerror(errno));
      }
#else
# ifndef SETEUID_BREAKS_SETUID
      if (seteuid(pw->pw_uid) < 0)
      {
            fatal("seteuid     %u:     %.100s",     (u_int)pw->pw_uid,
strerror(errno));
      }
# endif
      if (setuid(pw->pw_uid) < 0)
      {
            fatal("setuid     %u:     %.100s",     (u_int)pw->pw_uid,
strerror(errno));
      }
#endif
      //MYSEA: Drop the privileges now!!!!
      set_priv(old_priv);
      /* Try restoration of GID if changed (test clearing of saved gid)
*/
      if (old_gid != pw->pw_gid &&
          (setgid(old_gid) != -1 || setegid(old_gid) != -1))
      {
            fatal("%s: was able to restore old [e]gid", __func__);
      }

      /* Verify GID drop was successful */
      if (getgid() != pw->pw_gid || getegid() != pw->pw_gid) {
            fatal("%s: egid incorrect gid:%u egid:%u (should be %u)",
                __func__, (u_int)getgid(), (u_int)getegid(),
                (u_int)pw->pw_gid);
      }

#ifndef HAVE_CYGWIN
      /* Try restoration of UID if changed (test clearing of saved uid)
*/
      if (old_uid != pw->pw_uid &&
          (setuid(old_uid) != -1 || seteuid(old_uid) != -1))
            fatal("%s: was able to restore old [e]uid", __func__);
#endif

      /* Verify UID drop was successful */
      if (getuid() != pw->pw_uid || geteuid() != pw->pw_uid) {
            fatal("%s: euid incorrect uid:%u euid:%u (should be %u)",
                __func__, (u_int)getuid(), (u_int)geteuid(),
                (u_int)pw->pw_uid);
      }
}
```

E. MONITOR.C

```
/*
 * Author: Tatu Ylonen <ylo@cs.hut.fi>
 * Copyright (c) 1995 Tatu Ylonen <ylo@cs.hut.fi>, Espoo, Finland
 *                    All rights reserved
 * Code for uid-swapping.
 *
 * As far as I am concerned, the code I have written for this software
 * can be used freely for any purpose.  Any derived versions of this
 * software must be clearly marked as such, and if the derived work is
 * incompatible with the protocol description in the RFC file, it must
be
 * called by a name other than "ssh" or "Secure Shell".
 */

#include "includes.h"
RCSID("$OpenBSD: uidswap.c,v 1.24 2003/05/29 16:58:45 deraadt Exp $");

#include "log.h"
#include "uidswap.h"
/*MYSEA: Need to include extra headers to make the daemon privileged to
change the user and group ids of the processes*/
#include </usr/local/mysea/include/priv_util.h>
#include </usr/local/mysea/include/util.h>
#include <xts/types.h>

/*
 * Note: all these functions must work in all of the following cases:
 *    1. euid=0, ruid=0
 *    2. euid=0, ruid!=0
 *    3. euid!=0, ruid!=0
 * Additionally, they must work regardless of whether the system has
 * POSIX saved uids or not.
 */

#if defined(_POSIX_SAVED_IDS) && !defined(BROKEN_SAVED_UIDS)
/* Lets assume that posix saved ids also work with seteuid, even though
that
   is not part of the posix specification. */
#define SAVED_IDS_WORK_WITH_SETEUID
/* Saved effective uid. */
static uid_t      saved_euid = 0;
static gid_t      saved_egid = 0;
#endif

/* Saved effective uid. */
static int  privileged = 0;
static int  temporarily_use_uid_effective = 0;
static gid_t      saved_egroups[NGROUPS_MAX], user_groups[NGROUPS_MAX];
static int  saved_egroupslen = -1, user_groupslen = -1;

/*
 * Temporarily changes to the given uid.  If the effective user
 * id is not root, this does nothing.  This call cannot be nested.
 */
void
```

```
temporarily_use_uid(struct passwd *pw)
{
      /*MYSEA: variable needed for privileged code*/
      xts_privilege_t old_priv;
      /* Save the current euid, and egroups. */
#ifdef SAVED_IDS_WORK_WITH_SETEUID
      saved_euid = geteuid();
      saved_egid = getegid();
      debug("temporarily_use_uid: %u/%u (e=%u/%u)",
          (u_int)pw->pw_uid, (u_int)pw->pw_gid,
          (u_int)saved_euid, (u_int)saved_egid);

      /*MYSEA: Change this to be the user the program will run as*/
      if (saved_euid != 3) {
            privileged = 0;
            return;
      }
#else
      if (geteuid() != 3) {
            privileged = 0;
            return;
      }
#endif /* SAVED_IDS_WORK_WITH_SETEUID */
      privileged = 1;
      temporarily_use_uid_effective = 1;
      saved_egroupslen = getgroups(NGROUPS_MAX, saved_egroups);
      if (saved_egroupslen < 0)
      {
            fatal("getgroups: %.100s", strerror(errno));
      }
      /* set and save the user's groups */
      if (user_groupslen == -1) {
            //MYSEA: initgroups is not implemented on the XTS-400
            //Comment out for now
            //if (initgroups(pw->pw_name, pw->pw_gid) < 0)
            //{
            //    fatal("initgroups: %s: %.100s", pw->pw_name,
            //        strerror(errno));
            //}
            user_groupslen = getgroups(NGROUPS_MAX, user_groups);
            if (user_groupslen < 0)
            {
                  fatal("getgroups: %.100s", strerror(errno));
            }
      }
      /* Set the effective uid to the given (unprivileged) uid. */
      /*MYSEA: PRIVILEGED CODE*/
      old_priv = enable_uid_priv();
      //MYSEA: setgroups is not implemented on the XTS-400
      //Comment out for now
      //if (setgroups(user_groupslen, user_groups) < 0)
      //    fatal("setgroups: %.100s", strerror(errno));
#ifndef SAVED_IDS_WORK_WITH_SETEUID
      /* Propagate the privileged gid to all of our gids. */
      if (setgid(getegid()) < 0)
      {
```

```
            debug("setgid    %u:    %.100s",    (u_int)    getegid(),
strerror(errno));
      }
      /* Propagate the privileged uid to all of our uids. */
      if (setuid(geteuid()) < 0)
      {
            debug("setuid    %u:    %.100s",    (u_int)    geteuid(),
strerror(errno));
      }
#endif /* SAVED_IDS_WORK_WITH_SETEUID */
      if (setegid(pw->pw_gid) < 0)
      {
            fatal("setegid %u: %.100s", (u_int)pw->pw_gid,
                strerror(errno));
      }
      if (seteuid(pw->pw_uid) == -1)
      {
            fatal("seteuid %u: %.100s\n",(u_int)pw->pw_uid,
                  strerror(errno));
      }
      /*MYSEA: Release the Privileges*/
      set_priv(old_priv);
}

/*
 * Restores to the original (privileged) uid.
 */
void
restore_uid(void)
{
      /*MYSEA: Need to add privileges here too in order to change the
user and group ids*/
      xts_privilege_t old_priv;
      /* it's a no-op unless privileged */
      if (!privileged) {
            debug("restore_uid: (unprivileged)\n");
            return;
      }
      if (!temporarily_use_uid_effective)
      {
            fatal("restore_uid: temporarily_use_uid not effective");
      }
      /*MYSEA: Enable the Privileges*/
      old_priv = enable_uid_priv();
#ifdef SAVED_IDS_WORK_WITH_SETEUID
      debug("restore_uid:          %u/%u",          (u_int)saved_euid,
(u_int)saved_egid);
      /* Set the effective uid back to the saved privileged uid. */
      if (seteuid(saved_euid) < 0)
      {
            fatal("seteuid     %u:     %.100s",     (u_int)saved_euid,
strerror(errno));
      }
      if (setegid(saved_egid) < 0)
      {
            fatal("setegid     %u:     %.100s",     (u_int)saved_egid,
strerror(errno));
```

```
	}
#else /* SAVED_IDS_WORK_WITH_SETEUID */
	/*
	 *  We  are  unable  to  restore  the  real  uid  to  its  unprivileged
value.
	 * Propagate the real uid (usually more privileged) to effective
uid
	 * as well.
	 */
	setuid(getuid());
	setgid(getgid());
#endif /* SAVED_IDS_WORK_WITH_SETEUID */
	//MYSEA:  Setgroups  is  not  implmented  on  the  XTS-400,  for  now
comment out
	//if (setgroups(saved_egroupslen, saved_egroups) < 0)
	//    fatal("setgroups: %.100s", strerror(errno));
	temporarily_use_uid_effective = 0;
	/*MYSEA: Drop the Privileges*/
	set_priv(old_priv);
}

/*
 * Permanently sets all uids to the given uid.  This cannot be
 * called while temporarily_use_uid is effective.
 */
void
permanently_set_uid(struct passwd *pw)
{
	/*MYSEA: Privilege Code here as well*/
	xts_privilege_t old_priv;
	uid_t old_uid = getuid();
	gid_t old_gid = getgid();

	if (temporarily_use_uid_effective)
	{
		fatal("permanently_set_uid:              temporarily_use_uid
effective");
	}
	debug("permanently_set_uid: %u/%u", (u_int)pw->pw_uid,
	    (u_int)pw->pw_gid);

	/*MYSEA: Enable Privileges*/
	old_priv = enable_uid_priv();
#if defined(HAVE_SETRESGID)
	if (setresgid(pw->pw_gid, pw->pw_gid, pw->pw_gid) < 0)
	{
		fatal("setresgid     %u:     %.100s",     (u_int)pw->pw_gid,
strerror(errno));
	}
#elif defined(HAVE_SETREGID) && !defined(BROKEN_SETREGID)
	if (setregid(pw->pw_gid, pw->pw_gid) < 0)
	{
		fatal("setregid     %u:     %.100s",     (u_int)pw->pw_gid,
strerror(errno));
	}
#else
	if (setegid(pw->pw_gid) < 0)
```

```
      {
            fatal("setegid      %u:      %.100s",      (u_int)pw->pw_gid,
strerror(errno));
      }
      if (setgid(pw->pw_gid) < 0)
      {
            fatal("setgid      %u:      %.100s",      (u_int)pw->pw_gid,
strerror(errno));
      }
#endif

#if defined(HAVE_SETRESUID)
      if (setresuid(pw->pw_uid, pw->pw_uid, pw->pw_uid) < 0)
      {
            fatal("setresuid     %u:     %.100s",     (u_int)pw->pw_uid,
strerror(errno));
      }
#elif defined(HAVE_SETREUID) && !defined(BROKEN_SETREUID)
      if (setreuid(pw->pw_uid, pw->pw_uid) < 0)
      {
            fatal("setreuid     %u:     %.100s",     (u_int)pw->pw_uid,
strerror(errno));
      }
#else
# ifndef SETEUID_BREAKS_SETUID
      if (seteuid(pw->pw_uid) < 0)
      {
            fatal("seteuid      %u:      %.100s",      (u_int)pw->pw_uid,
strerror(errno));
      }
# endif
      if (setuid(pw->pw_uid) < 0)
      {
            fatal("setuid      %u:      %.100s",      (u_int)pw->pw_uid,
strerror(errno));
      }
#endif
      //MYSEA: Drop the privileges now!!!!
      set_priv(old_priv);
      /* Try restoration of GID if changed (test clearing of saved gid)
*/
      if (old_gid != pw->pw_gid &&
          (setgid(old_gid) != -1 || setegid(old_gid) != -1))
      {
            fatal("%s: was able to restore old [e]gid", __func__);
      }

      /* Verify GID drop was successful */
      if (getgid() != pw->pw_gid || getegid() != pw->pw_gid) {
            fatal("%s: egid incorrect gid:%u egid:%u (should be %u)",
                __func__, (u_int)getgid(), (u_int)getegid(),
                (u_int)pw->pw_gid);
      }

#ifndef HAVE_CYGWIN
      /* Try restoration of UID if changed (test clearing of saved uid)
*/
```

```
      if (old_uid != pw->pw_uid &&
          (setuid(old_uid) != -1 || seteuid(old_uid) != -1))
            fatal("%s: was able to restore old [e]uid", __func__);
#endif

      /* Verify UID drop was successful */
      if (getuid() != pw->pw_uid || geteuid() != pw->pw_uid) {
            fatal("%s: euid incorrect uid:%u euid:%u (should be %u)",
                __func__, (u_int)getuid(), (u_int)geteuid(),
                (u_int)pw->pw_uid);
      }
}
```

F. MONITOR_WRAP.C

```
/*
 * Copyright 2002 Niels Provos <provos@citi.umich.edu>
 * Copyright 2002 Markus Friedl <markus@openbsd.org>
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *     notice, this list of conditions and the following disclaimer in
the
 *         documentation and/or other materials provided with the
distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
 *  IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES
 *  OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED.
 * IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE OF
 * THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/*MYSEA: This file provides functions that the unprivileged child of
the
 *monitor process can use to communicate with the monitor.  All changes
 *are from m_recvfd to either m_childsendfd or m_childrecvfd.
 */
#include "includes.h"
RCSID("$OpenBSD: monitor_wrap.c,v 1.31 2003/08/28 12:54:34 markus Exp
$");
```

```
#include <openssl/bn.h>
#include <openssl/dh.h>

#include "ssh.h"
#include "dh.h"
#include "kex.h"
#include "auth.h"
#include "auth-options.h"
#include "buffer.h"
#include "bufaux.h"
#include "packet.h"
#include "mac.h"
#include "log.h"
#include "zlib.h"
#include "monitor.h"
#include "monitor_wrap.h"
#include "xmalloc.h"
#include "atomicio.h"
#include "monitor_fdpass.h"
#include "getput.h"
#include "servconf.h"

#include "auth.h"
#include "channels.h"
#include "session.h"

#ifdef GSSAPI
#include "ssh-gss.h"
#endif

/* Imports */
extern int compat20;
extern Newkeys *newkeys[];
extern z_stream incoming_stream;
extern z_stream outgoing_stream;
extern struct monitor *pmonitor;
extern Buffer input, output;
extern ServerOptions options;

void
mm_request_send(int socket, enum monitor_reqtype type, Buffer *m)
{
      u_int mlen = buffer_len(m);
      u_char buf[5];

      debug3("%s entering: type %d", __func__, type);

      PUT_32BIT(buf, mlen + 1);
      buf[4] = (u_char) type;       /* 1st byte of payload is mesg-type
*/
      if (atomicio(vwrite, socket, buf, sizeof(buf)) != sizeof(buf))
            fatal("%s: write check 1", __func__);
      if (atomicio(vwrite, socket, buffer_ptr(m), mlen) != mlen)
            fatal("%s: write check 2", __func__);
}
```

```
void
mm_request_receive(int socket, Buffer *m)
{
      u_char buf[4];
      u_int msg_len;
      ssize_t res;

      debug3("%s entering", __func__);

      res = atomicio(read, socket, buf, sizeof(buf));
      if (res != sizeof(buf)) {
            if (res == 0)
                  fatal_cleanup();
            fatal("%s: read: %ld", __func__, (long)res);
      }
      msg_len = GET_32BIT(buf);
      if (msg_len > 256 * 1024)
            fatal("%s: read: bad msg_len %d", __func__, msg_len);
      buffer_clear(m);
      buffer_append_space(m, msg_len);
      res = atomicio(read, socket, buffer_ptr(m), msg_len);
      if (res != msg_len)
            fatal("%s: read: %ld != msg_len", __func__, (long)res);
}

void
mm_request_receive_expect(int socket, enum monitor_reqtype type, Buffer
*m)
{
      u_char rtype;

      debug3("%s entering: type %d", __func__, type);

      mm_request_receive(socket, m);
      rtype = buffer_get_char(m);
      if (rtype != type)
            fatal("%s: read: rtype %d != type %d", __func__,
                rtype, type);
}

DH *
mm_choose_dh(int min, int nbits, int max)
{
      BIGNUM *p, *g;
      int success = 0;
      Buffer m;

      buffer_init(&m);
      buffer_put_int(&m, min);
      buffer_put_int(&m, nbits);
      buffer_put_int(&m, max);
      /*MYSEA*/
      //Change m_recvfd to childsendfd
      mm_request_send(pmonitor->m_recvfd, MONITOR_REQ_MODULI, &m);

      debug3("%s: waiting for MONITOR_ANS_MODULI", __func__);
      //Change m_recvfd to m_childrecvfd
```

```
	mm_request_receive_expect(pmonitor->m_recvfd,  MONITOR_ANS_MODULI,
&m);

	success = buffer_get_char(&m);
	if (success == 0)
		fatal("%s: MONITOR_ANS_MODULI failed", __func__);

	if ((p = BN_new()) == NULL)
		fatal("%s: BN_new failed", __func__);
	if ((g = BN_new()) == NULL)
		fatal("%s: BN_new failed", __func__);
	buffer_get_bignum2(&m, p);
	buffer_get_bignum2(&m, g);

	debug3("%s: remaining %d", __func__, buffer_len(&m));
	buffer_free(&m);

	return (dh_new_group(g, p));
}

int
mm_key_sign(Key *key, u_char **sigp, u_int *lenp, u_char *data, u_int
datalen)
{
	Kex *kex = *pmonitor->m_pkex;
	Buffer m;

	debug3("%s entering", __func__);

	buffer_init(&m);
	buffer_put_int(&m, kex->host_key_index(key));
	buffer_put_string(&m, data, datalen);
	/*MYSEA*/
	//Change m_recvfd tp childsendfd
	mm_request_send(pmonitor->m_recvfd, MONITOR_REQ_SIGN, &m);

	debug3("%s: waiting for MONITOR_ANS_SIGN", __func__);
	//Change m_recvfd to m_childrecvfd
	mm_request_receive_expect(pmonitor->m_recvfd,    MONITOR_ANS_SIGN,
&m);
	*sigp  = buffer_get_string(&m, lenp);
	buffer_free(&m);

	return (0);
}

struct passwd *
mm_getpwnamallow(const char *login)
{
	Buffer m;
	struct passwd *pw;
	u_int pwlen;

	debug3("%s entering", __func__);

	buffer_init(&m);
	buffer_put_cstring(&m, login);
```

```
	/*MYSEA*/
	//Change m_recvfd to m_childsendfd
	mm_request_send(pmonitor->m_recvfd, MONITOR_REQ_PWNAM, &m);

	debug3("%s: waiting for MONITOR_ANS_PWNAM", __func__);
	//Change m_recvfd to m_childrecvfd
	mm_request_receive_expect(pmonitor->m_recvfd,   MONITOR_ANS_PWNAM,
&m);

	if (buffer_get_char(&m) == 0) {
		buffer_free(&m);
		return (NULL);
	}
	pw = buffer_get_string(&m, &pwlen);
	if (pwlen != sizeof(struct passwd))
		fatal("%s: struct passwd size mismatch", __func__);
	pw->pw_name = buffer_get_string(&m, NULL);
	pw->pw_passwd = buffer_get_string(&m, NULL);
	pw->pw_gecos = buffer_get_string(&m, NULL);
#ifdef HAVE_PW_CLASS_IN_PASSWD
	pw->pw_class = buffer_get_string(&m, NULL);
#endif
	pw->pw_dir = buffer_get_string(&m, NULL);
	pw->pw_shell = buffer_get_string(&m, NULL);
	buffer_free(&m);

	return (pw);
}

char *mm_auth2_read_banner(void)
{
	Buffer m;
	char *banner;

	debug3("%s entering", __func__);

	buffer_init(&m);
	/*MYSEA*/
	//Change m_recvfd to m_childsendfd
	mm_request_send(pmonitor->m_recvfd,
MONITOR_REQ_AUTH2_READ_BANNER, &m);
	buffer_clear(&m);
	//Change m_recvfd to m_childrecvfd
	mm_request_receive_expect(pmonitor->m_recvfd,
MONITOR_ANS_AUTH2_READ_BANNER, &m);
	banner = buffer_get_string(&m, NULL);
	buffer_free(&m);

	return (banner);
}

/* Inform the privileged process about service and style */

void
mm_inform_authserv(char *service, char *style)
{
	Buffer m;
```

```
	debug3("%s entering", __func__);

	buffer_init(&m);
	buffer_put_cstring(&m, service);
	buffer_put_cstring(&m, style ? style : "");
	/*MYSEA*/
	//Change m_recvfd to m_childsendfd
	mm_request_send(pmonitor->m_recvfd, MONITOR_REQ_AUTHSERV, &m);

	buffer_free(&m);
}

/* Do the password authentication */
int
mm_auth_password(Authctxt *authctxt, char *password)
{
	Buffer m;
	int authenticated = 0;

	debug3("%s entering", __func__);

	buffer_init(&m);
	buffer_put_cstring(&m, password);
	/*MYSEA*/
	//Change m_recvfd to m_childsendfd
	mm_request_send(pmonitor->m_recvfd,	MONITOR_REQ_AUTHPASSWORD,
&m);

	debug3("%s: waiting for MONITOR_ANS_AUTHPASSWORD", __func__);
	//Change m_recvfd to m_childrecvfd
	mm_request_receive_expect(pmonitor->m_recvfd,
MONITOR_ANS_AUTHPASSWORD, &m);

	authenticated = buffer_get_int(&m);

	buffer_free(&m);

	debug3("%s: user %sauthenticated",
	    __func__, authenticated ? "" : "not ");
	return (authenticated);
}

int
mm_user_key_allowed(struct passwd *pw, Key *key)
{
	return (mm_key_allowed(MM_USERKEY, NULL, NULL, key));
}

int
mm_hostbased_key_allowed(struct passwd *pw, char *user, char *host,
    Key *key)
{
	return (mm_key_allowed(MM_HOSTKEY, user, host, key));
}

int
```

```
mm_auth_rhosts_rsa_key_allowed(struct passwd *pw, char *user,
    char *host, Key *key)
{
      int ret;

      key->type = KEY_RSA; /* XXX hack for key_to_blob */
      ret = mm_key_allowed(MM_RSAHOSTKEY, user, host, key);
      key->type = KEY_RSA1;
      return (ret);
}

static void
mm_send_debug(Buffer *m)
{
      char *msg;

      while (buffer_len(m)) {
            msg = buffer_get_string(m, NULL);
            debug3("%s: Sending debug: %s", __func__, msg);
            packet_send_debug("%s", msg);
            xfree(msg);
      }
}

int
mm_key_allowed(enum mm_keytype type, char *user, char *host, Key *key)
{
      Buffer m;
      u_char *blob;
      u_int len;
      int allowed = 0, have_forced = 0;

      debug3("%s entering", __func__);

      /* Convert the key to a blob and the pass it over */
      if (!key_to_blob(key, &blob, &len))
            return (0);

      buffer_init(&m);
      buffer_put_int(&m, type);
      buffer_put_cstring(&m, user ? user : "");
      buffer_put_cstring(&m, host ? host : "");
      buffer_put_string(&m, blob, len);
      xfree(blob);
      /*MYSEA*/
      //Change m_recvfd to m_childsendfd
      mm_request_send(pmonitor->m_recvfd, MONITOR_REQ_KEYALLOWED, &m);

      debug3("%s: waiting for MONITOR_ANS_KEYALLOWED", __func__);
      //Change m_recvfd to m_childrecvfd
      mm_request_receive_expect(pmonitor->m_recvfd,
MONITOR_ANS_KEYALLOWED, &m);

      allowed = buffer_get_int(&m);

      /* fake forced command */
      auth_clear_options();
```

```
	have_forced = buffer_get_int(&m);
	forced_command = have_forced ? xstrdup("true") : NULL;

	/* Send potential debug messages */
	mm_send_debug(&m);

	buffer_free(&m);

	return (allowed);
}

/*
 * This key verify needs to send the key type along, because the
 * privileged parent makes the decision if the key is allowed
 * for authentication.
 */

int
mm_key_verify(Key *key, u_char *sig, u_int siglen, u_char *data, u_int
datalen)
{
	Buffer m;
	u_char *blob;
	u_int len;
	int verified = 0;

	debug3("%s entering", __func__);

	/* Convert the key to a blob and the pass it over */
	if (!key_to_blob(key, &blob, &len))
		return (0);

	buffer_init(&m);
	buffer_put_string(&m, blob, len);
	buffer_put_string(&m, sig, siglen);
	buffer_put_string(&m, data, datalen);
	xfree(blob);
	/*MYSEA*/
	//Change m_recvfd to m_childsendfd
	mm_request_send(pmonitor->m_recvfd, MONITOR_REQ_KEYVERIFY, &m);

	debug3("%s: waiting for MONITOR_ANS_KEYVERIFY", __func__);
	//Change m_recvfd to m_childrecvfd
	mm_request_receive_expect(pmonitor->m_recvfd,
MONITOR_ANS_KEYVERIFY, &m);

	verified = buffer_get_int(&m);

	buffer_free(&m);

	return (verified);
}

/* Export key state after authentication */
Newkeys *
mm_newkeys_from_blob(u_char *blob, int blen)
{
```

```
	Buffer b;
	u_int len;
	Newkeys *newkey = NULL;
	Enc *enc;
	Mac *mac;
	Comp *comp;

	debug3("%s: %p(%d)", __func__, blob, blen);
#ifdef DEBUG_PK
	dump_base64(stderr, blob, blen);
#endif
	buffer_init(&b);
	buffer_append(&b, blob, blen);

	newkey = xmalloc(sizeof(*newkey));
	enc = &newkey->enc;
	mac = &newkey->mac;
	comp = &newkey->comp;

	/* Enc structure */
	enc->name = buffer_get_string(&b, NULL);
	buffer_get(&b, &enc->cipher, sizeof(enc->cipher));
	enc->enabled = buffer_get_int(&b);
	enc->block_size = buffer_get_int(&b);
	enc->key = buffer_get_string(&b, &enc->key_len);
	enc->iv = buffer_get_string(&b, &len);
	if (len != enc->block_size)
		fatal("%s: bad ivlen: expected %u != %u", __func__,
		    enc->block_size, len);

	if (enc->name == NULL || cipher_by_name(enc->name) != enc->cipher)
		fatal("%s: bad cipher name %s or pointer %p", __func__,
		    enc->name, enc->cipher);

	/* Mac structure */
	mac->name = buffer_get_string(&b, NULL);
	if (mac->name == NULL || mac_init(mac, mac->name) == -1)
		fatal("%s: can not init mac %s", __func__, mac->name);
	mac->enabled = buffer_get_int(&b);
	mac->key = buffer_get_string(&b, &len);
	if (len > mac->key_len)
		fatal("%s: bad mac key length: %u > %d", __func__, len,
		    mac->key_len);
	mac->key_len = len;

	/* Comp structure */
	comp->type = buffer_get_int(&b);
	comp->enabled = buffer_get_int(&b);
	comp->name = buffer_get_string(&b, NULL);

	len = buffer_len(&b);
	if (len != 0)
		error("newkeys_from_blob: remaining bytes in blob %u",
len);
	buffer_free(&b);
	return (newkey);
```

```
}

int
mm_newkeys_to_blob(int mode, u_char **blobp, u_int *lenp)
{
	Buffer b;
	int len;
	Enc *enc;
	Mac *mac;
	Comp *comp;
	Newkeys *newkey = newkeys[mode];

	debug3("%s: converting %p", __func__, newkey);

	if (newkey == NULL) {
		error("%s: newkey == NULL", __func__);
		return 0;
	}
	enc = &newkey->enc;
	mac = &newkey->mac;
	comp = &newkey->comp;

	buffer_init(&b);
	/* Enc structure */
	buffer_put_cstring(&b, enc->name);
	/* The cipher struct is constant and shared, you export pointer
*/
	buffer_append(&b, &enc->cipher, sizeof(enc->cipher));
	buffer_put_int(&b, enc->enabled);
	buffer_put_int(&b, enc->block_size);
	buffer_put_string(&b, enc->key, enc->key_len);
	packet_get_keyiv(mode, enc->iv, enc->block_size);
	buffer_put_string(&b, enc->iv, enc->block_size);

	/* Mac structure */
	buffer_put_cstring(&b, mac->name);
	buffer_put_int(&b, mac->enabled);
	buffer_put_string(&b, mac->key, mac->key_len);

	/* Comp structure */
	buffer_put_int(&b, comp->type);
	buffer_put_int(&b, comp->enabled);
	buffer_put_cstring(&b, comp->name);

	len = buffer_len(&b);
	if (lenp != NULL)
		*lenp = len;
	if (blobp != NULL) {
		*blobp = xmalloc(len);
		memcpy(*blobp, buffer_ptr(&b), len);
	}
	memset(buffer_ptr(&b), 0, len);
	buffer_free(&b);
	return len;
}

static void
```

```
mm_send_kex(Buffer *m, Kex *kex)
{
      buffer_put_string(m, kex->session_id, kex->session_id_len);
      buffer_put_int(m, kex->we_need);
      buffer_put_int(m, kex->hostkey_type);
      buffer_put_int(m, kex->kex_type);
      buffer_put_string(m, buffer_ptr(&kex->my), buffer_len(&kex->my));
      buffer_put_string(m,   buffer_ptr(&kex->peer),   buffer_len(&kex-
>peer));
      buffer_put_int(m, kex->flags);
      buffer_put_cstring(m, kex->client_version_string);
      buffer_put_cstring(m, kex->server_version_string);
}

void
mm_send_keystate(struct monitor *pmonitor)
{
      Buffer m;
      u_char *blob, *p;
      u_int bloblen, plen;
      u_int32_t seqnr, packets;
      u_int64_t blocks;

      buffer_init(&m);

      if (!compat20) {
            u_char iv[24];
            u_char *key;
            u_int ivlen, keylen;

            buffer_put_int(&m, packet_get_protocol_flags());

            buffer_put_int(&m, packet_get_ssh1_cipher());

            debug3("%s: Sending ssh1 KEY+IV", __func__);
            keylen = packet_get_encryption_key(NULL);
            key = xmalloc(keylen+1);      /* add 1 if keylen == 0 */
            keylen = packet_get_encryption_key(key);
            buffer_put_string(&m, key, keylen);
            memset(key, 0, keylen);
            xfree(key);

            ivlen = packet_get_keyiv_len(MODE_OUT);
            packet_get_keyiv(MODE_OUT, iv, ivlen);
            buffer_put_string(&m, iv, ivlen);
            ivlen = packet_get_keyiv_len(MODE_OUT);
            packet_get_keyiv(MODE_IN, iv, ivlen);
            buffer_put_string(&m, iv, ivlen);
            goto skip;
      } else {
            /* Kex for rekeying */
            mm_send_kex(&m, *pmonitor->m_pkex);
      }

      debug3("%s: Sending new keys: %p %p",
          __func__, newkeys[MODE_OUT], newkeys[MODE_IN]);
```

```
	/* Keys from Kex */
	if (!mm_newkeys_to_blob(MODE_OUT, &blob, &bloblen))
		fatal("%s: conversion of newkeys failed", __func__);

	buffer_put_string(&m, blob, bloblen);
	xfree(blob);

	if (!mm_newkeys_to_blob(MODE_IN, &blob, &bloblen))
		fatal("%s: conversion of newkeys failed", __func__);

	buffer_put_string(&m, blob, bloblen);
	xfree(blob);

	packet_get_state(MODE_OUT, &seqnr, &blocks, &packets);
	buffer_put_int(&m, seqnr);
	buffer_put_int64(&m, blocks);
	buffer_put_int(&m, packets);
	packet_get_state(MODE_IN, &seqnr, &blocks, &packets);
	buffer_put_int(&m, seqnr);
	buffer_put_int64(&m, blocks);
	buffer_put_int(&m, packets);

	debug3("%s: New keys have been sent", __func__);
 skip:
	/* More key context */
	plen = packet_get_keycontext(MODE_OUT, NULL);
	p = xmalloc(plen+1);
	packet_get_keycontext(MODE_OUT, p);
	buffer_put_string(&m, p, plen);
	xfree(p);

	plen = packet_get_keycontext(MODE_IN, NULL);
	p = xmalloc(plen+1);
	packet_get_keycontext(MODE_IN, p);
	buffer_put_string(&m, p, plen);
	xfree(p);

	/* Compression state */
	debug3("%s: Sending compression state", __func__);
	buffer_put_string(&m, &outgoing_stream, sizeof(outgoing_stream));
	buffer_put_string(&m, &incoming_stream, sizeof(incoming_stream));

	/* Network I/O buffers */
	buffer_put_string(&m, buffer_ptr(&input), buffer_len(&input));
	buffer_put_string(&m, buffer_ptr(&output), buffer_len(&output));
	/*MYSEA*/
	//Change m_recvfd to m_childsendfd
	mm_request_send(pmonitor->m_recvfd, MONITOR_REQ_KEYEXPORT, &m);
	debug3("%s: Finished sending state", __func__);

	buffer_free(&m);
}

int
mm_pty_allocate(int *ptyfd, int *ttyfd, char *namebuf, int namebuflen)
{
	Buffer m;
```

```
	char *p;
	int success = 0;
	/*MYSEA: add socket structures*/
	//int sockfd, length;
	//struct sockaddr_un unix_addr, serv_addr;
	//char path[] = "/tmp/";
	buffer_init(&m);
	/*MYSEA: Use Pipe instead of Socketpair*/
	//Change m_recvfd to m_childsendfd
	mm_request_send(pmonitor->m_recvfd, MONITOR_REQ_PTY, &m);
	/*MYSEA: set up sockets, but don't connect until after the
message*/
	//if ((sockfd = socket(AF_LOCAL, SOCK_STREAM, 0)) < 0)
	//{
	//    printf("Could not create child socket:
%s\n",strerror(errno));
	//}
	//memset(&unix_addr, 0, sizeof(unix_addr));
	//unix_addr.sun_family = AF_LOCAL;
	/*Set up child socket*/
	//sprintf(unix_addr.sun_path,"%s%d", path, getpid());
	//unlink(unix_addr.sun_path);
	/*Bind the child's socket*/
	//if     ((bind(sockfd,     (struct     sockaddr*)     &unix_addr,
sizeof(unix_addr)) < 0))
	//{
	//    printf("Could not bind child socket:
%s\n",strerror(errno));
	//}
	//printf("child:%s\n",unix_addr.sun_path);
	/*MYSEA:initialize server socket, but don't connect*/
	//memset(&serv_addr, 0, sizeof(serv_addr));
	//serv_addr.sun_family = AF_LOCAL;
	//sprintf(serv_addr.sun_path,"%s%d", path, getppid());
	debug3("%s: waiting for MONITOR_ANS_PTY", __func__);
	//Change m_recvfd to m_childrecvfd
	mm_request_receive_expect(pmonitor->m_recvfd,     MONITOR_ANS_PTY,
&m);

	/*MYSEA: now call connect*/
	//length = SUN_LEN(&serv_addr);
	//if ((connect(sockfd, (struct sockaddr*) &serv_addr, length)) <
0)
	//{
	//    printf("Could    not    connect    from    child:    %s\n",
strerror(errno));
	//}
	success = buffer_get_int(&m);
	if (success == 0) {
		debug3("%s: pty alloc failed", __func__);
		buffer_free(&m);
		return (0);
	}
	p = buffer_get_string(&m, NULL);
	buffer_free(&m);

	strlcpy(namebuf, p, namebuflen); /* Possible truncation */
```

```
	xfree(p);
	/*MYSEA: use the new file descriptor*/
	//Change both fds to sockfd
	*ptyfd = mm_receive_fd(pmonitor->m_recvfd);
	*ttyfd = mm_receive_fd(pmonitor->m_recvfd);

	/*MYSEA: destroy the sockets now*/
	//close(sockfd);
	//unlink(unix_addr.sun_path);
	/* Success */
	return (1);
}

void
mm_session_pty_cleanup2(void *session)
{
	Session *s = session;
	Buffer m;

	if (s->ttyfd == -1)
		return;
	buffer_init(&m);
	buffer_put_cstring(&m, s->tty);
	/*MYSEA*/
	//Change m_recvfd to m_childsendfd
	mm_request_send(pmonitor->m_recvfd, MONITOR_REQ_PTYCLEANUP, &m);
	buffer_free(&m);

	/* closed dup'ed master */
	if (close(s->ptymaster) < 0)
		error("close(s->ptymaster): %s", strerror(errno));

	/* unlink pty from session */
	s->ttyfd = -1;
}

#ifdef USE_PAM
void
mm_start_pam(char *user)
{
	Buffer m;

	debug3("%s entering", __func__);
	if (!options.use_pam)
		fatal("UsePAM=no, but ended up in %s anyway", __func__);

	buffer_init(&m);
	buffer_put_cstring(&m, user);
	/*MYSEA*/
	//Change m_recvfd to m_childsendfd
	mm_request_send(pmonitor->m_recvfd, MONITOR_REQ_PAM_START, &m);

	buffer_free(&m);
}

u_int
mm_do_pam_account(void)
```

```
{
      Buffer m;
      u_int ret;

      debug3("%s entering", __func__);
      if (!options.use_pam)
            fatal("UsePAM=no, but ended up in %s anyway", __func__);

      buffer_init(&m);
      /*MYSEA*/
      //Change m_recvfd to m_childsendfd
      mm_request_send(pmonitor->m_recvfd, MONITOR_REQ_PAM_ACCOUNT, &m);
      //Change m_recvfd to m_childrecvfd
      mm_request_receive_expect(pmonitor->m_recvfd,
          MONITOR_ANS_PAM_ACCOUNT, &m);
      ret = buffer_get_int(&m);

      buffer_free(&m);

      debug3("%s returning %d", __func__, ret);

      return (ret);
}

void *
mm_sshpam_init_ctx(Authctxt *authctxt)
{
      Buffer m;
      int success;

      debug3("%s", __func__);
      buffer_init(&m);
      buffer_put_cstring(&m, authctxt->user);
      /*MYSEA*/
      //Change m_recvfd to m_childsendfd
      mm_request_send(pmonitor->m_recvfd,      MONITOR_REQ_PAM_INIT_CTX,
&m);
      debug3("%s: waiting for MONITOR_ANS_PAM_INIT_CTX", __func__);
      //change m_recvfd to m_childrecvfd
      mm_request_receive_expect(pmonitor->m_recvfd,
MONITOR_ANS_PAM_INIT_CTX, &m);
      success = buffer_get_int(&m);
      if (success == 0) {
            debug3("%s: pam_init_ctx failed", __func__);
            buffer_free(&m);
            return (NULL);
      }
      buffer_free(&m);
      return (authctxt);
}

int
mm_sshpam_query(void *ctx, char **name, char **info,
    u_int *num, char ***prompts, u_int **echo_on)
{
      Buffer m;
      int i, ret;
```

```
	debug3("%s", __func__);
	buffer_init(&m);
	/*MYSEA*/
	//Change m_recvfd to m_childsendfd
	mm_request_send(pmonitor->m_recvfd, MONITOR_REQ_PAM_QUERY, &m);
	debug3("%s: waiting for MONITOR_ANS_PAM_QUERY", __func__);
	//Change m_recvfd to m_childrecvfd
	mm_request_receive_expect(pmonitor->m_recvfd,
MONITOR_ANS_PAM_QUERY, &m);
	ret = buffer_get_int(&m);
	debug3("%s: pam_query returned %d", __func__, ret);
	*name = buffer_get_string(&m, NULL);
	*info = buffer_get_string(&m, NULL);
	*num = buffer_get_int(&m);
	*prompts = xmalloc((*num + 1) * sizeof(char *));
	*echo_on = xmalloc((*num + 1) * sizeof(u_int));
	for (i = 0; i < *num; ++i) {
		(*prompts)[i] = buffer_get_string(&m, NULL);
		(*echo_on)[i] = buffer_get_int(&m);
	}
	buffer_free(&m);
	return (ret);
}

int
mm_sshpam_respond(void *ctx, u_int num, char **resp)
{
	Buffer m;
	int i, ret;

	debug3("%s", __func__);
	buffer_init(&m);
	buffer_put_int(&m, num);
	for (i = 0; i < num; ++i)
		buffer_put_cstring(&m, resp[i]);
	/*MYSEA*/
	//Change m_recvfd to m_childsendfd
	mm_request_send(pmonitor->m_recvfd, MONITOR_REQ_PAM_RESPOND, &m);
	debug3("%s: waiting for MONITOR_ANS_PAM_RESPOND", __func__);
	//Change m_recvfd to m_childrecvfd
	mm_request_receive_expect(pmonitor->m_recvfd,
MONITOR_ANS_PAM_RESPOND, &m);
	ret = buffer_get_int(&m);
	debug3("%s: pam_respond returned %d", __func__, ret);
	buffer_free(&m);
	return (ret);
}

void
mm_sshpam_free_ctx(void *ctxtp)
{
	Buffer m;

	debug3("%s", __func__);
	buffer_init(&m);
	/*MYSEA*/
```

```
	//Change m_recvfd to m_childsendfd
	mm_request_send(pmonitor->m_recvfd,	MONITOR_REQ_PAM_FREE_CTX,
&m);
	debug3("%s: waiting for MONITOR_ANS_PAM_FREE_CTX", __func__);
	//Change m_recvfd to m_childrecvfd
	mm_request_receive_expect(pmonitor->m_recvfd,
MONITOR_ANS_PAM_FREE_CTX, &m);
	buffer_free(&m);
}
#endif /* USE_PAM */

/* Request process termination */

void
mm_terminate(void)
{
	Buffer m;

	buffer_init(&m);
	/*MYSEA*/
	//Change m_recvfd to m_childsend
	mm_request_send(pmonitor->m_recvfd, MONITOR_REQ_TERM, &m);
	buffer_free(&m);
}

int
mm_ssh1_session_key(BIGNUM *num)
{
	int rsafail;
	Buffer m;

	buffer_init(&m);
	buffer_put_bignum2(&m, num);
	/*MYSEA*/
	//Change m_recvfd to m_childsendfd
	mm_request_send(pmonitor->m_recvfd, MONITOR_REQ_SESSKEY, &m);
	//Change m_recvfd to m_childrecvfd
	mm_request_receive_expect(pmonitor->m_recvfd,
MONITOR_ANS_SESSKEY, &m);

	rsafail = buffer_get_int(&m);
	buffer_get_bignum2(&m, num);

	buffer_free(&m);

	return (rsafail);
}

static void
mm_chall_setup(char **name, char **infotxt, u_int *numprompts,
    char ***prompts, u_int **echo_on)
{
	*name = xstrdup("");
	*infotxt = xstrdup("");
	*numprompts = 1;
	*prompts = xmalloc(*numprompts * sizeof(char *));
	*echo_on = xmalloc(*numprompts * sizeof(u_int));
```

```
      (*echo_on)[0] = 0;
}

int
mm_bsdauth_query(void *ctx, char **name, char **infotxt,
   u_int *numprompts, char ***prompts, u_int **echo_on)
{
      Buffer m;
      u_int success;
      char *challenge;

      debug3("%s: entering", __func__);

      buffer_init(&m);
      /*MYSEA*/
      //Change m_recvfd to m_childsendfd
      mm_request_send(pmonitor->m_recvfd,      MONITOR_REQ_BSDAUTHQUERY,
&m);
      //Change m_recvfd to m_childrecvfd
      mm_request_receive_expect(pmonitor->m_recvfd,
MONITOR_ANS_BSDAUTHQUERY,
          &m);
      success = buffer_get_int(&m);
      if (success == 0) {
            debug3("%s: no challenge", __func__);
            buffer_free(&m);
            return (-1);
      }

      /* Get the challenge, and format the response */
      challenge  = buffer_get_string(&m, NULL);
      buffer_free(&m);

      mm_chall_setup(name, infotxt, numprompts, prompts, echo_on);
      (*prompts)[0] = challenge;

      debug3("%s: received challenge: %s", __func__, challenge);

      return (0);
}

int
mm_bsdauth_respond(void *ctx, u_int numresponses, char **responses)
{
      Buffer m;
      int authok;

      debug3("%s: entering", __func__);
      if (numresponses != 1)
            return (-1);

      buffer_init(&m);
      buffer_put_cstring(&m, responses[0]);
      /*MYSEA*/
      //Change m_recvfd to m_childsendfd
      mm_request_send(pmonitor->m_recvfd,    MONITOR_REQ_BSDAUTHRESPOND,
&m);
```

```
	//Change m_recvfd to m_childrecvfd
	mm_request_receive_expect(pmonitor->m_recvfd,
	    MONITOR_ANS_BSDAUTHRESPOND, &m);

	authok = buffer_get_int(&m);
	buffer_free(&m);

	return ((authok == 0) ? -1 : 0);
}

int
mm_skey_query(void *ctx, char **name, char **infotxt,
   u_int *numprompts, char ***prompts, u_int **echo_on)
{
	Buffer m;
	int len;
	u_int success;
	char *p, *challenge;

	debug3("%s: entering", __func__);

	buffer_init(&m);
	/*MYSEA*/
	//Change m_recvfd to m_childsendfd
	mm_request_send(pmonitor->m_recvfd, MONITOR_REQ_SKEYQUERY, &m);
	//Change m_recvfd to m_childrecvfd
	mm_request_receive_expect(pmonitor->m_recvfd,
MONITOR_ANS_SKEYQUERY,
	    &m);
	success = buffer_get_int(&m);
	if (success == 0) {
		debug3("%s: no challenge", __func__);
		buffer_free(&m);
		return (-1);
	}

	/* Get the challenge, and format the response */
	challenge  = buffer_get_string(&m, NULL);
	buffer_free(&m);

	debug3("%s: received challenge: %s", __func__, challenge);

	mm_chall_setup(name, infotxt, numprompts, prompts, echo_on);

	len = strlen(challenge) + strlen(SKEY_PROMPT) + 1;
	p = xmalloc(len);
	strlcpy(p, challenge, len);
	strlcat(p, SKEY_PROMPT, len);
	(*prompts)[0] = p;
	xfree(challenge);

	return (0);
}

int
mm_skey_respond(void *ctx, u_int numresponses, char **responses)
{
```

```
	Buffer m;
	int authok;

	debug3("%s: entering", __func__);
	if (numresponses != 1)
		return (-1);

	buffer_init(&m);
	buffer_put_cstring(&m, responses[0]);
	/*MYSEA*/
	//Change m_recvfd to m_childsendfd
	mm_request_send(pmonitor->m_recvfd, MONITOR_REQ_SKEYRESPOND, &m);
	//Change m_recvfd to m_childrecvfd
	mm_request_receive_expect(pmonitor->m_recvfd,
	    MONITOR_ANS_SKEYRESPOND, &m);

	authok = buffer_get_int(&m);
	buffer_free(&m);

	return ((authok == 0) ? -1 : 0);
}

void
mm_ssh1_session_id(u_char session_id[16])
{
	Buffer m;
	int i;

	debug3("%s entering", __func__);

	buffer_init(&m);
	for (i = 0; i < 16; i++)
		buffer_put_char(&m, session_id[i]);
	/*MYSEA*/
	//Change m_recvfd to m_childsendfd
	mm_request_send(pmonitor->m_recvfd, MONITOR_REQ_SESSID, &m);
	buffer_free(&m);
}

int
mm_auth_rsa_key_allowed(struct passwd *pw, BIGNUM *client_n, Key
**rkey)
{
	Buffer m;
	Key *key;
	u_char *blob;
	u_int blen;
	int allowed = 0, have_forced = 0;

	debug3("%s entering", __func__);

	buffer_init(&m);
	buffer_put_bignum2(&m, client_n);
	/*MYSEA*/
	//Change m_recvfd to m_childsendfd
	mm_request_send(pmonitor->m_recvfd,	MONITOR_REQ_RSAKEYALLOWED,
&m);
```

```
	//Change m_recvfd to m_childrecvfd
	mm_request_receive_expect(pmonitor->m_recvfd,
MONITOR_ANS_RSAKEYALLOWED, &m);

	allowed = buffer_get_int(&m);

	/* fake forced command */
	auth_clear_options();
	have_forced = buffer_get_int(&m);
	forced_command = have_forced ? xstrdup("true") : NULL;

	if (allowed && rkey != NULL) {
		blob = buffer_get_string(&m, &blen);
		if ((key = key_from_blob(blob, blen)) == NULL)
			fatal("%s: key_from_blob failed", __func__);
		*rkey = key;
		xfree(blob);
	}
	mm_send_debug(&m);
	buffer_free(&m);

	return (allowed);
}

BIGNUM *
mm_auth_rsa_generate_challenge(Key *key)
{
	Buffer m;
	BIGNUM *challenge;
	u_char *blob;
	u_int blen;

	debug3("%s entering", __func__);

	if ((challenge = BN_new()) == NULL)
		fatal("%s: BN_new failed", __func__);

	key->type = KEY_RSA;	/* XXX cheat for key_to_blob */
	if (key_to_blob(key, &blob, &blen) == 0)
		fatal("%s: key_to_blob failed", __func__);
	key->type = KEY_RSA1;

	buffer_init(&m);
	buffer_put_string(&m, blob, blen);
	xfree(blob);
	/*MYSEA*/
	//CHange m_recvfd to m_childsendfd
	mm_request_send(pmonitor->m_recvfd,	MONITOR_REQ_RSACHALLENGE,
&m);
	//Change m_recvfd to m_childrecvfd
	mm_request_receive_expect(pmonitor->m_recvfd,
MONITOR_ANS_RSACHALLENGE, &m);

	buffer_get_bignum2(&m, challenge);
	buffer_free(&m);

	return (challenge);
```

```
}

int
mm_auth_rsa_verify_response(Key *key, BIGNUM *p, u_char response[16])
{
      Buffer m;
      u_char *blob;
      u_int blen;
      int success = 0;

      debug3("%s entering", __func__);

      key->type = KEY_RSA;    /* XXX cheat for key_to_blob */
      if (key_to_blob(key, &blob, &blen) == 0)
            fatal("%s: key_to_blob failed", __func__);
      key->type = KEY_RSA1;

      buffer_init(&m);
      buffer_put_string(&m, blob, blen);
      buffer_put_string(&m, response, 16);
      xfree(blob);
      /*MYSEA*/
      //Change m_recvfd to m_childsendfd
      mm_request_send(pmonitor->m_recvfd, MONITOR_REQ_RSARESPONSE, &m);
      //Change m_recvfd to m_childrecvfd
      mm_request_receive_expect(pmonitor->m_recvfd,
MONITOR_ANS_RSARESPONSE, &m);

      success = buffer_get_int(&m);
      buffer_free(&m);

      return (success);
}

#ifdef GSSAPI
OM_uint32
mm_ssh_gssapi_server_ctx(Gssctxt **ctx, gss_OID oid)
{
      Buffer m;
      OM_uint32 major;

      /* Client doesn't get to see the context */
      *ctx = NULL;

      buffer_init(&m);
      buffer_put_string(&m, oid->elements, oid->length);
      /*MYSEA*/
      //Change m_recvfd to m_childsendfd
      mm_request_send(pmonitor->m_recvfd, MONITOR_REQ_GSSSETUP, &m);
      //Change m_recvfd to m_childrecvfd
      mm_request_receive_expect(pmonitor->m_recvfd,
MONITOR_ANS_GSSSETUP, &m);

      major = buffer_get_int(&m);

      buffer_free(&m);
      return (major);
```

```
}

OM_uint32
mm_ssh_gssapi_accept_ctx(Gssctxt *ctx, gss_buffer_desc *in,
    gss_buffer_desc *out, OM_uint32 *flags)
{
      Buffer m;
      OM_uint32 major;
      u_int len;

      buffer_init(&m);
      buffer_put_string(&m, in->value, in->length);
      /*MYSEA*/
      //Change m_recvfd to m_childsendfd
      mm_request_send(pmonitor->m_recvfd, MONITOR_REQ_GSSSTEP, &m);
      //Change m_recvfd to m_childrecvfd
      mm_request_receive_expect(pmonitor->m_recvfd,
MONITOR_ANS_GSSSTEP, &m);

      major = buffer_get_int(&m);
      out->value = buffer_get_string(&m, &len);
      out->length = len;
      if (flags)
            *flags = buffer_get_int(&m);

      buffer_free(&m);

      return (major);
}

int
mm_ssh_gssapi_userok(char *user)
{
      Buffer m;
      int authenticated = 0;

      buffer_init(&m);
      /*MYSEA*/
      //Change m_recvfd to m_childsendfd
      mm_request_send(pmonitor->m_recvfd, MONITOR_REQ_GSSUSEROK, &m);
      //Change m_recvfd to m_childrecvfd
      mm_request_receive_expect(pmonitor->m_recvfd,
MONITOR_ANS_GSSUSEROK,
                           &m);

      authenticated = buffer_get_int(&m);

      buffer_free(&m);
      debug3("%s: user %sauthenticated",__func__, authenticated ? "" :
"not ");
      return (authenticated);
}
#endif /* GSSAPI */
```

APPENDIX C: SSH DAEMON CONFIGURATION FILE

The purpose of this appendix is provide instructions on how to modify the SSH daemon configuration files that will be used by the OpenSSH daemons running on the XTS-400. Section A provides instructions on the modifications that need to be made. Section B provides a sample configuration file. The key word MYSEA is used to identify where the changes should occur.

A. SUMMARY OF REQUIRED CHANGES

There are six lines in the sshd_config file that need to be changed.

The first line is the Protocol option. Remove the '#' from the beginning of the line and remove the 1. The line should look like the line in the sshd_config file provided.

The next line is the ListenAddress Option. Remove the '#' from the beginning of the line and change the 0.0.0.0 IP address to the IP address assigned to the network interface.

The next option is PasswordAuthentication. Remove the '#' from the beginning of the line and change the yes to no. Refer to the sample file provided.

The next line has the PrintMotd option. Remove the '#' and change the yes to no.

The next line is PrintLastLog. Remove the '#' and change the yes to no.

The next line is the UsePrivilegeSeparation. Remove the '#' and change the yes to no.

Refer to the sample configuration file below.

B. SAMPLE CONFIGURATION FILE

```
#       $OpenBSD: sshd_config,v 1.65 2003/08/28 12:54:34 markus Exp $

# This is the sshd server system-wide configuration file.  See
# sshd_config(5) for more information.

# This sshd was compiled with
PATH=/usr/bin:/bin:/usr/sbin:/sbin:/home/cherbig/bin

# The strategy used for options in the default sshd_config shipped with
# OpenSSH is to specify options with their default value where
# possible, but leave them commented.  Uncommented options change a
# default value.
```

```
#MYSEA: uncomment the option for Protocol and use only 2 as shown.
#MYSEA: uncomment the ListenAddress option and change the 0.0.0.0 to
# the IP address of the network interface that this daemon will listen
#to.

#Port 22
Protocol 2
ListenAddress 192.168.100.22
#ListenAddress ::

# HostKey for protocol version 1
#HostKey /home/cherbig/etc/ssh_host_key
# HostKeys for protocol version 2
#HostKey /home/cherbig/etc/ssh_host_rsa_key
#HostKey /home/cherbig/etc/ssh_host_dsa_key

# Lifetime and size of ephemeral version 1 server key
#KeyRegenerationInterval 1h
#ServerKeyBits 768

# Logging
#obsoletes QuietMode and FascistLogging
#SyslogFacility AUTH
#LogLevel INFO

# Authentication:

#LoginGraceTime 2m
#PermitRootLogin yes
#StrictModes yes

#RSAAuthentication yes
#PubkeyAuthentication yes
#AuthorizedKeysFile     .ssh/authorized_keys

# For this to work you will also need host keys in
/home/cherbig/etc/ssh_known_hosts
#RhostsRSAAuthentication no
# similar for protocol version 2
#HostbasedAuthentication no
# Change to yes if you don't trust ~/.ssh/known_hosts for
# RhostsRSAAuthentication and HostbasedAuthentication
#IgnoreUserKnownHosts no
# Don't read the user's ~/.rhosts and ~/.shosts files
#IgnoreRhosts yes

#MYSEA: Uncomment the PasswordAuthentication option and change the yes
# to no.

# To disable tunneled clear text passwords, change to no here!
PasswordAuthentication no
#PermitEmptyPasswords no

# Change to no to disable s/key passwords
#ChallengeResponseAuthentication yes

# Kerberos options
```

```
#KerberosAuthentication no
#KerberosOrLocalPasswd yes
#KerberosTicketCleanup yes

# GSSAPI options
#GSSAPIAuthentication no
#GSSAPICleanupCreds yes

# Set this to 'yes' to enable PAM authentication (via challenge-
response)
# and session processing. Depending on your PAM configuration, this may
# bypass the setting of 'PasswordAuthentication'
#UsePAM no

#MYSEA: uncomment PrintMotd, PrintLastLog and UsePrivilegeSeparation
# and change the options to no.
#AllowTcpForwarding yes
#GatewayPorts no
#X11Forwarding no
#X11DisplayOffset 10
#X11UseLocalhost yes
PrintMotd no
PrintLastLog no
#KeepAlive yes
#UseLogin no
UsePrivilegeSeparation no
#PermitUserEnvironment no
#Compression yes
#ClientAliveInterval 0
#ClientAliveCountMax 3
#UseDNS yes
#PidFile /var/run/sshd.pid
#MaxStartups 10

# no default banner path
#Banner /some/path

# override default of no subsystems
      Subsystem   sftp  /home/cherbig/libexec/sftp-server
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D: KEY GENERATION, CONVERSION AND STORAGE

This appendix provides instructions on key generation and conversion for use with the public key authentication mechanism in OpenSSH. Keys may be generated on the XTS-400 by the system administrator or by the users on their personal Windows or Linux machines. The Secure Attention Key (SAK) on the XTS-400 console is the "alt" and "Print Screen" keys pressed together.

A. XTS-400 GENERATED KEYS

This procedure assumes that all user home directories are at min:il3. (level $HOME = sl1:il3) Only perform these steps for users that will be granted remote access to the system.

On the XTS-400, login as admin with a session level of min:max.

1. Create a directory called ".ssh" directly under the user's home directory

   Issue SAK

   Type fsm

   Type mkdir

   For path enter: /home/<username>/.ssh

   Type no for deflection directory

   Type change

   For path enter: /home/<username>/.ssh

   Type yes to modify mandatory access levels.

   Enter "min" for security level and "il3" for integrity level

   Change the name of the owner from admin to the username of the user's directory.

   Change the name of the group to the user's default group.

   Answer yes to change discretionary access

   Enter: rwx for owner

Hit enter for specific user

Type none for group

Hit enter for specific group

Type none for others

Type no for display object.

Type yes for okay to change.

Type exit

2. This step is used to generate the keys for the users. Only one pair of keys can be generated for one user at a time. Before generating a new pair of keys for another user, complete step 6, then start from step 1.

Issue SAK

Change levels to min:il3.

Issue SAK

Type run.

cd to the "/usr/local/src/bin" directory.

Type: ssh-keygen –t dsa –f ./id_dsa

Enter a passphrase. Reenter to confirm the passphrase. Write this down to give to the user.

Two files have been created: "id_dsa" and "id_dsa.pub".

Move the "id_dsa" file to disk by using *mcopy*:

mcopy id_dsa a:id_dsa

Verify that the file has been copied to disk with the *mdir* command.

3. Issue SAK.

Set levels to min:max.

Issue SAK

Use fsm to copy the two files, “/usr/local/src/bin/id_dsa” and “/usr/local/src/bin/id_dsa.pub” over to the user’s “.ssh” directory created earlier.

Issue SAK

Type sl

Type min for the security level

Type max for the integrity level

Type yes for is the level correct.

Issue SAK

Type fsm

Type copy

Enter /usr/local/src/bin/id_dsa.pub as the input pathname

Enter <path to user’s home directory>/.ssh/id_dsa.pub as the output pathname

Type yes for create output file

Type copy

Enter /usr/local/src/bin/id_dsa as the input pathname

Enter <path to user’s home directory>/.ssh/id_dsa as the output pathname

The keys that are in the /usr/local/src/bin directory may now be deleted.

Type delete

Enter /usr/local/src/bin/id_dsa.pub as the input pathname

Type no to display the object

Type yes to delete it

Type delete

Enter /usr/local/src/bin/id_dsa as the input pathname

Type no to display the object

Type yes to delete the file.

4. Use fsm's change command to change the ownership and discretionary permission of the files.

   Type change

   Enter the path: /home/<username>/.ssh/id_dsa

   Type no to modify mandatory access levels. (the levels should be min:il3)

   Enter the username of the user for the owner

   Enter the group name of the user's default group for the group

   Type yes for change discretionary permissions

   Type rw for owner

   Hit enter for specific user

   Type none for group

   Hit enter for specific group

   Type none for others

   Type no to display

   Types yes for okay to change.

5. Type change

   Enter the path :/home/<username>/.ssh/id_dsa.pub

   Type no to modify mandatory access levels. (the levels should be min:il3)

   Enter the username of the user for the owner

   Enter the group name of the user's group for the group

   Type yes for change discretionary permissions

Type rw for owner

Hit enter for specific user

Type r for group

Hit enter for specific group

Type r for others

Type no to display

Types yes for okay to change.

6. Use fsm to copy the “id_dsa.pub” file to the same directory but rename it “authorized_keys2”.

   After the file has been created, change the ownership and permissions to match that of the “id_dsa.pub” file. Repeat step 5 substituting authorized_keys2 for id_dsa.pub

B. PUTTY CONVERSION OF KEYS FROM XTS-400

For users who do not want to generate their own keys, the keys generated on the XTS-400 must be converted before use.

Take the id_dsa file provided by the administrator.

Double click on *puttygen*.

From the Conversions menu, select Import Key

Choose the file provided from the administrator (the file name should be id_dsa).

The user will be asked to enter the passphrase for the key. The administrator should have given them the passphrase.

The key should successfully be imported.

Click on Save private key.

Enter a file name of any kind. Save the key to a secure location, a network drive or a USB drive.

Click save.

When connecting using PuTTY, specify this new key and not the old one.

The old key that the system administrator gave to the user may be deleted.

Safeguard the new private key. When attempting to use the key, the passphrase will be required.

C. PUTTY GENERATED KEYS

For users who want to generate their own keys, they can use PuTTY's *puttygen* to generate keys.

On a windows machine that has PuTTY installed. Double click on the *puttygen* icon. This will open a window.

Select the "SSH2 DSA" radio button in the parameters section of the window.

The number of bits should be 1024.

Click on the "Generate" button in the actions section of the window.

Follow the instructions for moving the mouse to help generate some randomness.

Enter a passphrase in the "Key Passphrase" field in the "key" section of the window. Reenter the passphrase in the "Confirm passphrase" field.

Click on the Save Public Key and Save Private Key buttons in the Actions section.

Go to the "Conversions" menu and select "Export OpenSSH Key".

For the name of the file to save enter: "id_dsa.pub" and click on Save.

This file should be moved to disk or CD and given to the Administrator of the XTS-400.

The Private key should be used when connecting to the XTS-400.

The Administrator should use *mcopy* to copy the key file from the disk that the user has provided to the /usr/local/src/bin directory.

Repeat steps 1, 3, 5, and 6 from section A of this Appendix. In step 3, do not attempt to copy the id_dsa file.

D. OPENSSH GENERATED KEYS ON LINUX

Login to Linux system under a normal user account, not the root user account.

Open a terminal.

Type the command:

ssh-keygen –t dsa

Enter a passphrase when prompted.

Reenter passphrase to confirm.

This will create a directory called ".ssh" directly under the user's home directory and there will be two files in that directory: "id_dsa.pub" and "id_dsa".

Copy the "id_dsa.pub" file to disk and give to the Administrator of the XTS-400.

The Administrator should use *mcopy* to copy the key file from the disk that the user has provided to the /usr/local/src/bin directory.

Repeat steps 1, 3, 5, and 6 from section A of this Appendix. In step 3, do not attempt to copy the id_dsa file (it should not be on the disk).

E. LINUX INSTALLATION OF KEYS FROM XTS-400

Take the private key file from the administrator. Copy the file over to the user's .ssh directory under their home directory. If the floppy drive is mounted as /mnt/floppy, then use the following command:

cp /mnt/floppy/id_dsa $HOME/.ssh/id_dsa

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E: TOOLS

This appendix provides instructions on how to use the SSH clients and information on the other tools used in this project Section A describes how to use the two types of OpenSSH clients used for testing to connect to the XTS-400. Section B describes the tools used for development.

A. TESTING TOOLS

1. OpenSSH Client on Linux

The OpenSSH client on Linux and UNIX systems can be used to connect to the ported OpenSSH on the XTS-400.

Make sure that each user has generated a DSA key pair. Refer to section D in Appendix D for instructions on how to generate DSA key pairs. The public key must be installed on the XTS-400 prior to login attempts.

To connect, use the command:

ssh username@<host name or IP address>

When prompted, enter the passphrase for the private key.

If the login succeeds, a shell prompt is returned.

If the login fails, a message appears stating that the connection to the host was closed.

To exit, do not type *exit* or *logout*. Use the key sequence of a tilde and a period to close the connection:

~.

2. Putty

Make sure the PuTTY client is installed on the Windows machine. Refer to section C in Appendix A for instructions on how to install PuTTY. Make sure that the user has a private key that corresponds to the public key installed on the XTS-400. Refer to Appendix D for instructions on how to generate keys and install the public key on the XTS-400.

Double click on the PuTTY icon. This opens the PuTTY configuration window.

In the Host Name field, enter the host name or IP address of the XTS-400.

Click on the "Auth" option under the SSH category on the lower left side of the Window. Click on the "Browse" button to select the private key file. A new "Open File" dialog box is presented. Locate the private key file and click on the "Open" button.

Click on the "Open" button in the PuTTY configuration window.

This will open a terminal.

Enter a username when prompted for one.

Enter the passphrase of the private key when asked.

A shell is returned.

To exit, close the window by clicking on the 'x' in the top right corner of the window.

B. DEVELOPMENT TOOLS

1. Fedora core 1 linux

This distribution of Linux was used as a hands-on experimentation system. The experiments conducted on this system were used to provide baseline behaviors and results that were then compared to the behaviors and results produced by conducting experiments on the XTS-400. Fedora Linux can be downloaded from http://fedora.redhat.com.

2. Linux Cross Reference

This tool works with a web-server to display C program files as web pages. All identifiers are treated as links. This makes source code navigation easier by allowing the user to treat the source code directory as a website. All variables and functions are treated as links, allowing users to quickly navigate to the specific line in the specific file where the variable or function is defined. This tool may be downloaded from http://sourceforge.net/projects/lxr. This tool was installed in the Fedora Core 1 Linux system mentioned earlier.

This tool requires that a web-server be installed on the system as well. Apache 2.0 was also installed on the Fedora Linux system. Apache can be downloaded from http://www.apache.org.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX F: TEST PROCEDURES

The purpose of this appendix to provide the testing procedures used in the tests as described in Chapter IV. The Secure Attention Key (SAK) on the XTS-400 console is the “alt” and “Print Screen” keys pressed together.

Three user accounts must be created for these tests: *demo*, *cherbig* and *testuser*. The username, *cherbig*, may be changed to *testuser2*. In this case all references to *cherbig* should be changed to *testuser2*. Their default session levels should be min:il3. The mandatory access levels of their home directories should be min:il3. Each user account should have a public and private key. Refer to Appendix D for instructions on how to create and install keys.

1. MAC POLICY ENFORCEMENT

Login at the console as the demo user at default session level.

Create the following directories with *mkdir* as specified by the directory name column in the table.

Directory Name	Mandatory Levels
sl0il3	sl0:il3
sl1il3	sl1:il3
sl2il3	sl2:il3
sl3il3	sl3:il3
sl4il3	sl4:il3
sl1il0	sl1:il0
sl1il1	sl1:il1
sl1il2	sl1:il2

Table 25. MAC Policy Test Directories

Create the following files with *vi* typing a short text message in each:

Filename	Mandatory Levels	Directory
test_sl0il3.txt	sl0:il3	sl0il3
test_sl1il3.txt	sl1:il3	sl1il3
test_sl2il3.txt	sl2:il3	sl2il3
test_sl3il3.txt	sl3:il3	sl3il3
test_sl4il3.txt	sl4:il3	sl4il3
test_sl1il0.txt	sl1:il0	sl1il0
test_sl1il1.txt	sl1:il1	sl1il1
test_sl1il2.txt	sl1:il2	sl1il2

Table 26. MAC Policy Test Files

Issue SAK.

Type fsm.

Use the *change* command in fsm to change the mandatory levels of the above files to their respective levels. Use the Table 26 to identify which levels to associate with the appropriate files. Make sure all permissions are turned on for all files and directories. This can be done through the *change* command in fsm when changing the levels.

Use the *change* command in fsm to change the mandatory levels of the directories. Use the above tables to specify the correct mandatory level for each directory.

Reattach to default session level

For tests a25 and a29:

Create a directory called sl1oss.

Create a file with vi called test_sl1oss.txt in the sl1oss directory.

Issue SAK

Type fsm.

Use the *change* command in fsm to upgrade the level of the home directory to min:oss. Now upgrade the "sl1oss" directory and "test_sl1oss.txt" file to min:oss. Only run Tests a25 and a29 after the other MAC tests have been completed.

Refer to Table 3 in Chapter IV to see what session level to login at through OpenSSH and what command to use on the file with the appropriate levels. For read operations, use *more* followed by the filename. For write operations use *vi* followed by the filename. When trying to save, do a normal ":wq", do not override with an '!'.

When this suite of tests is concluded, use fsm to change the levels of the sl1oxx.txt file and the user's home directory, and the sl1oss directory back to the default levels. This step must be followed before proceeding onto any other tests.

2. DAC POLICY ENFORCEMENT

Login at regular session level as the cherbig user.

Create a directory called "dactests"

Issue SAK

Use fsm to change the mandatory levels of the "dactests" directory to sl1:il3.

Login at sl1:il3 at the console

Create the following files with *vi*, typing a short text message in each:

test_ogarwx.txt

test_ogrwx.txt

test_orwx.txt

test_orw.txt

test_or.txt

test_none.txt

dtest_ogarwx.txt

dtest_ogrwx.txt

dtest_orwxgrw.txt

dtest_orwxgr.txt

dtest_orwx.txt

atest_grwx.txt

Issue the following commands:

chmod 777 test_ogarwx.txt

chmod 770 test_ogrwx.txt

chmod 700 test_orwx.txt

chmod 600 test_orw.txt

chmod 400 test_or.txt

chmod 000 test_none.txt

chmod 777 dtest_ogarwx.txt

chown demo dtest_ogarwx.txt

chmod 770 dtest_ogrwx.txt

chown demo dtest_ogrwx.txt

chmod 760 dtest_orwxgrw.txt

chown demo dtest_orwxgrw.txt

chmod 740 dtest_orwxgr.txt

chown demo dtest_orwxgr.txt

chmod 700 dtest_orwx.txt

chown demo dtest_orwx.txt

chmod 070 atest_grwx.txt

chgrp stop atest_grwx.txt

chown demo atest_grwx.txt

Use *vi* to create a C-program, “test.c”. This program should be the typical “hello world” program.

Compile the program: gcc –c test.c

Build the program: gcc –o test test.o

Issue the following commands:

cp test test_ogarwx

chmod 777 test_ogarwx

cp test test_ogrwx

chmod 770 test_ogrwx

cp test test_orwx

chmod 700 test_orwx

cp test test_orw

chmod 600 test_orw

cp test test_or

chmod 400 test_or

cp test test_none

chmod 000 test_none

cp test dtest_ogarwx

chmod 777 dtest_ogarwx

chown demo dtest_ogarwx

cp test dtest_ogrwx

chmod 770 dtest_ogrwx

chown demo dtest_ogrwx

cp test dtest_orwxgrw

chmod 740 dtest_orwxgr

chown demo dtest_orwxgr

cp test dtest_orwxgr

chmod 760 dtest_orwxgrw

chown demo dtest_orwxgrw

cp test dtest_orwx

chmod 700 dtest_orwx

chown demo dtest_orwx

cp test atest_grwx

chmod 070 atest_grwx

chgrp stop atest_grwx

chown demo atest_grwx.

Refer to Table 5 in Chapter IV for the tests to be performed. Login as the cherbig user at the sl1:il3 network interface. For read operations, use *more* followed by the filename. For write operations use *vi* followed by the filename. When trying to save, do a normal :wq, do not override with an '!'. For execute tests, type "./<executable filename>

3. TPE TESTING WITH FILES CREATED BY OPENSSH

Login at the console at default level as the demo user.

Create a directory called "public_html":

mkdir public_html

Change permissions by:

chmod 755 public_html

Change to that directory

cd public_html

Create the following directories

mkdir unclass

mkdir secret

mkdir topsecret

Issue SAK

Use fsm to change the mandatory levels of the following directories:

unclass = sl1:il3

secret = sl2:il3

topsecret = sl3:il3

Login through OpenSSH at each network and try to create a file in each directory.

Login through the TPE at each level and try to view the files created through the web-browser.

4. TPE TESTING WITH FILES MODIFIED BY OPENSSH

Login at the console, as the demo user, at each network level (sl1:il3, sl2:il3, sl3:il3) and create a file in the respective directory under the "public_html" directory.

Login through OpenSSH at each network and modify the file for the corresponding network.

Login through the TPE at each network level and try to view the files through the web-browser.

5. SINGLE LEVEL LAN – SIMULTANEOUS USER LOGINS

Ensure that there is a ".ssh" directory under each user's home directory on the XTS-400 and that within that directory there is a file named "authorized_keys2" that holds the user's public key. Refer to Appendix D for instructions on key generation and installation.

Connect three clients to the switch or network. Make sure they are on the proper subnet.

Use each client, as a different user, to connect to the same daemon on the same network.

6. MULTIPLE SINGLE LEVEL LANS – SIMULTANEOUS USER LOGINS

Repeat the above procedure, but make sure that each client machine is on a different LAN and connects to a different daemon.

7. PUBLIC-KEY AUTHENTICATION

Connect one client to the network. Any user may be used for this test.

Attempt to login with a valid username and a valid private key with a valid passphrase

Attempt to login with a valid username, a valid private key and the wrong passphrase

Attempt to login with a valid username with the wrong private key file. This can be done in PuTTY by selecting the "auth" category on the left hand side of the configuration window. Next specify a private key file by using the browse button. Make sure that the private key does not correspond to the user's public key stored in the "authorized_key2" file.

Attempt to login with an invalid username. Choose any key file, but make sure that the user does not exist on the system.

8. MISCELLANEOUS TESTS

Login at the console as admin

Set levels to max:max.

Type ua_edit

Add a user to the system.

Edit the /etc/passwd file to include the user's username, user ID, group ID, home directory and shell.

Before creating a directory, try logging in though OpenSSH. This attempt should fail.

After creating a directory and installing the "authorized_keys2" file, use FSM to revoke all permissions on the home directory.

Try logging in through OpenSSH. This attempt should fail.

Use fsm to restore the permissions to the home directory. Use the change command in fsm to set the secrecy level of the home directory to sl2. Change the levels of the following files and directories in the following order: authorized_keys2, id_dsa, id_dsa.pub, .ssh, then the home directory.

Try logging in through OpenSSH. This attempt should fail.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[BAR01] Barrett, D. J., & Silverman, R. E. (2001). SSH: The Secure Shell. Sebastopol, CA: O'Reilly.

[BEL76] Bell, D.E. & La Padula, L.J. (1976). Secure Computer System: Unified Exposition and Multics Interpretation. ESD-TR-75-306. Mitre Corporation, Bedford, MA.

[BIB77] Biba, K.J. (1977). Integrity Considerations for Secure Computer Systems. ESD-TR-76-372. Mitre Corporation, Bedford, MA.

[DIG04] DigitalNet Government Solutions, LLC. (2004). Security Target. Version 1.7 for XTS-400, Version 6.0.E. Available: http://niap.nist.gov/cc-scheme/st/ST-VID3012-ST.pdf. Accessed: 12 December 2004.

[DIG03a] DigitalNet Government Solutions, LLC. (2003). XTS-400: Programmer's Guide. XTDOC0006-02. Herndon, VA.

[DIG03b] DigitalNet Government Solutions LLC. (2003). XTS-400: Trusted Facility Manual. XTDOC0004-02. Herndon, VA.

[DIG03c] DigitalNet Government Solutions, LLC. (2002). XTS-400: User's Manual. XTDOC0005-02. Herndon, VA.

[IRV04] Irvine, C. E., Levin, T. E., Nguyen, T. D., Shifflett, D., Khosalim, J., Clark, P. C., Wong, A., Afinidad, F., Bibighaus, D., & Sears, J. (2004). "Overview of a High Assurance Architecture for Distributed Multilevel Security." Proceedings of the 5th IEEE Systems, Man and Cybernetics Information Assurance Workshop, 38-45.

[LAM74] Lampson, B. W. (1974). "Protection". Proc. Fifth Symposium on Information Sciences and Systems. Reprinted in Operating Systems Review, 8, 1 January 1974. pp.18-24

[SSH04] OpenSSH (2004, September). Available: http://www.openssh.org. Accessed: December 15, 2004

[SSL04] OpenSSL (2004, November). Available: http://www.openssl.org. Accessed: December 15, 2004

[PRO03] Provos, N., Friedl, M., & Honeyman, P. (2003). "Preventing Privilege Escalation". The 12th USENIX Security Symposium.

[SAL75] Saltzer, J. H., & Schroeder, M. D. (1975). "The Protection of Information in an Information System". Fourth ACM Symposium on Operating System Principles.

[STE93] Stevens, W. R. (1993). Advanced Programming in the UNIX Environment. Indianapolis, IN: Addison-Wesley.

[TAT04] Tatham, S. (2004). PuTTY: A Free Telnet/SSH Client. Available: http://www.chiark.greeenend.org.uk/~sgtathum/putty/. Accessed: December 15, 2004

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

3. Dr. Diana Gant
   National Science Foundation
   Arlington, VA

4. Dr. Cynthia E. Irvine
   Naval Postgraduate School
   Monterey, CA

5. Thuy D. Nguyen
   Naval Postgraduate School
   Monterey, CA

6. Chris Herbig
   Civilian, Naval Postgraduate School
   Monterey, CA