Security Target for Oracle Linux 7.3

CSEC Certification ID: CSEC2017014

Version 1.4

Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
USA
Tel.: +1.650.506.7000

Trademarks

Oracle Linux and the Oracle logo are trademarks or registered trademarks of Oracle Corporation in the United States, other countries, or both.

atsec is a trademark of atsec information security GmbH.

Linux is a registered trademark of Linus Torvalds.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Intel, Xeon, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Legal Notice

Table of contents

Index of Tables

Illustration Index

References

CC: Common Criteria for Information Technology Security Evaluation, Version 3.1 Revision 5, April 2017

OSPP: Protection Profile for General Purpose Operating Systems, Version 4.1, 2016-03-09

SSH-EP: Extended Package for Secure Shell, Version 1.0, 2016-02-19

Revision History

Version	Date	Author	Changes
1.3	2019-01-29	Stephan Müller	First public release
1.4	2019-02-27	Stephan Müller	Add NIAP SSH TD references

1 Introduction

1.1 Security Target Identification

Title: Security Target for Oracle Linux 7.3

Version: 1.4

Status: atsec public

Publication Date: 2019-02-27

Author: Stephan Müller, atsec information security GmbH

Certification ID: CSEC2017014

CC-Version: 3.1 Revision 5

Keywords: Operating System, general-purpose Operating Systems

1.2 TOE Identification

The TOE is Oracle Linux 7.3.

Details can be found at the Oracle Linux product website.

1.3 TOE Type

The TOE type is a Linux-based general-purpose operating system.

1.4 TOE Overview

This security target documents the security characteristics of the Oracle Linux distribution (abbreviated with OL throughout this document).

1.4.1 Configurations Defined With this ST

This security target documents the security characteristics of the Oracle Linux distribution.

1.4.2 Overview Description

Oracle Linux is a highly-configurable Linux-based operating system which has been developed to provide a good level of security as required in commercial environments. It also meets all functional requirements of the Operating System Protection Profile OSPP v4.1.

1.4.3 Required Hardware and Software

The following hardware / firmware allows the installation of the TOE:

- x86 64-bit Intel Xeon processors:
  - Oracle Server X7-2

1.4.4 Intended Method of Use

1.4.4.1 General-purpose Computing Environment

The TOE is a Linux-based multi-user multi-tasking operating system. The TOE may provide services to several users at the same time. After successful login, the users have access to a general computing environment, allowing the start-up of user applications, issuing user commands at shell level, creating and accessing files. The TOE provides adequate mechanisms to separate the users and protect their data. Privileged commands are restricted to administrative users.

The TOE is intended to operate in a networked environment with other instantiations of the TOE as well as other well-behaved peer systems operating within the same management domain. All those systems need to be configured in accordance with a defined common security policy.

It is assumed that responsibility for the safeguarding of the user data protected by the TOE can be delegated to human users of the TOE if such users are allowed to log on and spawn processes on their behalf. All user data is under the control of the TOE. The user data is stored in named objects, and the TOE can associate a description of the access rights to that object with each named object.

The TOE enforces controls such that access to data objects can only take place in accordance with the access restrictions placed on that object by its owner, and by administrative users. Ownership of named objects may be transferred under the control of the access control policies implemented by the TOE.

Discretionary access rights (e.g. read, write, execute) can be assigned to data objects with respect to subjects identified with their UID, GID and supplemental GIDs. Once a subject is granted access to an object, the content of that object may be used freely to influence other objects accessible to this subject.

1.4.4.2 Operating Environment

The TOE permits one or more processors and attached peripheral and storage devices to be used by multiple applications assigned to different UIDs to perform a variety of functions requiring controlled shared access to the data stored on the system. With different UIDs proper access restrictions to resources assigned to processes can be enforced using the access control mechanisms provided by the TOE. Such installations and usage scenarios are typical for systems accessed by processes or users local to, or with otherwise protected access to, the computer system.

Note: The TOE provides the platform for installing and running arbitrary services. These additional services are not part of the TOE. The TOE is solely the operating system which provides the runtime environment for such services.

All human users, if existent, as well as all services offered by Oracle Linux are assigned unique user identifiers within the single host system that forms the TOE. This user identifier is used together with the attributes assigned to the user identifier as the basis for access control decisions. Except for virtual machine accesses, the TOE authenticates the claimed identity of the user before allowing the user to perform any further actions. Services may be spawned by the TOE without the need for user-interaction. The TOE internally maintains a set of identifiers associated with processes, which are derived from the unique user identifier upon login of the user or from the configured user identifier for a TOE-spawned service. Some of those identifiers may change during the execution of the process according to a policy implemented by the TOE.

1.4.5 Major Security Features

The primary security features of the TOE are specified as part of section 1.5.2.2.

These primary security features are supported by domain separation and reference mediation, which ensure that the features are always invoked and cannot be bypassed.

1.5 TOE Description

1.5.1 Introduction

Oracle Linux is a general purpose, multi-user, multi-tasking Linux based operating system. It provides a platform for a variety of applications. In addition, virtual machines provide an execution environment for a large number of different operating systems.

The Oracle Linux evaluation covers a potentially distributed network of systems running the evaluated versions and configurations of Oracle Linux as well as other peer systems operating within the same management domain. The hardware platforms selected for the evaluation consist of machines which are available when the evaluation has completed and to remain available for a substantial period of time afterwards.

The TOE Security Functions (TSF) consist of functions of Oracle Linux that run in kernel mode plus a set of trusted processes. These are the functions that enforce the security policy as defined in this Security Target. Tools and commands executed in user mode that are used by an administrative user need also to be trusted to manage the system in a secure way. But as with other operating system evaluations they are not considered to be part of this TSF.

The hardware, the BIOS firmware and potentially other firmware layers between the hardware and the TOE are considered to be part of the TOE environment.

The TOE includes standard networking applications, including applications allowing access of the TOE via cryptographically protected communication channels, such as SSH.

System administration tools include the standard command line tools. A graphical user interface for system administration or any other operation is not included in the evaluated configuration.

The TOE environment also includes applications that are not evaluated, but are used as unprivileged tools to access public system services. For example a network server using a port above 1024 may be used as a normal application running without root privileges on top of the TOE. The additional documentation specific for the evaluated configuration provides guidance how to set up such applications on the TOE in a secure way.

1.5.2 TOE Boundaries

1.5.2.1 Physical

The Target of Evaluation is based on the following system software:

- Oracle Linux in the above mentioned version

The TOE and its documentation are supplied on ISO images distributed via the Oracle Linux web site. The TOE includes a package holding the additional user and administrator documentation.

In addition to the installation media, the following documentation is provided:

- Common Criteria Guide for Oracle Linux 7.3 version 1.0
- Manual pages for all applications, configuration files and system calls

The hardware applicable to the evaluated configuration is listed in section 1.4.3. The analysis of the hardware capabilities as well as the firmware functionality is covered by this evaluation to the extent that the following capabilities supporting the security functionality are analyzed and tested:

- Memory separation capability
- Unavailability of privileged processor states to untrusted user code (like the hypervisor state or the SMM)
- Full testing of the security functionality on all listed hardware systems

1.5.2.2 Logical

The primary security features of the TOE are:

1.5.2.2.1 Auditing

The Lightweight Audit Framework (LAF) is designed to be an audit system making Linux compliant with the requirements from Common Criteria. LAF is able to intercept all system calls as well as retrieving audit log entries from privileged user space applications. The subsystem allows configuring the events to be actually audited from the set of all events that are possible to be audited.

1.5.2.2.2 Cryptographic support

The TOE provides cryptographically secured communication to allow remote entities to log into the TOE. For interactive usage, the SSHv2 protocol is provided. The TOE provides the server side as well as the client side applications. Using OpenSSH, password-based and public-key-based authentication are allowed.

Furthermore, the TOE provides TLS-based communication channels for a cryptographically secured communication with other remote entities. TLS is offered for the key negotiating aspect. The implementations of TLS allow a certificate based authentication of the remote peer (the option for pre-shared keys is disallowed in the evaluated configuration).

Also, the TOE provides confidentiality protected data storage using the device mapper target dm_crypt. Using this device mapper target, the Linux operating system offers administrators and users cryptographically protected block device storage space. With the help of a Password-Based Key-Derivation Function version 2 (PBKDF2) implemented with the LUKS mechanism, a user-provided passphrase protects the volume key which is the symmetric key for encrypting and decrypting data stored on disk. Any data stored on the block devices protected by dm_crypt is encrypted and cannot be decrypted unless the volume key for the block device is decrypted with the passphrase processed by PBKDF2. With the device mapper mechanism, the TOE allows for transparent encryption and decryption of data stored on block devices, such as hard disks.

1.5.2.2.3 Identification and Authentication

User identification and authentication in the TOE includes all forms of interactive login (e.g. using the SSH protocol or log in at the local console) as well as identity changes through the su or sudo command. These all rely on explicit authentication information provided interactively by a user.

The authentication security function allows password-based authentication. For SSH access, public-key-based authentication is also supported.

Password quality enforcement mechanisms are offered by the TOE which are enforced at the time when the password is changed.

Using X.509 certificates, users can also perform authentication.

1.5.2.2.4 Discretionary Access Control

DAC allows owners of named objects to control the access permissions to these objects. These owners can permit or deny access for other users based on the configured permission settings. The DAC mechanism is also used to ensure that untrusted users cannot tamper with the TOE mechanisms.

In addition to the standard Unix-type permission bits for file system objects as well as IPC objects, the TOE implements POSIX access control lists. These ACLs allow the specification of the access to individual file system objects down to the granularity of a single user.

1.5.2.2.5 Security Management

The security management facilities provided by the TOE are usable by authorized users and/or authorized administrators to modify the configuration of TSF.

1.5.2.2.6 Self Protection

The TOE implements self-protection mechanisms that protect the security mechanisms of the TOE as well as software executed by the TOE. The following self-protection mechanisms are implemented and enforced:

- Address Space Layout Randomization for user space code.
- Stack buffer overflow protection using stack canaries.
- Secure Boot ensuring that the boot chain up to and including the kernel together with the boot image (initramfs) is not tampered with.
- Updates to the operating system are only installed after their signatures have been successfully validated.

1.5.2.3 Additional Functions

The TOE provides many more functions and mechanisms. The evaluation ensures that all these additional functions do not interfere with the above mentioned security mechanisms in the evaluated configuration. The mechanisms given in the following list, however, may interfere with the security functionality of the TOE and should be allowed in the evaluated configuration. Therefore, the evaluation assesses the functionality to verify that the impact on the security functionality at most adds further restrictions as outlined below.

- Virtualization support: The TOE offers full virtualization support allowing other operating systems to execute on the TOE. The Linux kernel operates as a hypervisor and the supporting software components like QEMU operate as unprivileged processes. Also the guest operating system executes as an unprivileged application from the view-point of the Linux kernel. The libvirt daemon is allowed to run with privileges of the root user to allow management of the virtual machines.

- Linux Container support: The TOE offers userspace virtualization support via Linux Container. That virtualization support shall be allowed to be used such that it does not interfere with the operation of the security functions. The evaluation ensures that the constraints associated with the use of Linux Containers in the evaluated configuration guide has no adverse impact on the security functionality. In addition, the libvirt daemon is allowed to run with the privileges of the root user to allow management of Linux Containers (note, Linux Containers are not referred to by the term containers used in the remainder of this ST).

Additional mechanisms and functions that would interfere with the operation of the security functions are disallowed in the evaluated configuration and the Evaluation Configuration Guide provides instructions to the administrator on how to disable them. Note: TOE mechanism which provide additional restrictions to the above claimed security functions are allowed in the evaluated configuration. For example, the eCryptFS cryptographic file system provided with the TOE and permitted in the evaluated configuration even though they have not been subject to this evaluation. The eCryptFS provides further restrictions on, for example, the security function of discretionary access control mechanism for file system objects and therefore cannot breach the security functionality as the discretionary access control rules of the "lower" file system are still enforced. The following table enumerates mechanisms that are provided with the TOE but which are excluded from the evaluation:

Functions	**Exclusion discussion**
eCryptFS	eCryptFS are not allowed to be used in the evaluated configuration. The encryption capability provided with this file system is therefore unavailable to any user.
LSM Support	The mandatory access control functionality offered by the Linux Security Module (LSM) framework found in the Linux kernel is not assessed by the evaluation and disabled in the evaluated configuration. All LSM modules such as SELinux, AppArmor, SMACK and others are not assessed as part of the evaluation. The evaluated configuration enables aspects of the LSM though.
GSS-API Security Mechanisms	The GSS-API is used to secure the connection between different audit daemons. The security mechanisms used by the GSS-API, however, is not part of the evaluation. Therefore, A.CONNECT applies to the audit-related communication link.

Table 1: Non-evaluated functionalities

Note: Packages and mechanisms not covered with security claims and subsequent assessments during the evaluation or disabling the respective functionality in the evaluated configuration result from resource constraints during the evaluation as well as the restriction specified in the protection profile but does not imply that the respective package or functionality is implemented insecurely.

1.5.2.4 Configurations

The evaluated configurations are defined as follows:

- The CC evaluated package set must be selected at install time in accordance with the description provided in the Evaluated Configuration Guide and installed accordingly.
- The installation specified by the CC guide allows the installation of two different Linux kernels: the Unbreakable Enterprise Kernel (UEK) as well as the derivative of the Red Hat Enterprise Linux kernel. The administrator is free to choose which kernel is used to boot the system as both kernels are allowed in the evaluated configuration.
- The TOE supports the use of IPv4 and IPv6, both are also supported in the evaluated configuration. IPv6 conforms to the following RFCs:
  - RFC 2460 specifying the basic IPv6 protocol<
  - IPv6 source address selection as documented in RFC 3484 Linux implements several new socket options (IPV6_RECVPKTINFO, IPV6_PKTINFO, IPV6_RECVHOPOPTS, IPV6_HOPOPTS, IPV6_RECVDSTOPTS, IPV6_DSTOPTS, IPV6_RTHDRDSTOPTS, IPV6_RECVRTHDR, IPV6_RTHDR, IPV6_RECVHOPOPTS, IPV6_HOPOPTS, IPV6_{RECV,}TCLASS) and ancillary data in order to support advanced IPv6 applications including ping, traceroute, routing daemons and others. The following section introduces Internet Protocol Version 6 (IPv6). For additional information about referenced socket options and advanced IPv6 applications, see RFC 3542
  - Transition from IPv4 to IPv6: dual stack, and configured tunneling according to RFC 4213.
- The default configuration for identification and authentication are the defined password-based PAM modules as well as public-key based authentication for OpenSSH. Support for other authentication options, e.g. smart card authentication, is not included in the evaluation configuration.
- If the system console is used, it must be subject to the same physical protection as the TOE.

Deviations from the configurations and settings specified with the Evaluated Configuration Guide are not permitted.

The TOE comprises a single system (and optional peripherals) running the TOE software listed. Cluster configurations are not permitted in the evaluated configuration.

1.5.2.5 TOE Environment

Several TOE systems may be interlinked in a network, and individual networks may be joined by bridges and/or routers, or by TOE systems which act as routers and/or gateways. Each of the TOE systems implements its own security policy. The TOE does not include any synchronization function for those policies. As a result a single user may have user accounts on each of those systems with different UIDs, different roles, and other different attributes. (A synchronization method may optionally be used, but it not part of the TOE. The administrator must ensure that the synchronized UIDs to not conflict with the security policy applicable to the TOE.)

If other systems are connected to a network they need to be configured and managed by the same authority using an appropriate security policy that does not conflict with the security policy of the TOE. All links between this network and untrusted networks (e. g. the Internet) need to be protected by appropriate measures such as carefully configured firewall systems that prohibit attacks from the untrusted networks. Those protections are part of the TOE environment.

1.5.2.6 Security Policy Model

The security policy for the TOE is defined by the security functional requirements in chapter 6. The following is a list of the subjects and objects participating in the policy.

Subjects:

- Processes acting on behalf of a human user or technical entity.

Named objects:

- File system objects in the following allowed file systems:
  - Ext4 - standard file system for general data
  - XFS - standard file system for general data
  - VFAT - special purpose file system for UEFI BIOS support mounted at /boot/efi
  - iso9660 - ISO9660 file system for CD-ROM and DVD
  - tmpfs - the temporary file system backed by RAM
  - rootfs - the virtual root file system used temporarily during system boot
  - procfs - process file system holding information about processes, general statistical data and tunable kernel parameters
  - sysfs - system-related file system covering general information about resources maintained by the kernel including several tunable parameters for these resources
  - devpts - pseudoterminal file system for allocating virtual TTYs on demand
  - devtmpfs - temporary file system that allows the kernel to generate character or block device nodes
  - binfmt_misc - configuration interface allowing the assignment of executable file formats with user space applications
  - securityfs - interface for loadable security modules (LSM) to provide tunables and configuration interfaces to user space
  - cgroup - interface for configuring the control groups mechanism provided by the kernel
  - debugfs - interface for accessing low-level kernel data

  Please note that the TOE supports a number of additional virtual (i.e. without backing of persistent storage) file systems which are only accessible to the TSF - they are not or cannot be mounted. All above mentioned virtual file systems implement access decisions based DAC attributes inferred from the underlying process' DAC attributes. Additional restrictions may apply for specific objects in this file system.
- Inter Process Communication (IPC) objects:
  - Semaphores
  - Shared memory
  - Message queues
  - Named pipes

    - UNIX domain socket special files
    - DBUS queues
- Network sockets (irrespective of their type - such as Internet sockets and netlink sockets)
- Storage device objects (covered by dm_crypt - note that such storage device objects may be provided by either block devices or LVM devices)
- cron job queues maintained for each user

TSF data:

- TSF executable code
- Subject meta data - all data used for subjects except data which is not interpreted by the TSF and does not implement parts of the TSF (this data is called user data)
- Named object meta data - all data used for the respective objects except data which is not interpreted by the TSF and does not implement parts of the TSF (this data is called user data)
- User accounts, including the security attributes defined by FIA_ATD.1
- Audit records
- Volume keys for dm_crypt block devices and passphrases protecting the session keys

User data:

- Non-TSF executable code used to drive the behavior of subjects
- Data not interpreted by TSF and stored or transmitted using named objects
- Any code executed within the virtual machine environment as well as any data stored in resources assigned to virtual machines

1.6 Applied Technical Decisions

The ST claims compliance to the claimed protection profile and the extended packages. NIAP issued the following technical decisions which are considered in this ST:

- 0332 – Support for RSA SHA2 host keys
- 0331 – SSH Rekey Testing
- 0305 – Handling of TLS connections with and without mutual authentication
- 0304 – Update to FCS_TLSC_EXT.1.2
- 0246 – Assurance Activity for FIA_UAU.5.2
- 0244 – FCS_TLSC_EXT - TLS Client Curves Allowed
- 0243 – SSH Key-Based Authentication
- 0240 – FCS_COP.1.(1) Platform provided crypto for encryption/decryption
- 0239 – Cryptographic Key Destruction in OS PP
- 0208 – Remote Users in OSPP

- 0163 – Update to FCS_TLSC_EXT.1.1 Test 5.4 and FCS_TLSS_EXT.1.1 Test
- 0107 – FCS_CKM - ANSI X9.31-1998, Section 4.1.for Cryptographic Key Generation
- 0104 – FMT_SMF and FMT_MOF in OS PP

2 Conformance Claims

2.1 Conformance with CC parts 2 and 3

This Security Target is CC Part 2 extended and CC Part 3 extended.

Common Citeria [CC] version 3.1 revision 5 is the basis for this conformance claim.

2.2 Conformance with Packages

This Security Target does not claim conformance with a package.

2.3 Conformance with other Protection Profiles

This Security Target claims exact conformance to:

- [OSPP]
- [SSH-EP]

3 Security Problem Definition

3.1 Threats

Threats to be countered by the TOE are characterized by the combination of an asset being subject to a threat, a threat agent and an adverse action.

3.1.1 Assets

Assets to be protected are:

- Persistent storage objects used to store user data and/or TSF data, where this data needs to be protected from any of the following operations:
  - Unauthorized read access
  - Unauthorized modification
  - Unauthorized deletion of the object
  - Unauthorized creation of new objects
  - Unauthorized management of object attributes
- Transient storage objects, including network data
- TSF functions and associated TSF data
- The resources managed by the TSF that are used to store the above-mentioned objects, including the metadata needed to manage these objects.

3.1.2 Threat Agents

Threat agents are external entities that potentially may attack the TOE. They satisfy one or more of the following criteria:

- External entities not authorized to access assets may attempt to access them either by masquerading as an authorized entity or by attempting to use TSF services without proper authorization.
- External entities authorized to access certain assets may attempt to access other assets they are not authorized to either by misusing services they are allowed to use or by masquerading as a different external entity.
- Untrusted subjects may attempt to access assets they are not authorized to either by misusing services they are allowed to use or by masquerading as a different subject.

Threat agents are typically characterized by a number of factors, such as expertise, available resources, and motivation, with motivation being linked directly to the value of the assets at stake. The TOE protects against intentional and unintentional breach of TOE security by attackers possessing a basic attack potential.

3.1.3 Threats countered by the TOE

T.NETWORK_ATTACK An attacker is positioned on a communications channel or elsewhere on the network infrastructure. Attackers may engage in communications with applications and services running on or part of the OS with the intent of compromise. Engagement may consist of altering existing legitimate communications.

T.NETWORK_EAVESDROP An attacker is positioned on a communications channel or elsewhere on the network infrastructure. Attackers may monitor and gain access to data exchanged between applications and services that are running on or part of the OS.

T.LOCAL_ATTACK An attacker may compromise applications running on the OS. The compromised application may provide maliciously formatted input to the OS through a variety of channels including unprivileged system calls and messaging via the file system.

T.LIMITED_PHYSICAL_ACCESS An attacker may attempt to access data on the OS while having a limited amount of time with the physical device.

3.2 Organizational Security Policies

Organizational security policies are not defined.

3.3 Assumptions

The specific conditions below are assumed to exist in a PP-conformant TOE environment.

3.3.1 Physical aspects

A.PLATFORM The OS relies upon a trustworthy computing platform for its execution. This underlying platform is out of scope of this PP.

3.3.2 Personnel aspects

A.PROPER_USER The user of the OS is not willfully negligent or hostile, and uses the software in compliance with the applied enterprise security policy. At the same time, malicious software could act as the user, so requirements which confine malicious subjects are still in scope.

A.PROPER_ADMIN The administrator of the OS is not careless, willfully negligent or hostile, and administers the OS within compliance of the applied enterprise security policy.

4 Security Objectives

The following sections describe the security objectives of the Operating System Protection Profile.

4.1 Security Objectives for the TOE

The following objectives are defined for the TOE.

O.ACCOUNTABILITY	Conformant OSs ensure that information exists that allows administrators to discover unintentional issues with the configuration and operation of the operating system and discover its cause. Gathering event information and immediately transmitting it to another system can also enable incident response in the event of system compromise.
O.INTEGRITY	Conformant OSs ensure the integrity of their update packages. OSs are seldom if ever shipped without errors, and the ability to deploy patches and updates with integrity is critical to enterprise network security. Conformant OSs provide execution environment-based mitigations that increase the cost to attackers by adding complexity to the task of compromising systems.
O.MANAGEMENT	To facilitate management by users and the enterprise, conformant OSes provide consistent and supported interfaces for their security-relevant configuration and maintenance. This includes the deployment of applications and application updates through the use of platform-supported deployment mechanisms and formats, as well as providing mechanisms for configuration and application execution control.
O.PROTECTED_STORAGE	To address the issue of loss of confidentiality of credentials in the event of loss of physical control of the storage medium, conformant OSs provide data-at-rest protection for credentials. Conformant OSes also provide access controls which allow users to keep their files private from other users of the same system.
O.PROTECTED_COMMS	To address both passive (eavesdropping) and active (packet modification) network attack threats, conformant OSs provide mechanisms to create trusted channels for CSP and sensitive data. Both CSP and sensitive data should not be exposed outside of the platform.

4.2 Security Objectives for the Operational Environment

The following objectives are to be met by the operational environment of the TOE.

OE.PLATFORM	The OS relies on being installed on trusted hardware.
OE.PROPER_USER	The user of the OS is not willfully negligent or hostile, and uses the software within compliance of the applied enterprise security policy. Standard user accounts are provisioned in accordance with the least privilege model. Users requiring higher levels of access should have a separate account dedicated for that use.

OE.PROPER_ADMIN	The administrator of the OS is not careless, willfully negligent or hostile, and administers the OS within compliance of the applied enterprise security policy.

4.3 Rationale for Security Objectives

The following tables provide a mapping of security objectives to the environment defined by the threats, policies and assumptions, illustrating that each security objective covers at least one threat, assumption or policy and that each threat, assumption or policy is covered by at least one security objective.

4.3.1 Security Objectives coverage

Objectives	SPD coverage
O.ACCOUNTABILITY	(Unmapped by [OSPP])
O.INTEGRITY	T.NETWORK_ATTACK, T.LOCAL_ATTACK
O.MANAGEMENT	T.NETWORK_ATTACK, T.NETWORK_EAVESDROP
O.PROTECTED_STORAGE	T.LIMITED_PHYSICAL_ACCESS
O.PROTECTED_COMMS	T.NETWORK_ATTACK, T.NETWORK_EAVESDROP

Table 2: Coverage of security objectives for the TOE

Objectives	SPD coverage
OE.PLATFORM	A.PLATFORM
OE.PROPER_USER	A.PROPER_USER
OE.PROPER_ADMIN	A.PROPER_ADMIN

Table 3: Coverage of security objectives for the TOE environment

4.3.2 Security Objectives sufficiency

Threats	Security Objectives
T.NETWORK_ATTACK	The threat T.NETWORK_ATTACK is countered by O.PROTECTED_COMMS as this provides for integrity of transmitted data. The threat T.NETWORK_ATTACK is countered by O.INTEGRITY as this provides for integrity of software that is installed onto the system from the network. The threat T.NETWORK_ATTACK is countered by O.MANAGEMENT as this provides for the ability to configure the OS to defend against network attack.

Threats	Security Objectives
T.NETWORK_EAVESDROP	The threat T.NETWORK_EAVESDROP is countered by O.PROTECTED_COMMS as this provides for confidentiality of transmitted data. The threat T.NETWORK_EAVESDROP is countered by O.MANAGEMENT as this provides for the ability to configure the OS to protect the confidentiality of its transmitted data.
T.LOCAL_ATTACK	The objective O.INTEGRITY protects against the use of mechanisms that weaken the TOE with regard to attack by other software on the platform.
T.LIMITED_PHYSICAL_ACCESS	The objective O.PROTECTED_STORAGE protects against unauthorized attempts to access physical storage used by the TOE.

Table 4: TOE threats sufficiency

Assumptions	Security Objectives
A.PLATFORM	The operational environment objective OE.PLATFORM is realized through A.PLATFORM.
A.PROPER_USER	The operational environment objective OE.PROPER_USER is realized through A.PROPER_USER.
A.PROPER_ADMIN	The operational environment objective OE.PROPER_ADMIN is realized through A.PROPER_ADMIN.

Table 5: Assumptions sufficiency

5 Extended Components Definition

The definition of all SFRs with the appendix of "_EXT" is supplied by the protection profile. These SFRs are not defined in this section.

6 Security Requirements

All of the following SFRs are derived from the OSPP.

The operations of assignments and selections are marked with bold font. The operation of refinement is marked with strike through (deletion) or italics (addition). Iterations are marked with an ID added to the SFR number.

The following styles of marking operations are applied with this Protection Profile:

- Assignments and selections are marked in bold face font.
- Iterations are marked by appending a suffix to the SFR identification.
- Refinements are marked in bold and italic face font.

6.1 Security Functional Requirements

6.1.1 Cryptographic Support

6.1.1.1 FCS_CKM.1(1) Cryptographic key generation

FCS_CKM.1.1 The OS shall generate asymmetric cryptographic keys in accordance with a specified cryptographic key generation algorithm

RSA schemes using cryptographic key sizes of 2048-bit or greater that meet the following: FIPS PUB 186-4, “Digital Signature Standard (DSS)”, Appendix B.3,

ECC schemes using “NIST curves” P-256, P-384 and P-521 that meet the following: FIPS PUB 186-4, “Digital Signature Standard (DSS)”, Appendix B.4,

FFC schemes using cryptographic key sizes of 2048-bit or greater that meet the following: FIPS PUB 186-4, “Digital Signature Standard (DSS)”, Appendix B.1.

Application Note: The TOE supports the generation of RSA and ECDSA keys for the OpenSSH host key as well as the OpenSSH user keys using the ssh-keygen(1) application. The following CAVS certificates apply:

a) RSA implemented by OpenSSL: 2873

b) ECDSA implemented by OpenSSL: 1417

Application Note: The TOE supports the generation of RSA and ECDSA keys for the host authentication used as part of the TLS protocol using the openssl(1) application.

a) RSA implemented by OpenSSL: 2873

b) ECDSA implemented by OpenSSL: 1417

Application Note: TLS provided with NSS generates the ephemeral Diffie-Hellman keys. The following CAVS certificates apply:

a) ECDSA key generation implemented by NSS: 1528

b) DSA key generation implemented by NSS: 1454

6.1.1.2 FCS_CKM.2(1) Cryptographic Key Establishment

FCS_CKM.2.1 The OS shall implement functionality to perform cryptographic key establishment in accordance with a specified cryptographic key establishment method:

RSA-based key establishment schemes that meets the following: NIST Special Publication 800-56B, “Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography”

and

Elliptic curve-based key establishment schemes that meets the following: NIST Special Publication 800-56A, “Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography”

Finite field-based key establishment schemes that meets the following: NIST Special Publication 800-56A, “Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography”

Application Note: The TOE performs key agreement for the SSHv2 protocols using the OpenSSL library. The following CAVS certificates apply:

a) RSA based key wrapping implemented by OpenSSL: 2873

b) Diffie-Hellman implemented by OpenSSL: CVL 1837

c) EC Diffie-Hellman implemented by OpenSSL: CVL 1837

Application Note: The TOE performs key agreement for the TLS protocols using the NSS library. The following CAVS certificates apply:

a) RSA based key wrapping implemented by NSS: 3044

b) Diffie-Hellman implemented by NSS : CVL 2046

c) EC Diffie-Hellman implemented by NSS : CVL 2046

Application Note As part of the key agreement for SSHv2, the OpenSSH client and server applications implement the key derivation function (KDF) according to SP800-135. The following CAVS certificates apply:

a) SSHv2 KDF: CVL 1870

6.1.1.3 FCS_CKM.4 Cryptographic Key Distribution

FCS_CKM.4.1 The OS shall destroy cryptographic keys in accordance with the specified cryptographic key destruction methods:

For volatile memory, the destruction shall be executed by a single overwrite consisting of zeroes;

For non-volatile memory [that consists of the invocation of an interface provided by the underlying platform that

a) **logically addresses the storage location of the key and performs a single overwrite consisting of zeroes.**

6.1.1.4 FCS_COP.1(1) Cryptographic operation – Encryption/Decryption

FCS_COP.1.1 The OS shall perform encryption/decryption services for data in accordance with a specified cryptographic algorithm

AES-XTS (as defined in NIST SP 800-38E)

AES-CBC (as defined in NIST SP 800-38A)

and

AES-GCM (as defined in NIST SP 800-38D)

AES-CTR (as defined in NIST SP 800-38A) mode

and cryptographic key sizes **128-bit, 256-bit**.

Application Note: The AES-CTR mode is added mandated by [SSH-EP]. The refinement "SSH software" specified in the EP is considered merged into "OS". The selection "[selection: perform, invoke platform-provided]" is not marked as the base PP does not contain this operation.

Application Note: The TOE performs symmetric encryption and decryption for the SSHv2 protocols using the OpenSSL library. The following CAVS certificates apply:

a) AES: 5370

Application Note: The TOE performs symmetric encryption and decryption for the TLS protocols using the NSS library. The following CAVS certificates apply:

a) AES: 5654

Application Note: The TOE performs symmetric encryption and decryption part of the block device encryption using the Linux kernel crypto API. The following CAVS certificates apply:

a) AES: 5402 (UEK), 5409 (RHCK)

6.1.1.5 FCS_COP.1(2) Cryptographic operation – Hashing

FCS_COP.1.1 The OS shall perform hashing services in accordance with a specified cryptographic algorithm SHA-1 and

SHA-256

SHA-384

SHA-512

and message digest sizes 160 bits and

256 bits

384 bits

512 bits

that meet the following: FIPS Pub 180-4.

Application Note: The TOE performs hashing for the SSHv2 protocols using the OpenSSL library. The following CAVS certificates apply:

a) SHA: 4312

Application Note: The TOE performs hashing for the TLS protocols using the NSS library. The following CAVS certificates apply:

a) SHA: 4535

Application Note: The TOE performs hashing to support PBKDF2 to derive the KEK protecting the DEK for the block device encryption mechanism using the libgcrypt library. The following CAVS certificates apply:

a) SHA: 4217

Application Note: The TOE performs hashing to support the ESSIV initialization vector derivation from a sector number for CBC-based disk encryption. The following CAVS certificates apply:

a) SHA: 4331 (UEK), 4341 (RHCK)

6.1.1.6 FCS_COP.1(3) Cryptographic operation – Signing

FCS_COP.1.1 The OS shall perform cryptographic signature services (generation and verification) in accordance with a specified cryptographic algorithm

RSA schemes using cryptographic key sizes of 2048-bit or greater that meet the following: FIPS PUB 186-4, “Digital Signature Standard (DSS)”, Section 4,

ECDSA schemes using “NIST curves” P-256, P-384 and P-521 that meet the following: FIPS PUB 186-4, “Digital Signature Standard (DSS)”Section 5.

Application Note: The TOE performs signature operation for the SSHv2 protocols as well as the trusted update signature verification using the OpenSSL library. The following CAVS certificates apply:

a) RSA: 2873

b) ECDSA: 1417

Application Note: The TOE performs signature operation for the TLS protocols as well as the trusted update signature verification using the NSS library. The following CAVS certificates apply:

a) RSA: 3044

b) ECDSA: 1528

6.1.1.7 FCS_COP.1(4) Cryptographic operation - Keyed-hash Message Authentication

FCS_COP.1.1 The OS shall perform keyed-hash message authentication services in accordance with a specified cryptographic algorithm

SHA-1

SHA-256

SHA-384

SHA-512

with key sizes **key sizes larger than 112 bits** and message digest sizes **160 bits, 256 bits, 384 bits, 512 bits** that meet the following: FIPS Pub 198-1 The Keyed-Hash Message Authentication Code and FIPS Pub 180-4 Secure Hash Standard.

Application Note: The TOE generates MACs for the SSHv2 protocols using the OpenSSL library. The following CAVS certificates apply:

a) HMAC SHA: 3558

Application Note: The TOE generates MACs for the TLS protocols using the NSS library. The following CAVS certificates apply:

a) HMAC SHA: 3767

Application Note: The TOE generates MACs to support PBKDF2 to derive the KEK protecting the DEK for the block device encryption mechanism using the libgcrypt library. The following CAVS certificates apply:

a) HMAC SHA: 3469

6.1.1.8 FCS_RBG_EXT.1 Random Bit Generation

FCS_RBG_EXT.1.1 The OS shall perform all deterministic random bit generation (DRBG) services in accordance with NIST Special Publication 800-90A using

HMAC_DRBG (any),

Hash_DRBG (any),

CTR_DRBG (AES).

FCS_RBG_EXT.1.2 The deterministic RBG used by the OS shall be seeded by an entropy source that accumulates entropy from a **platform-based noise source** with a minimum of **256 bits** of entropy at least equal to the greatest security strength (according to NIST SP 800-57) of the keys and hashes that it will generate.

Application Note: The TOE generates random bits for the SSHv2 protocols using the OpenSSL library. The following CAVS certificates apply:

a) CTR_DRBG: 2079

Application Note: The TOE generates random bits for the TLS protocols using the NSS library. The following CAVS certificates apply:

a) Hash_DRBG: 2284

Application Note: The TOE generates random bits to generate the DEK for the block device encryption mechanism using the libgcrypt library. The following CAVS certificates apply:

a) HMAC_DRBG: 2003

6.1.1.9 FCS_STO_EXT.1 Storage of Sensitive Data

FCS_STO_EXT.1.1 The OS shall implement functionality to encrypt sensitive data stored in non-volatile storage and provide interfaces to applications to invoke this functionality.

6.1.1.10 FCS_TLSC_EXT.1 TLS Client Protocol

FCS_TLSC_EXT.1.1 The OS shall implement TLS 1.2 (RFC 5246) supporting the following cipher suites:

Mandatory cipher suites: TLS_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246

Optional cipher suites:

TLS_DHE_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246

TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5246

TLS_DHE_RSA_WITH_AES_256_CBC_SHA as defined in RFC 5246

TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 as defined in RFC 5246

TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA as defined in RFC 4492

TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289

TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA as defined in RFC 4492

TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289

TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA as defined in RFC 4492

TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289

TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA as defined in RFC 4492

TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289

FCS_TLSC_EXT.1.2 The OS shall verify that the presented identifier matches the reference identifier according to RFC 6125.

FCS_TLSC_EXT.1.3 The OS shall only establish a trusted channel if the peer certificate is valid.

6.1.1.11 FCS_TLSC_EXT.2 - TLS Client Curves Allowed

FCS_TLSC_EXT.2.1 The OS shall present the Supported Elliptic Curves Extension in the Client Hello with the following NIST curves: **secp256r1, secp384r1, secp521r1**.

6.1.2 User Data Protection

6.1.2.1 FDP_ACF_EXT.1 Access Controls for Protecting User Data

FDP_ACF_EXT.1.1 The OS shall implement access controls which can prohibit unprivileged users from accessing files and directories owned by other users.

6.1.2.2 FDP_IFC_EXT.1 Information flow control

FDP_IFC_EXT.1.1 The OS shall **provide an interface which allows a VPN client to protect all IP traffic using IPsec** with the exception of IP traffic required to establish the VPN connection.

6.1.3 Security Management

6.1.3.1 FMT_MOF_EXT.1 Management of security functions behavior

FMT_MOF_EXT.1.1 The TSF shall restrict the ability to perform the function indicated in column 3 of the “Management Functions” table in FMT_SMF_EXT.1.1 to the administrator.

6.1.3.2 FMT_SMF_EXT.1 Extended: Specification of Management Functions

FMT_SMF_EXT.1.1 The TSF shall be capable of performing the following management functions:

Management Function	FMT_SMF_EXT.1	FMT_MOF_EXT.1
Enable/disable screen lock	X	-
Configure screen lock inactivity timeout	X	-
Configure local audit storage capacity	X	X
Configure minimum password Length	X	X
Configure minimum number of special characters in password	X	X
Configure minimum number of numeric characters in password	X	X
Configure minimum number of uppercase characters in password	X	X
Configure minimum number of lowercase characters in password	X	X
Configure remote connection inactivity timeout	X	-
Enable/disable unauthenticated logon	-	-

Management Function	FMT_SMF_EXT.1	FMT_MOF_EXT.1
Configure lockout policy for unsuccessful authentication attempts through **limiting number of attempts during a time period**	X	X
Configure host-based firewall	X	X
Configure name/address of directory server to bind with	-	-
Configure name/address of remote management server from which to receive management settings	-	-
Configure name/address of audit/logging server to which to send audit/logging records	X	X
Configure audit rules	X	X
Configure name/address of network time server	X	X
Enable/disable automatic software update	X	X
Configure WiFi interface	-	-
Enable/disable Bluetooth interface	X	X
Configure USB interfaces	-	-
Enable/disable **no other devices**	X	X
No other management functions	X	X

Table 6: Management Functions

6.1.4 Protection of the TSF

6.1.4.1 FPT_ACF_EXT.1 Access Controls

FPT_ACF_EXT.1.1 The OS shall implement access controls which prohibit unprivileged users from modifying:

- Kernel and its drivers/modules
- Security audit logs
- Shared libraries
- System executables
- System configuration files
- **no other object**

FPT_ACF_EXT.1.2 The OS shall implement access controls which prohibit unprivileged users from reading:

- Security audit logs
- System-wide credential repositories
- **no other object**

6.1.4.2 FPT_ASLR_EXT.1 Address Space Layout Randomization

FPT_ASLR_EXT.1.1 The OS shall always randomize process address space memory locations except for **the Linux kernel, non-Position-Independent-Executable applications, non-Position-Intependent-Code shared libraries**.

6.1.4.3 FPT_SBOP_EXT.1 Stack Buffer Overflow Protection

FPT_SBOP_EXT.1.1 The OS shall be compiled with stack-based buffer overflow protections enabled.

6.1.4.4 FPT_TST_EXT.1 Boot Integrity

FPT_TST_EXT.1.1 The OS shall verify the integrity of the bootchain up through the OS kernel and **no other software component** prior to its execution through the use of **a digital signature using a hardware-protected asymmetric key**.

6.1.4.5 FPT_TUD_EXT.1 Trusted Update

FPT_TUD_EXT.1.1 The OS shall provide the ability to check for updates to the OS software itself.

FPT_TUD_EXT.1.2 The OS shall cryptographically verify updates to itself using a digital signature prior to installation using schemes specified in FCS_COP.1(3).

6.1.4.6 FPT_TUD_EXT.2 Trusted Update for Application Software

FPT_TUD_EXT.2.1 The OS shall provide the ability to check for updates to application software.

FPT_TUD_EXT.2.2 The OS shall cryptographically verify the integrity of updates to applications using a digital signature specified by FCS_COP.1(3) prior to installation.

6.1.5 Audit Data Generation

FAU_GEN.1 Audit data generation

FAU_GEN.1.1 The TSF shall be able to generate an audit record of the following auditable events:

a) Start-up and shutdown of the audit functions;

b) All auditable events for the not specified level of audit; and

c) Authentication events (Success, Failure);

d) Use of privileged, special rights events (Successful and unsuccessful security, audit, and configuration changes);

e) Privilege or role escalation events (Success/Failure);

f) no other event

FAU_GEN.1.2 The TSF shall record within each audit record at least the following information:

a) Date and time of the event, type of event, subject identity (if applicable), and outcome of the event; and

b) For each audit event type, based on the auditable event definitions of the functional components included in the PP/ST, **User identity (if applicable)**

6.1.6 Identification and Authentication

6.1.6.1 FIA_AFL.1 Authentication failure handling

FIA_AFL.1.1 The OS shall detect when **an administrator configurable positive integer within a any range of positive integers** unsuccessful authentication attempts for **authentication based on user name and password** occur related to **authentication on local console, password-based authentication via SSHv2 protocol**.

FIA_AFL.1.2 When the defined number of unsuccessful authentication attempts for an account has been met, the OS shall: **Account Disablement**

6.1.6.2 FIA_UAU.5 Multiple Authentication Mechanisms

FIA_UAU.5.1 The OS shall provide the following authentication mechanisms **authentication based on user name and password, SSH public key-based authentication as specified by the Extended Package for Secure Shell** to support user authentication.

FIA_UAU.5.2 The OS shall authenticate any user's claimed identity according to the **following rule: authentication on the local console is based on user name and password, authentication via the SSHv2 protocol first performs the certificate-based authentication which is followed by the user name and password authentication if the certificate-based authentication was unsuccessful.**

6.1.6.3 FIA_X509_EXT.1 X.509 Certificate Validation

FIA_X509_EXT.1.1 The OS shall implement functionality to validate certificates in accordance with the following rules:

- RFC 5280 certificate validation and certificate path validation.
- The certificate path must terminate with a trusted CA certificate.

- The OS shall validate a certificate path by ensuring the presence of the basicConstraints extension and that the CA flag is set to TRUE for all CA certificates.
- The OS shall validate the revocation status of the certificate using **the Online Certificate Status Protocol (OCSP) as specified in RFC 2560, a Certificate Revocation List (CRL) as specified in RFC 5759, an OCSP TLS Status Request Extension (i.e., OCSP stapling) as specified in RFC 6066**.
- The OS shall validate the extendedKeyUsage field according to the following rules:
  - Certificates used for trusted updates and executable code integrity verification shall have the Code Signing purpose (id-kp 3 with OID 1.3.6.1.5.5.7.3.3) in the extendedKeyUsage field.
  - Server certificates presented for TLS shall have the Server Authentication purpose (id-kp 1 with OID 1.3.6.1.5.5.7.3.1) in the extendedKeyUsage field.
  - Client certificates presented for TLS shall have the Client Authentication purpose (id-kp 2 with OID 1.3.6.1.5.5.7.3.2) in the extendedKeyUsage field.
  - S/MIME certificates presented for email encryption and signature shall have the Email Protection purpose (id-kp 4 with OID 1.3.6.1.5.5.7.3.4) in the extendedKeyUsage field.
  - OCSP certificates presented for OCSP responses shall have the OCSP Signing purpose (id-kp 9 with OID 1.3.6.1.5.5.7.3.9) in the extendedKeyUsage field.
  - (Conditional) Server certificates presented for EST shall have the CMC Registration Authority (RA) purpose (id-kp-cmcRA with OID 1.3.6.1.5.5.7.3.28) in the extendedKeyUsage field.

FIA_X509_EXT.1.2 The OS shall only treat a certificate as a CA certificate if the basicConstraints extension is present and the CA flag is set to TRUE.

6.1.6.4 FIA_X509_EXT.2 X.509 Certificate Authentication

FIA_X509_EXT.2.1 The OS shall use X.509v3 certificates as defined by RFC 5280 to support authentication for TLS and **no other protocols** connections.

6.1.7 Trusted Path/Channel

6.1.7.1 FTP_ITC_EXT.1 Trusted channel communication

FTP_ITC_EXT.1.1 The OS shall use **TLS as conforming to FCS_TLSC_EXT.1, SSH as conforming to the Extended Package for Secure Shell** to provide a trusted communication channel between itself and authorized IT entities supporting the following capabilities: **management server** that is logically distinct from other communication channels and provides assured

identification of its end points and protection of the channel data from disclosure and detection of modification of the channel data.

6.1.7.2 FTP_TRP.1 Trusted Path

FTP_TRP.1.1	The OS shall provide a communication path between itself and **local** users that is logically distinct from other communication paths and provides assured identification of its endpoints and protection of the communicated data from modification and disclosure.
FTP_TRP.1.2	The OS shall permit **the TSF, local users, remote users** to initiate communication via the trusted path.
FTP_TRP.1.3	The OS shall require use of the trusted path for all remote administrative actions.

6.1.8 Extended Package for Secure Shell

6.1.8.1 FCS_SSH_EXT.1 SSH Protocol

FCS_SSH_EXT.1.1	The SSH software shall implement the SSH protocol that complies with RFCs 4251, 4252, 4253, 4254 and **5647, 5656, 6668** as a **client, server**.

6.1.8.2 FCS_SSHC_EXT.1 SSH Protocol - Client

FCS_SSHC_EXT.1.1	The SSH client shall ensure that the SSH protocol implementation supports the following authentication methods as described in RFC 4252: public key-based, and **password-based**.
FCS_SSHC_EXT.1.2	The SSH client shall ensure that, as described in RFC 4253, packets greater than **262144** bytes in an SSH transport connection are dropped.
FCS_SSHC_EXT.1.3	The SSH software shall ensure that the SSH transport implementation uses the following encryption algorithms and rejects all other encryption algorithms: aes128-ctr, aes256-ctr, **aes128-cbc, aes256-cbc, AEAD_AES_128_GCM, AEAD_AES_256_GCM**.
FCS_SSHC_EXT.1.4	The SSH client shall ensure that the SSH transport implementation uses **ssh-rsa, ecdsa-sha2-nistp256** and **ecdsa-sha2-nistp384** as its public key algorithm(s) and rejects all other public key algorithms.
FCS_SSHC_EXT.1.5	The SSH client shall ensure that the SSH transport implementation uses **hmac-sha1, hmac-sha1-96, hmac-sha2-256, hmac-sha2-512** and **AEAD_AES_128_GCM, AEAD_AES_256_GCM** as its data integrity MAC algorithm(s) and rejects all other MAC algorithm(s).
FCS_SSHC_EXT.1.6	The SSH client shall ensure that **diffie-hellman-group14-sha1, ecdh-sha2-nistp256** and **ecdh-sha2-nistp384, ecdh-sha2-nistp521** are the only allowed key exchange methods used for the SSH protocol.
FCS_SSHC_EXT.1.7	The SSH server shall ensure that the SSH connection be rekeyed after **no more than 2^{28} packets have been transmitted** using that key.
FCS_SSHC_EXT.1.8	The SSH client shall ensure that the SSH client authenticates the identity of the SSH server using a local database associating each host name with its

corresponding public key or **no other methods** as described in RFC 4251 section 4.1.

6.1.8.3 FCS_SSHS_EXT.1 SSH Protocol - Server

FCS_SSHS_EXT.1.1 The SSH server shall ensure that the SSH protocol implementation supports the following authentication methods as described in RFC 4252: public key-based, and **password-based**.

FCS_SSHS_EXT.1.2 The SSH server shall ensure that, as described in RFC 4253, packets greater than **262144** bytes in an SSH transport connection are dropped.

FCS_SSHS_EXT.1.3 The SSH server shall ensure that the SSH transport implementation uses the following encryption algorithms and rejects all other encryption algorithms: aes128-ctr, aes256-ctr, **aes128-cbc, aes256-cbc, AEAD_AES_128_GCM, AEAD_AES_256_GCM**.

FCS_SSHS_EXT.1.4 The SSH server shall ensure that the SSH transport implementation uses **ssh-rsa, ecdsa-sha2-nistp256** and **ecdsa-sha2-nistp384** as its public key algorithm(s) and rejects all other public key algorithms.

FCS_SSHS_EXT.1.5 The SSH server shall ensure that the SSH transport implementation uses **hmac-sha1, hmac-sha1-96, hmac-sha2-256, hmac-sha2-512** and **AEAD_AES_128_GCM, AEAD_AES_256_GCM** as its MAC algorithm(s) and rejects all other MAC algorithm(s).

FCS_SSHS_EXT.1.6 The SSH server shall ensure that **diffie-hellman-group14-sha1, ecdh-sha2-nistp256** and **ecdh-sha2-nistp384, ecdh-sha2-nistp521** are the only allowed key exchange methods used for the SSH protocol.

FCS_SSHS_EXT.1.7 The SSH server shall ensure that the SSH connection be rekeyed after **no more than 2^{28} packets have been transmitted** using that key.

6.2 Rationale for Security Functional Requirements

All SFRs are reproduced exactly from the PP and the extended packages. No SFR is added. As the Protection Profile does not provide an SFR rationale, this ST does not require one.

6.3 Security Assurance Requirements

The Protection Profile specifies the Security Assurance Requirements which are not re-iterated in this document.

6.4 Rationale for Security Assurance Requirements

The Security Target claims exact compliance to the Protection Profile, including to the Security Assurance Requirements. As the Protection Profile does not provide an SAR rationale, this ST does not require one.

7 TOE Summary Specification

The following section explains how the security functions are implemented. The different TOE security functions cover the various SFR classes.

7.1 Cryptographic Support

The TOE implements different cryptographic service providers enumerated as documented in the following sections.

Symmetric key material and Diffie-Hellman / EC Diffie-Hellman public and private keys are always considered ephemeral and stored in volatile memory only irrespective of the cryptographic service provider listed below.

Asymmetric key material with the exception of Diffie-Hellman / EC Diffie-Hellman public and private keys is considered to be reused multiple times and is therefore stored on hard disk. The following locations for key material is used:

- OpenSSH (the TOE acts as a sender and recipient):
  - /etc/ssh contains the system-wide keys like host keys. They are generated using ssh-keygen during the first boot after installation.
  - $HOME/.ssh for per-user keys (note, the SSH client applications allows providing a key file name with command line options which implies that the user can specify arbitrary key files). These key files are generated using ssh-keygen invoked by the user. In addition, for key-based authentication, the public key of the authenticating user is generated remotely and added to $HOME/.ssh/authorized_keys.
  - The symmetric session key and the integrity key are derived using the SSH KDF from the shared secret calculated with the Diffie-Hellman / EC Diffie-Hellman operation performed during the handshake of the SSH session establishment.
  - OpenSSH supports the following ciphers:
    - Data protection: AES-128 CBC, AES-256 CBC, AES-128 CTR, AES-256 CTR, AES-128 GCM, AES-256 GCM, HMAC SHA-1, HMAC SHA-256, HMAC SHA-512
    - Authentication: RSA 2048 through 3072, ECDSA with P-256, P-384 and P-521 using SHA-1 or SHA-2
    - Key-agreement: DH with Group 14, DH with domain parameters specified in /etc/ssh/moduli, ECDH with P-256, P-384 and P-521
- NSS (the TOE acts as a sender and recipient):
  - Per default, keys used for the TLS operation are stored in /etc/pki. These keys may be generated using the openssl command. It is also permissible to import key material generated remotely.
  - The symmetric session key and the integrity key are derived using the TLS KDF from the shared secret calculated with the Diffie-Hellman / EC Diffie-Hellman operation performed during the handshake of the TLS session establishment. In addition, the shared secret may be exchanged using RSA key wrapping if the respective TLS cipher

suite is used. All key material is stored in volatile memory of the NSS-consuming applications.

- The certificate verification supports matching of the remote identifier with the certificate's CN (either host full qualified DNS name or IP address), DNS SAN, URI SAN. The TOE does not support certificate pinning.

7.1.1 Linux kernel crypto API

To support cryptographic operations inside the Linux kernel, the kernel crypto API is used. This implementation is used by disk encryption.

The Linux kernel crypto API implements the following ciphers:

- AES with 128 and 256 bits and block chaining modes CBC, XTS.
- SHA-1, SHA-256, SHA-384, SHA-512

In case of decryption errors, the TOE will return an error to the remote entity.

The kernel crypto API clears all RAM buffers holding sensitive data or keys by overwriting the memory with zeros before releasing it.

7.1.2 OpenSSL

The OpenSSL library is used to support the SSHv2 protocol implementation. In addition, OpenSSL supports the generation of RSA and ECDSA key pairs conformant to FIPS 186-4. In addition, OpenSSL provides the generation of Diffie-Hellman and EC Diffie-Hellman public and private key pairs. Also, OpenSSL implements the Diffie-Hellman and EC Diffie-Hellman key agreement.

OpenSSL implements the following ciphers that supports the OpenSSH application:

- RSA key pair generation for key sizes of 2048, and 3072 bits following FIPS 186-4, Appendix B.3.
- ECC key pair generation using NIST P-256, NIST P-384, NIST P-521 following FIPS 186-4, Appendix B4.
- FFC key pair generation with key sizes of 2048, 3072 and 4096 bits following FIPS 186-4 Appendix A.1.
- EC Diffie-Hellman key agreement using NIST P-256, NIST P-384, NIST P-521.
- Diffie Hellman key agreement using key sizes of 2048, 3072, 4096 bits.
- AES with 128 and 256 bits and block chaining modes CBC, GCM, CTR.
- SHA-1, SHA-256, SHA-384, SHA-512 used for signature operations and HMAC operations and offered as a generic service
- RSA signature generation and verification using SHA-1 and SHA-2 together with RSA keys of the size of 2048, and 3072 bits following FIPS 186-4.
- ECDSA signature generation and verification using SHA-1 and SHA-2 using NIST P-256, NIST P-384, NIST P-521.
- HMAC SHA-1, HMAC SHA-256, HMAC SHA-384, HMAC SHA-512
- CTR DRBG with AES 256 core, without prediction resistance, with derivation function

Please note that the HMAC SHA-1-96 cipher used in SSHv2 is a truncation of the HMAC SHA-1 keyed message digest to 96 bits performed by OpenSSH.

In case of decryption errors, the TOE will return an error to the remote entity.

The OpenSSL library clears all RAM buffers holding sensitive data or keys by overwriting the memory with a data pattern before releasing it.

7.1.3 NSS

The NSS shared library enables the cryptographic support in the TLS protocol implementation. All cryptographic primitives are implemented by NSS. In addition, NSS supports the generation of RSA key pairs conformant to FIPS 186-4. In addition, NSS provides the generation of Diffie-Hellman and EC Diffie-Hellman public and private key pairs. Also, NSS implements the Diffie-Hellman and EC Diffie-Hellman key agreement.

NSS implements the following ciphers:

- RSA key pair generation for key sizes of 2048, and 3072 bits following FIPS 186-4, Appendix B.3.
- FFC key pair generation with key sizes of 2048, and 3072 bits following FIPS 186-4 Appendix A.1.
- Diffie Hellman key agreement using key sizes of 2048, and 3072 bits.
- AES with 128 and 256 bits and block chaining modes CBC, GCM.
- SHA-1, SHA-256, SHA-384, SHA-512 used for signature operations, HMAC operations and the DRBG
- RSA signature generation and verification using SHA-1 and SHA-2 together with RSA keys of the size of 2048, and 3072 bits following FIPS 186-4.
- HMAC SHA-1, HMAC SHA-256, HMAC SHA-384, HMAC SHA-512
- Hash DRBG with SHA 256 core, without prediction resistance

NSS supports all TLS cipher suites listed in FCS_TLSC_EXT.1. NSS allows using the following reference identifies to be verified during TLS channel establishment:

- DNS host name or IP address found in Common Name of the X.509 certificate. Wild cards are supported.
- DNS host name found in the SAN for DNS names of the X.509 certificate.
- URI name found in the SAN for URI names of the X.509 certificate.

7.1.4 Libgcrypt

The libgcrypt library is used by the system service configuring and enabling the full disk encryption.

Libgcrypt implements the following ciphers:

- SHA-1, SHA-256, SHA-384, SHA-512 used for HMAC operations
- HMAC SHA-1, HMAC SHA-256, HMAC SHA-384, HMAC SHA-512 used for the PBKDF2 operation and DRBG

- HMAC DRBG with SHA 256 core, without prediction resistance

In case of decryption errors, the TOE will return an error to the remote entity.

The libgcrypt library clears all RAM buffers holding sensitive data or keys by overwriting the memory with zeros before releasing it.

7.1.5 Block Device Encryption Support

The TOE offers block device encryption support where zero or more disk partitions can be encrypted as a whole. Using the block device encryption support, a full-disk encryption (FDE) schema can be achieved. When using FDE, at least the directory /boot must remain in clear.

7.1.5.1 Device Mapper

Logical volume management provides a higher-level view of the disk storage on a computer system than the traditional view of disks and partitions. This gives the system administrator much more flexibility in allocating storage to applications and users.

Storage volumes created under the control of the logical volume manager can be resized and moved around almost at will, although this may need some upgrading of file system tools.

The device mapper implements a framework that allows “targets” to remap access requests. The device mapper is called by the generic block layer before the physical device is accessed.

The information flow between VFS and the physical block device covers the following steps:

1. VFS issues a request to access data on some block device. VFS informs the block layer about which data segments are requested on which block device and potentially hands over the data to be stored in the given segments.
2. The generic block layer receives the request. Instead of accessing the requested data segments on the given block device, the block layer diverts into the device mapper framework, giving it the requested data segments, the data and the block device. The device mapper looks up the device mapper targets defined for the given block device. The device mapper invokes the configured target and gives it requested block device and data segment. The target performs its operation by altering either the block device, the requested data segments and/or the contents of the data segments.
3. The device mapper target relays the potentially altered block device, data segments and data back to the device mapper framework.
4. The block layer uses the new information from the device mapper to perform the requested operation on the block device with the given segment and data.

When data is returned from the block device, the discussed steps are followed in reverse order.

The goal of the device mapper is to allow the device mapper target to alter one or more of the:

1. block device;
2. location of the data segments on the block device;
3. contents of the data

while the request is in transit between the VFS layer and the physical device. This discussion shows that the device mapper operation is fully transparent to the VFS layer. As the device mapper does

not interpret the data that it manages, it is therefore fully agnostic of the VFS data. This ultimately means that the VFS layer and the device mapper do not need to have any knowledge about each other.

The origin of the device mapper lies in implementing a Logical Volume Manager (LVM), allowing administrators to configure different physical block devices to be used as one single disk by VFS and therefore a file system. LVM device mapper targets usually alter the of block device and the location of the data segments on the block device.

Note that the Device Mapper supports stacked targets where the output of one target is the input to another target instead of the input to either the physical device or VFS.

As part of the device mapper framework, many device mapper targets are available. These targets are documented in Documentation/device-mapper/.

7.1.5.2 dm_crypt Target

The dm_crypt target is a device mapper target that is intended to transparently encrypt and decrypt data. Therefore, it is intended to modify the data that is flowing between a physical disk and the VFS. As already mentioned, the device mapper including the dm_crypt target has no knowledge about the meaning of the data they manage. This implies that the dm_crypt target receives a block of data, encrypts or decrypts it and forwards it.

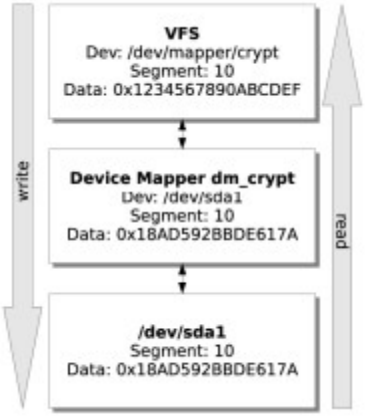

Figure 1: dm_crypt Device Mapper Target Operation

Figure 1 depicts the operation of the dm_crypt target on the data exchange between VFS and the physical disk device.

When VFS issues a read request, the following steps are performed by the kernel:

1. VFS issues the read requests by giving the intended block device (/dev/mapper/crypt in our example) and the data segment (10 in our example).
2. The block layer receives the request and forwards it to the Device Mapper framework. The device mapper has an association between the given block device name /dev/mapper/crypt and the target handling that device. In our example, this device is handled by dm_crypt. Therefore, the Device Mapper framework invokes the appropriate instance of the dm_crypt target covering /dev/mapper/crypt by supplying the read request and the segment number to be read.

3. The dm_crypt target fetches the requested segment from the physical device configured to back /dev/mapper/crypt (/dev/sda1 in our example) – in figure 1, dm_crypt fetches the 16 bytes of data 0x18AD592BBDE617A from /dev/sda1. After fetching the segment, dm_crypt performs a decryption operation with the cipher, and key with the current instance of the dm_crypt target. In our example, dm_crypt decrypts the data read from disk into 0x1234567890ABCDEF.
4. The Device Mapper returns the data 0x1234567890ABCDEF to the calling VFS for initial request.

When VFS issues a write request, the following steps are performed by the kernel:

1. VFS issues the write request by giving the intended block device (/dev/mapper/crypt in our example), the data segment (10 in our example) and the data to be written (0x1234567890ABCDEF in our example).
2. The block layer receives the request and forwards it to the Device Mapper framework. The device mapper has an association between the given block device name /dev/mapper/crypt and the target handling that device. In our example, this device is handled by dm_crypt. Therefore, the Device Mapper framework invokes the appropriate instance of the dm_crypt target covering /dev/mapper/crypt by supplying the write request, the segment number to be written and the data to be written.
3. The dm_crypt target performs an encryption operation on the supplied data with the cipher and key associated with the current instance of the dm_crypt target. The dm_crypt target transforms the original data into 0x18AD592BBDE617A in our example.
4. The dm_crypt target issues a write request to the physical device configured as a backend to /dev/mapper/crypt (/dev/sda1 in our example) and writes the data 0x18AD592BBDE617A to segment 10.

The configuration of the dm_crypt target is performed with the cryptsetup application. That application provides the following information to the kernel when instantiating one dm_crypt target:

- The physical device hosting the encrypted data (/dev/sda1).
- The logical device name (crypt which is used to form /dev/mapper/crypt).
- The cipher type to perform the cryptographic operations.
- The key material required by the cipher.

The discussion of the cryptsetup application in subsequent sections will show where it obtains this information from.

7.1.5.2.1 Initialization Vector Handling

In addition to considering the general idea of the dm_crypt target, the handling of the initialization vector must be considered as well. dm_crypt implements several different flavors for obtaining the IV:

- plain The initialization vector is the 32-bit little-endian version of the sector number, padded with zeros if necessary.
- plain64 The initialization vector is the 64-bit little-endian version of the sector number, padded with zeros if necessary.

- essiv The term “essiv” is the abbreviation for “encrypted sector|salt initial vector", the sector number is encrypted with the bulk cipher using a salt as key. The salt is derived from the cipher key used for encrypting the data with via hashing.
- benbi benbi is the 64-bit “big-endian `narrow block'-count” IV handling method. The IV starts at 1 which is needed for LRW-32-AES and possible other narrow block modes.
- null The initial vector is always zero. This IV handling method provides compatibility with obsolete loop_fish2 devices. It is not recommended for new devices.

7.1.5.2.2 XTS tweak generation

The XTS tweak is generated from the IV provided by dm-crypt. Depending on the chosen IV generation strategy outlined above, different types of IVs are generated by dm-crypt. For XTS, the most common format is plain64.

7.1.5.2.3 Cryptographic Support

The dm-crypt Device Mapper target uses the kernel crypto API which provides the implementation of the cryptographic primitives.

All ciphers offered by the kernel crypto API can be used by dm_crypt. Per default, AES 256 in XTS mode is selected during the initialization of an encrypted block device.

7.1.5.2.4 Sensitive Data Processing

The master volume key, i.e. the AES key used to encrypt the entire partition with is stored on the protected partition in a section called the LUKS header. This AES key, i.e. the data encryption key (DEK), is encrypted with a key-encryption key (KEK). The encryption algorithm used for encrypting the DEK is identical to the encryption algorithm used to encrypt the block device.

The sensitive data process, i.e. the generation of the KEK to decrypt the DEK and to initialize the access to the encrypted block device is handed by the cryptsetup application. Using the LUKS schema, the interaction with the kernel and the key generation can be characterized with figure 2.

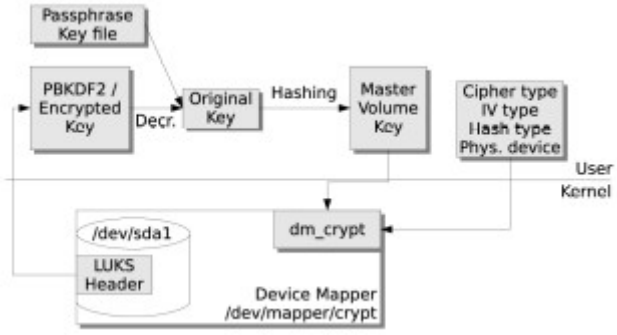

Figure 2: Cryptsetup Operation

1. Cryptsetup is invoked with the physical device hosting the dm-encrypted data – /dev/sda1 in the figure above. Cryptsetup reads the first blocks off the device containing the LUKS header.
2. Cryptsetup identifies the encrypted key in the LUKS header and decrypts it with either the user-provided passphrase or a key file that is provided by the calling user.

3. After the decryption operation, the original key is extracted. This original key is hashed repeatedly to generate the session key used for the encryption and decryption operation.
4. Now, cryptsetup injects the session key along with configuration information about the used cipher, the IV, the hash and the physical device used to store the data (/dev/sda1 in this example) into the kernel to instantiate a device mapping using dm_crypt which is named /dev/mapper/crypt.
5. The kernel has all required data to encrypt all data written into /dev/mapper/crypt and write it to the intended device of /dev/sda1. When reading data out of /dev/mapper/crypt, the kernel is able to decrypt the data from /dev/sda1 and present it to the caller.

When a partition is initialized to encrypt data, the master volume key, i.e. the DEK is generated by cryptsetup using the DRBG of the libgcrypt shared library.

7.1.6 Self Tests

All cryptographic service providers mentioned above are covered with FIPS 140-2 integrity tests.

7.2 User Data Protection

The general policy enforced is that subjects (i.e., processes) are allowed only the accesses specified by the policies applicable to the object the subject requests access to. Further, the ability to propagate access permissions is limited to those subjects who have that permission, as determined by the policies applicable to the object the subject requests access to.

A subject may possess one or more of the following capabilities which provide the following exemptions from the DAC mechanism:

- CAP_DAC_OVERRIDE: A process with this capability is exempt from all restrictions of the discretionary access control and can perform any action desired. For the execution of a file, the permission bit vector of that file must contain at least one execute bit.
- CAP_DAC_READ_SEARCH: A process with this capability overrides all DAC restrictions regarding read and search on files and directories.
- CAP_CHOWN: A process with this capability is allowed to make arbitrary changes to a file's UID or GID.
- CAP_FOWNER: Setting permissions and ownership on objects even if the process' UID does not match the UID of the object.
- CAP_FSETID: Don't clear SUID and SGID permission bits when a file is modified.

DAC provides the mechanism that allows users to specify and control access to objects that they own. DAC attributes are assigned to objects at creation time and remain in effect until the object is destroyed or the object attributes are changed. DAC attributes exist for, and are particular to, each type of named object known to the TOE. DAC is implemented with permission bits and, when specified, ACLs.

The outlined DAC mechanism applies only to named objects which can be used to store or transmit user data. Other named objects are also covered by the DAC mechanism but may be supplemented by further restrictions. These additional restrictions are out of scope for this evaluation. Examples of objects which are accessible to users that cannot be used to store or transmit user data are: virtual

file systems externalizing kernel data structures (such as most of procfs, sysfs, binfmt_misc) and process signals.

During creation of objects, the TSF ensures that all residual contents is removed from that object before making it accessible to the subject requesting the creation.

When data is imported into the TOE (such as when mounting disks created by other trusted systems), the TOE enforces the permission bits and ACLs applied to the file system objects.

During the creation of file system objects, the TOE ensures that new and zeroized memory is used for the newly allocated object. This ensures that any data previously present in the storage area is overwritten.

7.2.1 Permission Bits

The TOE supports standard UNIX permission bits to provide one form of DAC for file system objects in all supported file systems. There are three sets of three bits that define access for three categories of users: the owning user, users in the owning group, and other users. The three bits in each set indicate the access permissions granted to each user category: one bit for read (r), one for write (w) and one for execute (x). Note that write access to file systems mounted as read only (e. g. CD-ROM) is always rejected (the exceptions are character and block device files which can still be written to as write operations do not modify the information on the storage media). The SAVETXT attribute is used for world-writable temp directories preventing the removal of files by users other than the owner.

Each process has an inheritable “umask” attribute which is used to determine the default access permissions for new objects. It is a bit mask of the user/group/other read/write/execute bits, and specifies the access bits to be removed from new objects. For example, setting the umask to “002” ensures that new objects will be writable by the owner and group, but not by others. The umask is defined by the administrator in the /etc/login.defs file or 022 by default if not specified.

7.2.2 Access Control Lists (ACLs)

The TOE provides support for POSIX type ACLs to define a fine grained access control on a user basis. ACLs are supported for all file system objects stored with the following file systems:

- ext4
- XFS
- tmpfs

An ACL entry contains the following information:

- A tag type that specifies the type of the ACL entry
- A qualifier that specifies an instance of an ACL entry type
- A permission set that specifies the discretionary access rights for processes identified by the tag type and qualifier

An ACL contains exactly one entry of three different tag types (called the "required ACL entries” forming the "minimum ACL"). The standard UNIX file permission bits as described in the previous section are represented by the entries in the minimum ACL.

A default ACL is an additional ACL which may be associated with a directory. This default ACL has no effect on the access to this directory. Instead the default ACL is used to initialize the ACL for any file that is created in this directory. If the new file created is a directory it inherits the default ACL from its parent directory. When an object is created within a directory and the ACL is not defined with the function creating the object, the new object inherits the default ACL of its parent directory as its initial ACL.

7.2.3 Special Permission

In addition, the following additional access control bits are processed by the kernel:

- SUID bit: When an executable marked with the SUID bit is executed, the effective UID of the process is changed to the UID of the owner of the file. The SUID bit for file system objects other than files is ignored.
- SGID bit: When an executable marked with the SGID bit is executed, the effective GID of the process is changed to the owning GID of the file. The SGID bit for file system objects other than files is ignored.
- SAVETXT: When a directory is marked with the SAVETXT bit, only the owner of a file system object in that directory can remove it. This bit is commonly used for world-writable directories like /tmp. Only processes with the CAP_FOWNER capability are able to remove the file system object if their UID is different than the owning UID of the file system object.

7.3 Protection of TSF Data

All TSF data is commonly read-only for users based on the DAC permission bits. For sensitive TSF data such as user passwords or private keys, the permission bits do not allow any access by a user.

The following listing enumerates the storage location of TSF data:

- The directory /etc/ contains all configuration files for system-wide configurations. This includes the location for key material.
- The directory /dev/ contains device files to allow interaction between applications and devices.
- The directories /lib, /lib64, /usr/lib and /usr/lib64 contain shared libraries.
- The directory /lib/modules contains kernel drivers.
- The directory /usr/share and all directories in /var except /var/tmp contains TSF data specific to certain shared libraries and applications.
- The directories /bin, /sbin, /usr/sbin, /usr/bin contain TSF executables.
- The directories /sys and /proc contain kernel interfaces usable by applications.
- The directory /boot contains the kernel binary and the boot file system (initramfs).

7.3.1 Stack Buffer Overflow Protection

The GCC compiler is used to generate all system binaries including the kernel. All binaries (i.e. the kernel, kernel modules, executables, shared libraries) are compiled with the option “stack-protector-strong” to add a stack canary and associated verification code during the entry and exit of function frames.

7.3.2 Boot Process

When a computer with Linux is turned on, the operating system is loaded into memory by a special program called a boot loader. A boot loader usually exists on the system's primary hard drive, or other media device, and has the sole responsibility of loading the Linux kernel with its required files or, in some cases, other operating systems, into memory. Each architecture capable of running Linux uses a different boot loader.

7.3.2.1 Boot Loader

A boot loader is a program that resides in the starting sectors of a disk, that is, the Master Boot Record (MBR) of the hard disk. After testing the system during boot, the Basic Input-Output System (BIOS) transfers control to the MBR if the system is set to be booted from there. Then the program residing in MBR gets executed. This program is called the boot loader. Its duty is to transfer control to the operating system, which will then proceed with the boot process.

The boot process consists of the following steps when the CPU is powered on or reset:

1. The firmware performs any hardware initialization steps.
2. The BIOS searches for the boot loader to boot in an order predefined by the firmware setting. Once a valid device is found, the firmware copies the contents of its first sector containing the boot loader into RAM, and starts executing the code just copied.

Every boot loader performs the following general steps to initialize Linux:

1. Loading the kernel image it is configured to load (the actual way of configuring the boot loader is different for each boot loader implementation). The loading process ensures that the kernel image is loaded to a well-defined memory location.
2. Loading the initramfs image it is configured to load. Again, this image is loaded to a well-defined memory location.
3. The kernel is compiled such that the setup function will always be loaded into a well-known memory location. This allows the boot loader to jump to the setup function to transfer control to the kernel.

7.3.2.2 Kernel Boot Process

The following initialization process is followed by the kernel. The details of the boot process are very different for each architecture. However, the following high-level steps are followed by each architecture.

Note, the kernel binary is compressed, except for a small code portion. That portion contains the setup code and the decompression routines in the kernel code to allow the kernel code to decompress itself.

The following steps are performed by the kernel after being loaded by the boot loader.

1. The setup function reinitializes the hardware devices in the computer and sets up the environment for the execution of the kernel program. The setup function initializes and configures hardware devices, such as the keyboard, video card, disk controller, and floating point unit.
2. The kernel is loaded into memory, and if its a compressed image, it is decompressed.

3. The kernel calls a second start-up (e.g. startup_32 on x86) function to set the execution environment for process 0.
4. The kernel initializes the memory management system.
5. The kernel sets the kernel mode stack for process 0.
6. The kernel initializes the provisional Page Tables and enables paging.
7. The kernel sets up the exception handlers.
8. start_kernel completes the kernel initialization by initializing Page Tables, Memory Handling Data Structures, the SLUB allocator, system date, and system time.

7.3.2.3 User Space Boot Process

After the kernel is fully initialized, the user space is started up. There are the following two phases covering the boot process:

- initramfs: This state is intended to perform any initialization work to make the root file system available, such as loading kernel modules with special drivers needed to access the non-volatile storage holding the root file system.
- systemd: This state initializes the entire user space by loading applications and daemons and performs any setup and configuration process necessary to get the system into the operational state.

Both phases are invoked by the Linux kernel where the kernel code uses the execve system call on a hard-coded path.

7.3.2.3.1 Initramfs

The following steps are performed to initialize the initramfs:

1. After the kernel is loaded and initialized, it locates the compressed initramfs image in memory.
2. The Linux kernel uncompresses the image.
3. The kernel performs a loopback mount of the uncompressed initramfs image to mount it as the root file system.
4. The kernel executes the /linuxrc or /sbin/init executable. This is a copy of systemd which executes out of the initramfs.
5. systemd does whatever it needs to do to for setting up the system to allow accessing the root file system based on the configuration.
6. After the systemd application terminates, the kernel unmounts the initramfs, and mounts the root file system pointed to by the “root” kernel command line parameter.

7.3.2.3.2 Systemd

On every Unix system there is one process with the special process identifier 1. It is started by the kernel before all other processes and is the parent process for all those other processes that have nobody else to be child of. Due to that it can do a lot of stuff that other processes cannot do. And it is also responsible for some things that other processes are not responsible for, such as bringing up and maintaining userspace during boot.

systemd starts up and supervises the entire system. It is based around the notion of units. In systemd, a unit refers to a resource that is managed. Each resource is defined by a configuration file called a unit file. Example: a unit avahi.service is the unit file for the Avahi daemon. Units are categorized by the type of their resource. The suffix portion of the unit's file name is the type. The following types are:

- service: these are the most obvious kind of unit: daemons that can be started, stopped, restarted, reloaded. For compatibility with SysV systemd not only supports its own configuration files for services, but also are able to read classic SysV init scripts, in particular we parse the LSB header, if it exists. /etc/init.d is hence not much more than just another source of configuration.
- socket: this unit encapsulates a socket in the file-system or on the Internet. systemd currently supports AF_INET, AF_INET6, AF_UNIX sockets of the types stream, datagram, and sequential packet. In addition, it also supports classic FIFOs as transport. Each socket unit has a matching service unit, that is started if the first connection comes in on the socket or FIFO. Example: nscd.socket starts nscd.service on an incoming connection.
- device: this unit encapsulates a device in the Linux device tree. If a device is marked for this via udev rules, it will be exposed as a device unit in systemd. Properties set with udev can be used as configuration source to set dependencies for device units.
- mount: this unit encapsulates a mount point in the file system hierarchy. systemd monitors all mount points how they come and go, and can also be used to mount or unmount mount-points. /etc/fstab is used here as an additional configuration source for these mount points, similar to how SysV init scripts can be used as additional configuration source for service units.
- automount: this unit type encapsulates an automount point in the file system hierarchy. Each automount unit has a matching mount unit, which is started (i.e. mounted) as soon as the automount directory is accessed.
- target: this unit type is used for logical grouping of units: instead of actually doing anything by itself it simply references other units, which thereby can be controlled together. Examples for this are: multi-user.target, which is a target that basically plays the role of run-level 5 on classic SysV system, or bluetooth.target which is requested as soon as a bluetooth dongle becomes available and which simply pulls in bluetooth related services that otherwise would not need to be started: bluetoothd and obexd and suchlike.
- swap: A unit configuration file whose name ends in ".swap'" encodes information about a swap device or file for memory paging controlled and supervised by systemd.
- path: A unit configuration file whose name ends in ".path'" encodes information about a path monitored by systemd, for path-based activation.
- timer: A unit configuration file whose name ends in ".timer" encodes information about a timer controlled and supervised by systemd, for timer-based activation.
- slice: A unit configuration file whose name ends in ".slice" encodes information about a slice which is a concept for hierarchically managing resources of a group of processes. This management is performed by creating a node in the Linux Control Group (cgroup) tree. Units that manage processes (primarily scope and service units) may be assigned to a specific slice. For each slice, certain resource limits may be set that apply to all processes of

all units contained in that slice. Slices are organized hierarchically in a tree. The name of the slice encodes the location in the tree. The name consists of a dash-separated series of names, which describes the path to the slice from the root slice. The root slice is named, -.slice. Example: foo-bar.slice is a slice that is located within foo.slice, which in turn is located in the root slice -.slice.

- scope: Scope units are not configured via unit configuration files, but are only created programmatically using the bus interfaces of systemd. They are named similar to filenames. A unit whose name ends in ``.scope'' refers to a scope unit. Scopes units manage a set of system processes. Unlike service units, scope units manage externally created processes, and do not fork off processes on its own.

All these units can have dependencies between each other (both positive and negative, i.e. 'Requires' and 'Conflicts'): a device can have a dependency on a service, meaning that as soon as a device becomes available a certain service is started. Mounts get an implicit dependency on the device they are mounted from. Mounts also gets implicit dependencies to mounts that are their prefixes (i.e. a mount /home/lennart implicitly gets a dependency added to the mount for /home) and so on.

In addition to the mentioned core functions, the following support is provided by systemd:

- For each process that is spawned, the following may be controlled: the environment, resource limits, working and root directory, umask, OOM killer adjustment, nice level, IO class and priority, CPU policy and priority, CPU affinity, timer slack, user id, group id, supplementary group ids, readable/writable/inaccessible directories, shared/private/slave mount flags, capabilities/bounding set, secure bits, CPU scheduler reset of fork, private /tmp name-space, cgroup control for various subsystems. Also, an adminitrator can easily connect stdin/stdout/stderr of services to syslog, /dev/kmsg, arbitrary TTYs. If connected to a TTY for input systemd will make sure a process gets exclusive access, optionally waiting or enforcing it.
- Every executed process gets its own cgroup (currently by default in the debug subsystem, since that subsystem is not otherwise used and does not much more than the most basic process grouping), and it is very easy to configure systemd to place services in cgroups that have been configured externally, for example via the libcgroups utilities.
- The native configuration files use a syntax that closely follows the well-known .desktop files. It is a simple syntax for which parsers exist already in many software frameworks.
- As mentioned, systemd provides compatibility with SysV init scripts. systemd takes advantages of LSB and Red Hat chkconfig headers if they are available. If they are not available, ssystemd tries to make the best of the otherwise available information, such as the start priorities in /etc/rc.d. These init scripts are simply considered a different source of configuration, hence an easy upgrade path to proper systemd services is available. Optionally systemd can read classic PID files for services to identify the main pid of a daemon. Note that systemd makes use of the dependency information from the LSB init script headers, and translate those into native systemd dependencies.
- Similarly, systemd reads the existing /etc/fstab configuration file, and consider it just another source of configuration. Using the comment= fstab option you can even mark /etc/fstab entries to become systemd controlled automount points.

- If the same unit is configured in multiple configuration sources (e.g. /etc/systemd/system/avahi.service exists, and /etc/init.d/avahi too), then the native configuration will always take precedence, the legacy format is ignored, allowing an easy upgrade path and packages to carry both a SysV init script and a systemd service file for a while.
- Systemd supports a simple templating/instance mechanism. Example: instead of having six configuration files for six gettys, systemd only has one getty@.service file which gets instantiated to getty@tty2.service and suchlike. The interface part can even be inherited by dependency expressions, i.e. it is easy to encode that a service dhcpcd@eth0.service pulls in avahi-autoipd@eth0.service, while leaving the eth0 string wild-carded.
- For socket activation systemd supports full compatibility with the traditional inetd modes, as well as a very simple mode that tries to mimic launchd socket activation and is recommended for new services. The inetd mode only allows passing one socket to the started daemon, while the native mode supports passing arbitrary numbers of file descriptors. Systemd also supports one instance per connection, as well as one instance for all connections modes. In the former mode systemd names the cgroup the daemon will be started in after the connection parameters, and utilize the templating logic mentioned above for this. Example: sshd.socket might spawn services sshd@192.168.0.1-4711-192.168.0.2-22.service with a cgroup of sshd@.service/192.168.0.1-4711-192.168.0.2-22 (i.e. the IP address and port numbers are used in the instance names. For AF_UNIX sockets we use PID and user id of the connecting client). This provides a nice way for the administrator to identify the various instances of a daemon and control their runtime individually. The native socket passing mode is very easily implementable in applications: if $LISTEN_FDS is set it contains the number of sockets passed and the daemon will find them sorted as listed in the .service file, starting from file descriptor 3 (a nicely written daemon could also use fstat() and getsockname() to identify the sockets in case it receives more than one). In addition systemd sets $LISTEN_PID to the PID of the daemon that shall receive the file descriptors, because environment variables are normally inherited by sub-processes and hence could confuse processes further down the chain.
- Systemd provides compatibility with /dev/initctl to a certain extent. This compatibility is in fact implemented with a FIFO-activated service, which simply translates these legacy requests to D-Bus requests. Effectively this means the old shutdown, poweroff and similar commands from Upstart and sysvinit continue to work with systemd.
- Systemd also provides compatibility with utmp and wtmp.
- Systemd supports several kinds of dependencies between units. After/Before can be used to fix the ordering how units are activated. It is completely orthogonal to Requires and Wants, which express a positive requirement dependency, either mandatory, or optional. Then, there is Conflicts which expresses a negative requirement dependency. Finally, there are three further, less used dependency types.
- Systemd has a minimal transaction system. Meaning: if a unit is requested to start up or shut down we will add it and all its dependencies to a temporary transaction. Then, systemd will verify if the transaction is consistent (i.e. whether the ordering via After/Before of all units is cycle-free). If it is not, systemd will try to fix it up, and removes non-essential jobs from the transaction that might remove the loop. Also, systemd tries to suppress non-essential jobs in the transaction that would stop a running service. Non-essential jobs are those which the

original request did not directly include but which where pulled in by Wants type of dependencies. Finally systemd checks whether the jobs of the transaction contradict jobs that have already been queued, and optionally the transaction is aborted then. If all worked out and the transaction is consistent and minimized in its impact it is merged with all already outstanding jobs and added to the run queue. Effectively this means that before executing a requested operation, systemd will verify that it makes sense, fixing it if possible, and only failing if it really cannot work.

- Systemd records start/exit time as well as the PID and exit status of every process systemd spawns and supervises. This data can be used to cross-link daemons with their data in abrtd, auditd and syslog. Think of an UI that will highlight crashed daemons, and allows to easily navigate to the respective UIs for syslog, abrt, and auditd that will show the data generated from and for this daemon on a specific run.
- Systemd supports re-execution of the init process itself at any time. The daemon state is serialized before the re-execution and de-serialized afterwards. That way systemd provides a simple way to facilitate init system upgrades as well as handover from an initramfs daemon to the final daemon. Open sockets and autofs mounts are properly serialized away, so that they stay connectible all the time, in a way that clients will not even notice that the init system re-executed itself. Also, the fact that a big part of the service state is encoded anyway in the cgroup virtual file system would even allow systemd to resume execution without access to the serialization data. The re-execution code paths are actually mostly the same as the init system configuration reloading code paths, which guarantees that re-execution (which is probably more seldom triggered) gets similar testing as reloading (which is probably more common).
- Systemd re-implements parts of the basic system setup in C and moved it directly into systemd. The historic start scripts are not available any more. Among that is mounting of the API file systems (i.e. virtual file systems such as /proc, /sys and /dev.) and setting of the host-name.
- Server state is introspectable and controllable via D-Bus.
- While systemd emphasizes socket-based and bus-name-based activation, and systemd hence supports dependencies between sockets and services, systemd also supports traditional inter-service dependencies. Systemd supports multiple ways how such a service can signal its readiness: by forking and having the start process exit (i.e. traditional daemonize() behaviour), as well as by watching the bus until a configured service name appears.
- There's an interactive mode which asks for confirmation each time a process is spawned by systemd. This may be enabled by passing systemd.confirm_spawn=1 on the kernel command line.
- With the systemd.default= kernel command line parameter administrators can specify which unit systemd should start on boot-up. Normally administrators specify something like multi-user.target here, but another choice could even be a single service instead of a target, for example out-of-the-box systemd ships a service emergency.service that is similar in its usefulness as init=/bin/bash, however has the advantage of actually running the init system, hence offering the option to boot up the full system from the emergency shell.

After explanation of systemd, a look at the generic boot sequence after the initramfs operation is finished is given. The kernel has mounted the root file system which hosts /sbin/init – note that this

init application is implemented with the systemd framework. This application is the driver of the user space boot process.

The kernel executes /sbin/init which finalizes the boot sequence implemented in the kernel.

1. Before executing the /sbin/init application, it resets the pid table to assign process ID one to the init process.
2. systemd is an event-driven system as described in systemd(8).
3. The boot process driven by systemd is purely based on events. If one event is observed, all tasks associated with that event are executed in parallel. The implemented boot sequence with all events is outlined in bootup(7). The boot process covers the following aspects:
    1. Mounts the /proc special file system.
    2. Mounts the /dev/pts special file system.
    3. Generate the /etc/nologin file early in the boot process.
    4. Saves and restores the system entropy tool for higher quality random number generation.
    5. Configures network interfaces.
    6. Starts the system logging daemons.
    7. Starts the sshd daemon.
    8. Starts the cron daemon.
    9. Probes hardware for setup and configuration.
    10. Removes the /etc/nologin file late in the boot process.

        For more detail about services started at run level 3, refer to the scripts in /etc/rc.d/rc3.d or /etc/rc3.d on a Linux system.

7.3.3 Secure Boot Support[1]

The Unified Extensible Firmware Interface (UEFI) Secure Boot technology ensures that the system firmware checks whether the system boot loader is signed with a cryptographic key authorized by a database of public keys contained in the firmware. With signature verification in the next-stage boot loader and kernel, it is possible to prevent the execution of kernel space code which has not been signed by a trusted key.

A chain of trust is established from the firmware to the signed drivers and kernel modules as follows. The first-stage boot loader, shim.efi, is signed by a UEFI private key and authenticated by a public key, signed by a certificate authority (CA), stored in the firmware database. The shim.efi contains the Oracle public key, “Oracle Secure Boot (CA key 1)”, which is used to authenticate both the GRUB 2 boot loader, grubx64.efi, and the Oracle kernel. The kernel in turn contains public keys to authenticate drivers and modules.

Secure Boot is the boot path validation component of the Unified Extensible Firmware Interface (UEFI) specification. The specification defines:

[1] This section has been derived from https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/System_Administrators_Guide/sec-UEFI_Secure_Boot.html

- a programming interface for cryptographically protected UEFI variables in non-volatile storage,
- how the trusted X.509 root certificates are stored in UEFI variables,
- validation of UEFI applications like boot loaders and drivers,
- procedures to revoke known-bad certificates and application hashes.

UEFI Secure Boot does not prevent the installation or removal of second-stage boot loaders, nor require explicit user confirmation of such changes. Signatures are verified during booting, not when the boot loader is installed or updated. Therefore, UEFI Secure Boot does not stop boot path manipulations, it helps in the detection of unauthorized changes. A new boot loader or kernel will work as long as it is signed by a key trusted by the system.

7.3.3.1 UEFI Secure Boot Support

Oracle Linux 7 includes support for the UEFI Secure Boot feature, which means that Oracle Linux 7 can be installed and run on systems where UEFI Secure Boot is enabled. On UEFI-based systems with the Secure Boot technology enabled, all drivers that are loaded must be signed with a trusted key, otherwise the system will not accept them. All drivers provided by Oracle are signed by one of Oracle's private keys and authenticated by the corresponding Oracle public key in the kernel.

As UEFI Secure Boot support in Oracle Linux 7 is designed to ensure that the system only runs kernel mode code after its signature has been properly authenticated, certain restrictions exist.

GRUB 2 module loading is disabled as there is no infrastructure for signing and verification of GRUB 2 modules, which means allowing them to be loaded would constitute execution of untrusted code inside the security perimeter that Secure Boot defines. Instead, Oracle provides a signed GRUB 2 binary that has all the modules supported on Oracle Enterprise Linux 7 already included.

7.3.4 Trusted Installation and Update

The TOE software is delivered and installed using “Red Hat Packages” (RPMs). These packages are archives of files and contain meta data to allow managing these packages.

As part of the RPM meta data, a signature of the entire RPM file is provided. During installation of an RPM, this signature is verified with an Oracle controlled certificate deployed on the system during installation time. Only if the signature verification is successful, an RPM package is installed. Otherwise it is rejected and not installed.

To ensure that the integrity and authenticity verification enforced by the TOE for individual RPM packages is appropriate, the administrator must ensure that the installation media originates from Oracle. This is performed by downloading the installation media and verifying its integrity and authenticity using signatures and hashes as outlined by the Oracle download server.

Additional software as well as updates are deployed using RPMs. These RPMs originate from Oracle as only Oracle is able to create the signature. However, as each RPM is signed individually, the RPMs can be stored on arbitrary mirror servers which are not under the control of either Oracle or the administrator of the TOE.

Oracle servers distribute update lists pointing to RPMs on a regular basis. These lists indicate whether update RPMs are available. These lists are signed by Oracle as well. However, to prevent rollback attacks, these lists are fetched from Oracle servers by the TOE.

The TOE pulls the latest update lists from Oracle servers nightly and either installs new RPMs automatically or informs the administrator about the presence of update RPMs, depending on the system configuration.

Updates are provided on a quarterly basis and can be reviewed at the Oracle Linux patch website. In case of critical security patches, these patches are published as soon as they become available. Upon availability of these patches, the aforementioned update list is refreshed to allow Oracle Linux to automatically pull this new information.

All security related issues (published) which are flagged as Critical or High based on CVSS ratings should be publicly available within 24 hours of the fix being finalized (including testing). Oracle utilizes CVSS 3.0 specification for scoring CVEs. All other security errata (with severity lower than high) will be evaluated for the next platform release window and prioritized by severity and risk:

- Kernel - quarterly release process for UEK so prioritized will be included in quarterly release. Low risk, low score, may wait to be included in the next major release.
- In addition, there is also a monthly errata for UEK where pending high level security issues can be consolidated.

7.4 Security Management

The security management facilities provided by the TOE are usable by authorized users and/or authorized administrators to modify the configuration of TSF. The configuration of TSF are hosted in the following locations:

- Configuration files (or TSF databases)
- Data structures maintained by the kernel and within the kernel memory

The TOE provides applications to authorized users as well as authorized administrators to perform various administrative tasks. These applications are documented as part of the administrator and user guidance. These applications are either used to modify configuration files or to access parameters controlled and enforced by the kernel via kernel-provided interfaces to user space.

Configuration options are stored in different configuration files. These files are protected using the DAC mechanisms against unauthorized access where usually the root user only is allowed to write to the files. In some special cases (like for /etc/shadow), the file is even readable to the root user only. It is the task of the persons responsible for setting up and administrating the system to ensure that the access control features of the TOE are used throughout the lifetime of the system to protect those databases. These configuration files are accessed using applications which are able to interpret the contents of these configuration files. Each TOE instance maintains its own TSF database. Synchronizing those databases is not performed in the evaluated configuration. If such synchronization is required by an organization it is the responsibility of an administrative user of the TOE to achieve this either manually or with some automated assistance.

To access data structures maintained by the kernel, applications use the kernel-provided interfaces, such as system calls, virtual file systems, netlink sockets, and device files. These kernel interfaces are restricted to authorized administrators or authorized users, if applicable, by either using DAC (for virtual file system objects) or special kernel-internal verification checks for each interface.

All management activities are restricted to the root user. Administrative users assigned to the “wheel” group are eligible to switch to the root user to perform administrative actions using the

sudo application. Therefore, those users are defined to be administrators. This covers all management functions for the mechanisms specified in this chapter.

The following listing enumerates the directories containing security relevant data:

- /etc: System-wide configuration files are stored here.
- /lib, /lib64, /usr/lib and /usr/lib64 contains shared libraries
- /lib/modules: Kernel modules and device drivers are located in this director.
- /var/log/audit: Audit data is stored in this directory.
- /bin, /sbin, /usr/bin, and /usr/sbin contains executables.
- /dev/ contains all device-related interfaces.

7.4.1 Privileges

Privileges to perform administrative actions are maintained by the TOE. These privileges are separated into privileges to act on data or access functionality in user space and in kernel space.

Functionality accessible in user space are applications that can be invoked by users. Also, data accessible in user space is either data maintained with an application or data stored in persistent or transient storage objects. Privileges are controlled by permissions to invoke applications and to access data. For example, the configuration files including the user databases of /etc/passwd and /etc/shadow are accessible to the root user only. Therefore, the root user is given the privilege to perform modifications on this configuration data which constitutes administrative actions.

Functionality and data maintained by the kernel must be accessed using system calls. The kernel implements a privilege check for functions and data that shall not be accessible by normal users. These privileges are controlled with capabilities that can be assigned to processes. If a process is assigned with a capability, it is allowed to request special operations that other processes cannot. To implement consistency with the Unix legacy, processes with the effective UID of zero are implicitly given all capabilities. However, these processes may decide to drop capabilities. Such capabilities are marked by names with the prefix of "CAP_" throughout this document. The Linux kernel implements many more capabilities than mentioned in this document. These unmentioned capabilities protect functions that do not directly cover SFR functionality but need to be protected to ensure the integrity of the system and its resources.

7.5 Audit Data Generation

The Lightweight Audit Framework (LAF) is designed to be an audit system for Linux compliant with the requirements from Common Criteria. LAF is able to intercept all system calls as well as retrieving audit log entries from privileged user space applications. The subsystem allows configuring the events to be actually audited from the set of all events that are possible to be audited. Those events are configured in a specific configuration file and then the kernel is notified to build its own internal structure for the events to be audited.

7.5.1 Audit Functionality

The Linux kernel implements the core of the LAF functionality. It gathers all audit events, analyzes these events based on the audit rules and forwards the audit events that are requested to be audited to the audit daemon executing in user space.

Audit events are generated in various places of the kernel. In addition, a user space application can create audit records which needs to be fed to the kernel for further processing.

The audit functionality of the Linux kernel is configured by user space applications which communicate with the kernel using a specific netlink communication channel. This netlink channel is also to be used by applications that want to send an audit event to the kernel.

The kernel netlink interface is usable only by applications possessing the following capabilities:

- CAP_AUDIT_CONTROL: Performing management operations like adding or deleting audit rules, setting or getting auditing parameters;
- CAP_AUDIT_WRITE: Submitting audit records to the kernel which in turn forwards the audit records to the audit daemon.

Based on the audit rules, the kernel decides whether an audit event is discarded or to be sent to the user space audit daemon for storing it in the audit trail. The kernel sends the message to the audit daemon again using the above mentioned netlink communication channel. The audit daemon writes the audit records to the audit trail. An internal queuing mechanism is used for this purpose. When the queue does not have sufficient space to hold an audit record the TOE switches into single user mode, is halted or the audit daemon executes an administrator-specified notification action depending on the configuration of the audit daemon. This ensures that audit records do not get lost due to resource shortage and the administrator can backup and clear the audit trail to free disk space for new audit logs.

Access to audit data by normal users is prohibited by the discretionary access control function of the TOE, which is used to restrict the access to the audit trail and audit configuration files to the system administrator only.

The system administrator can define the events to be audited from the overall events that the Lightweight Audit Framework using simple filter expressions. This allows for a flexible definition of the events to be audited and the conditions under which events are audited. The system administrator is also able to define a set of user IDs for which auditing is active or alternatively a set of user IDs that are not audited.

The system administrator can select the audited events. Individual files can be configured to be audited by adding them to a watch list that is loaded into the kernel. In addition, audit rules can be specified to generate audit data based on a large number of different attributes, including:

- Subject or user identifiers
- Result of the operation (success/failure)
- Object identity
- Operation performed on an object
- System call number

The complete list of auditable operations can be obtained from the auditctl(8) man page.

The audit system can be configured to take actions if the audit trail is full or reaches a given theshold of disk space. The actions that can be configured include a halting of the system, preventing further auditable actions, notifications to an administrator or the execution of a configured command.

The TOE provides a management application that uses the aforementioned netlink interface. This application is used during boot time to load the audit rules from the configuration file /etc/audit/audit.rules. The audit rules can be modified at runtime of the system.

7.5.2 Audit Trail

An audit record consists of one or more lines of text containing fields in a “keyword=value” tagged format. The following information is contained in all audit record lines:

- Type: indicates the source of the event, such as SYSCALL, PATH, USER_LOGIN, or LOGIN
- Timestamp: Date and time the audit record was generated
- Audit ID: unique numerical event identifier
- Login ID (“auid”), the user ID of the user authenticated by the system (regardless if the user has changed his real and / or effective user ID afterwards)
- Effective user and group ID: the effective user and group ID of the process at the time the audit event was generated
- Success or failure (where appropriate)
- Process ID of the subject that caused the event (PID)
- Hostname or terminal the subject used for performing the operation
- Information about the intended operation

This information is followed by event specific data. In some cases, such as SYSCALL event records involving file system objects, multiple text lines will be generated for a single event, these all have the same time stamp and audit ID to permit easy correlation.

The audit trail is stored in ASCII text. The TOE provides tools for managing ASCII files that can be used for post-processing of audit data. The application ausearch allows selective extraction of records from the audit trail using defined selection criteria. Using the ausearch, the administrator is able to select the information he wants to review. The tools allow the specification of a fine-grained search pattern where each information component can be searched for, including combinations of these patterns.

The audit trail is stored in files which are accessible by root only. If the audit trail fills up and reaches a warning threshold the administrator is notified about reaching the configured level. If the audit trail is full, the audit daemon rejects fetching new audit logs from the kernel to store them into a file. The kernel buffer holding audit messages fills up. When the kernel audit message buffer is full, the kernel suspends every subject that triggered an auditable event until the buffer is cleared again. This way, operations causing auditable events are prevented. In addition, the audit daemon can inform the administrator about the full audit trail, can switch to single user mode or halt the system, depending on the configuration.

7.5.3 Audit Subsystem Implementation

An auditing facility records information about actions that may affect the security of a computer system. In particular, an auditing facility records any action by any user that may represent a breach of system security. For each action, the auditing facility records enough information about those actions to verify the following:

- The user who performed the action
- The kernel object on which the action was performed
- The exact date and time it was performed
- The success or failure of the action
- The identity of the object involved

The TOE includes a comprehensive audit framework called Linux Audit Framework (LAF), which is composed of user-space and kernel-space components. The framework records security events in the form of an audit trail and provides tools for an administrative user. These tools enable the administrator to configure the subsystem and to search for particular audit records, providing the administrator with the ability to identify attempted and realized violations of the system's security policy.

This section describes the design and operation of the audit subsystem at a high level.

7.5.3.1 Audit Components

The following figure illustrates the various components that make up the audit framework and how they interact with each other. In general, there are user-space components and kernel-space components that use a netlink socket for communication. Whenever a security event of interest occurs, the kernel queues a record describing the event and its result to the netlink socket. If listening to the netlink, the audit daemon, auditd, reads the record and writes it to the audit log.

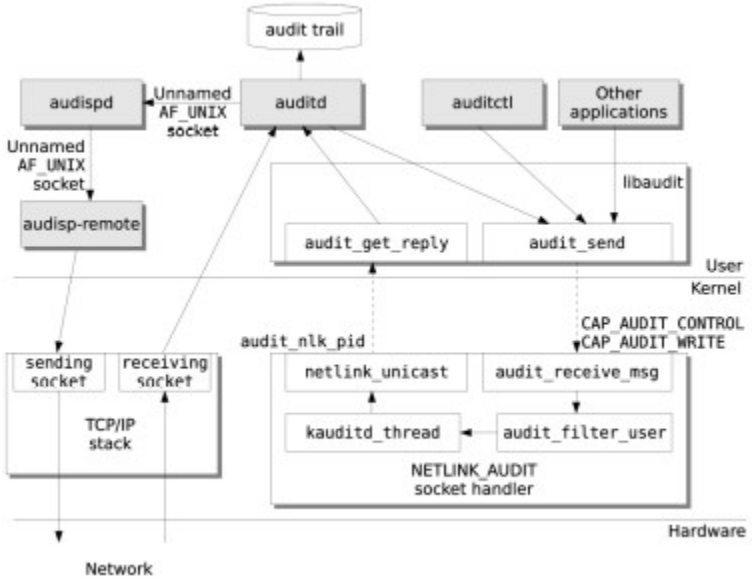

Figure 3: Audit Framework

This section describes the various components of the audit subsystem, starting with the kernel components and then followed by the user-level components.

7.5.3.2 Kernel-Userspace Interface

On top of netlink, there exists the generic netlink family that provides simplified access for less demanding users. This introduces a control for ID management and name resolution, and possesses a new type of safety interface for netlink messages and attributes handling. This interface also features simplified message constructing, validation capabilities, and documentation.

This mechanism also receives user-space commands to control the operation of the audit framework and to set the audit filter rules and file system watch points.

When user space applications want to generate an audit entry, they also have to use the netlink interface to send the message to the kernel.

The kernel checks the effective capabilities of the sender process. If the sender does not possess the right capability (CAP_AUDIT_WRITE), the netlink message is discarded.

As outlined above, the kernel sends the completely formatted audit entry to the audit daemon for storage. The interface the kernel uses is also the same netlink mechanism. However, how does the kernel know to which process it has to send the message to? During startup time, the audit daemon opens the netlink socket and sends a specific control message with its PID to the kernel. That control message registers the PID with the kernel-internal audit mechanisms. From the time of the registering on, this PID is used as the receiver of kernel messages.

7.5.3.3 Task Structure Extensions for Audit

The audit subsystem extends the task structure to potentially include an audit context. By default, on task creation, the audit context is built, unless specifically denied by the per-task filter rules. Then, during system calls, the audit context data is filled. The audit subsystem further extends the audit context to allow for more auxiliary audit information, which might be needed for specific audit events.

The following fields are part of the audit context:

- Login ID: Login ID is the user ID of the logged-in user. It remains unchanged through the setuid or seteuid system calls. Login ID is required to irrefutably associate a user with that user’s actions, even across su(8) calls or use of SUID binaries. The Login ID is set by writing the ID to /proc/<PID>/loginuid, which is performed during login time with the pam_loginuid.so module. The loginuid file is only writable by root and is readable by everyone. The /proc file system triggers the kernel function audit_set_loginuid to set the login uid for the user in the audit context. From then on, this login uid is maintained throughout the session to trace back all operations done in the session to the login user.
- state: State represents the audit state that controls the creation of per-task audit context and filling of system call specifics in the audit context. It can take the following values:
  - AUDIT_DISABLED: Do not create per-task audit_context. No syscall-specific audit records will be generated for the task.
  - AUDIT_SETUP_CONTEXT: Create the per-task audit_context, but don't necessarily fill it in a syscall entry time (i.e., filter instead).

    - AUDIT_BUILD_CONTEXT: Create the per-task audit_context, and always fill it in at syscall entry time. This makes a full syscall record available if some other part of the kernel decides it should be recorded.
    - AUDIT_RECORD_CONTEXT: Create the per-task audit_context, always fill it in at syscall entry time, and always write out the audit record at syscall exit time.
- in_syscall: States whether the process is running in a syscall versus in an interrupt.
- serial: A unique number that helps identify a particular audit record.Along with ctime, it can determine which pieces belong to the same audit record. The (timestamp, serial) tuple is unique for each syscall and it lives from syscall entry to syscall exit.
- ctime: Time at system call entry
- major: System call number
- argv array: The first 4 arguments of the system call.
- name_count: Number of names. The maximum defined is 20.
- audit_names: An array of audit_names structure which holds the data process copied by getname.
- auditable: This field is set to 1 if the audit_context needs to be written on syscall exit.
- pwd: Current working directory from where the task has started.
- pwdmnt: Current working directory mount point. Pwdmnt and pwd are used to set the cwd field of FS_WATCH audit record type.
- aux: A pointer to auxiliary data structure to be used for event specific audit information.
- pid: Process ID.
- arch: The machine architecture.
- personality: The OS personality number.
- Other fields: The audit context also holds the various user and group real, effective, user and file system id’s: uid, euid, suid, fsuid, gid, egid, sgid, fsgid.
- security: An LSM can register its information pertaining its access control rules regarding a process in this void pointer. For SELinux / AppArmor this field returns the label information.

7.5.3.4 System Call Auditing

The audit framework is hooked into the system call glue code of the kernel which is part of the system call interrupt handling routine. Every time a system call is called by a process, the following two states are triggered by the system call glue code:

1. Upon entering the kernel realm but before the actual function implementing the invoked system call is called, a callback to the audit framework is made (audit_syscall_entry). This callback first verifies whether the system call is to be audited based on the audit rules. If it determines that the system call is to be audited, it retrieves the system call number, converts the arguments to an ASCII string to store them with the audit trail and obtains other information like the caller PID and its IDs.

2. After the function implementing the invoked system call completes, but before control is returned to user space, another audit hook (audit_syscall_exit) is called by the system call glue code. This hook code verifies whether there is data generated in step 1. If so, it receives the return code of the system call, stores it together with the initial information to complete the audit entry. This audit entry is now forwarded to the audit daemon via the netlink interface.

To bridge the gap between step one and two, the kernel audit framework uses the audit context registered with the task_struct. This audit context data structure is filled with the information obtained in step 1.

If an architecture implements the system call handling as a kernel-internal thread, the kernel must expect the possibility that the same process can issue another system call before the first is completed. In this case, the kernel uses the audit context pointer of the data structure and generates a double linked list with the pointer to the latest audit context structure as the head of the list. This list is walked during step 2 to find the right entry and merge the exit audit data with the right entry information.

7.5.3.5 Socket call and IPC audit record generation

Some system calls pass an argument to the kernel specifying which function the system call is requesting from the kernel. These system calls request multiple services from the kernel through a single entry point. For example, the first argument to the ipc call specifies whether the request is for semaphore operation, shared memory operation, and so forth. In the same manner, the socketcall system call is a common kernel entry point for the socket system calls. The socketcall and the ipc call are extended to audit the arguments and therefore audit the exact service being performed. Following is a typical flow:

1. The kernel encounters a socket or ipc call.
2. The kernel invokes an audit framework function to collect appropriate data to be used in the auxiliary audit context.
3. The call is processed.
4. On exit the audit record that includes the auxiliary audit information is placed on the netlink.

7.5.3.6 Filesystem auditing

File system auditing is implemented using of the inotify kernel file modification notification system. The audit_init kernel audit subsystem initialization routine registers a vector of inotify operations using the inotify_init function. The operations vector contains the audit_handle_ievent audit subsystem inotify event notification function and the audit_free_parent audit subsystem inotify destroy function. The audit subsystem inotify handle is returned by a successful audit_init call. When audit inotify events occur, audit_handle_ievent updates audit context inode data to reflect changes in watched file status.

When the audit subsystem receives an instruction from auditctl to set a watch on a file system object, the audit_recieve_skb function receives the netlink packet in the kernel. It in turn calls audit_receive_message, which dispatches the appropriate function based upon the operation requested. For audit rule updates, it calls audit_receive_filter. The audit_receive_filter routine calls audit_data_to_entry, which converts the audit data to a watch and calls audit_to_watch to initialize the audit watch data structure, and then calls audit_add_rule. The audit add_rule_function adds the

inotify watch for the watch rule by calling audit_add_watch, which scans the list of active audit inotify watch parents and adds the parent if it does not already exist by calling audit_init_parent. The audit_init_parent function calls inotify_init_watch and inotify_add_watch to initialize the inotify watch and register it with the inotify subsystem. It finally adds the watch to the parent by calling the audit_add_to_parent function, which associates the watch rule with the watch parent.

When a filesystem object the audit subsystem is watching changes, the inotify subsystem calls the audit_handle_ievent function. audit_handle_ievent in turn updates the audit subsystem's watch data for the watched entity.

Permission changes, as well as access and modification of the object security attributes chown, chmod, setxattr, and removexattr, are audited by audit_inode hooks inserted into the system calls. The hooks directly update the inode information in the audit context.

When a watched object is accessed by a system call, the audit subsystem's information about the inode and its watches is updated. A typical sequence of file system operations that generates audit records for a watched object follows these steps:

1. A system call is entered.
2. The system call modifies a watched file's inode information, triggering an inotify event that calls the audit_handle_ievent function with the inotify watch event information, which updates the audit context's inode information. In certain cases, a hooked system call updates the audit context's inode information.
3. At syscall exit, audit_log_exit detects the updated inode information in the audit context and emits PATH and SYSCALL records for the watch event via the audit netlink interface.

7.5.3.7 Auditing of other kernel actions

In addition to the auditing of system calls and file system objects, the audit mechanism inside the kernel provides service functions for any other functional area inside the kernel. These service functions can be used to generate an audit entry with arbitrary contents. That audit entry is forwarded, like any other audit entry, to the auditd daemon for storage.

7.5.3.8 Kernel audit initialization

At kernel startup four lists are created to hold the filter rules. One list is checked at task creation, another is checked at syscall entry time, the third is checked at syscall exit time, and the fourth is used to filter user messages. These lists hold the filter rules set by user-space components. Multiple variables are used to control the operation of audit.

During boot time, the audit enabled flag is set according to audit_default or to the boot parameter audit. No syscall or file system auditing takes place without audit_enabled being set to true.

The file system auditing is initialized by creating the watch lists and the hash table for the file system auditing.

7.5.3.9 Audit Record Format

Each audit record consists of the type of record, a time stamp, login ID, and process ID, along with variable audit data depending on the audit record type. In other words, the record depends on the audit event. Since audit records are written to user-space as soon as they are generated, a complete audit record might be written in several pieces. A time stamp and a serial number pair identify the

various pieces of the audit records. The timestamp of the record and the serial number are used by the user-space daemon to determine which pieces belong to the same audit record. The tuple is unique for each syscall and lasts from syscall entry to syscall exit. The tuple is composed of the timestamp and the serial number.

Each audit record for system calls contains the system call return code, which indicates if the call was successful or not. The following table lists security-relevant events for which an audit record is generated on the TOE.

Event description	**LAF audit events**
Startup and shutdown of audit functions	DAEMON_START, DAEMON_END, generated by auditd
Modification of audit configuration file	DAEMON_CONFIG, DAEMON_RECONFIG generated by auditd. Syscalls open, link, unlink, rename, truncate, (write access to configuration files)
Successful and unsuccessful file read/write	Syscall open
Audit storage space exceeds a threshold	Space_left_action, admin_space_left_action configuration parameters for auditd.
Audit storage space failure	Disk_full_action, disk_error_action configuration parameters for auditd.
Operation on file system objects and IPC objects	system calls accessing the objects
Rejection or acceptance by the TSF of any tested secret.	Audit record type: USER_AUTH from PAM Framework and audit record type: USER_CHAUTHTOK
Use of identification and authentication mechanism	Audit record type: USER_AUTH, USER_CHAUTHTOK from PAM framework.
Success and failure of binding user security attributes to a subject (e.g. success and failure to create a subject)	Audit record type: LOGIN from pam_login.so module. Syscalls: fork and clone.
All modification of subject security values	Syscalls chmod, chown, setxattr, msgctl, semctl, shmctl, removexattr, truncate
Modifications of the default setting of permissive of restrictive rules	Syscalls umask, open
Modification of TSF data	System calls to access file system objects; audit record type: USER_CHAUTHTOK
Changes to system time	Syscall settimeofday, adjtimex; execution of hwclock and access to /dev/rtc

Table 7: LAF Audit Events

7.5.3.10 Auditing Support for OpenSSH

The OpenSSH server generates audit records for the following operations:

- The audit records contain an identifier that the sshd process generated the audit records and therefore implicitly identifying the used communication protocol.
- Origin of the communication channel by logging the remote IP and remote port are logged.
- Indication of a success establishment of a connection is logged. Note, the absence of that log entry indicates a failure of establishing a communication channel.
- Indication when a connection is terminated is logged.
- Authentication of a user (success and failure) including the user name is logged.
- Authentication type is logged (such as password-based or key-based authentication).
- The OpenSSH server logs cryptographic information of key exchange mechanism and the used user or host based authentication mechanisms. In addition, the server logs when a new ephemeral session key is established.
- If the server executes a command, this command will be logged.

7.5.3.11 Time Stamp Maintenance

The Linux kernel maintains various time stamps which has the following properties:

- Time with a resolution in seconds is available to user space. The time is obtained from the firmware or hardware at boot time.
- Time with a nanosecond resolution since the system started.
- Time with a microsecond resolution since Epoch (01.01.1970).

The auditing mechanism uses the available time information to add a time stamp to each audit record.

The configuration files including the auditd.conf and the audit.rules files for the audit framework covering all management aspects are writable by the root user only.

7.6 Identification and Authentication

User identification and authentication in the TOE includes all forms of interactive login (e.g. using the SSH protocol or log in at the local console) as well as identity changes through the su and sudo commands. These all rely on explicit authentication information provided interactively by a user. In addition, the key-based authentication mechanism of the OpenSSH server is another form of of authentication.

7.6.1 PAM-based identification and authentication mechanisms

When a user possesses an identity in a system in the form of a login ID, that user has Identification. Identification establishes user accountability and access restrictions for actions on a system. Authentication is verification that the user’s claimed identity is valid, and is implemented through a user password at login time.

All discretionary access-control decisions made by the kernel are based on the process's user ID established at login time and all mandatory access control decisions made by the kernel are based on the process domain established through login, which make the authentication process a critical component of a system.

The Linux system implements identification and authentication through a set of trusted programs and protected databases. These trusted programs use an authentication infrastructure called the Pluggable Authentication Module (PAM). PAM allows different trusted programs to follow a consistent authentication policy. PAM provides a way to develop programs that are independent of the authentication scheme. These programs need authentication modules to be attached to them at run-time in order to work. Which authentication module is to be attached is dependent upon the local system setup and is at the discretion of the local system administrator.

Linux uses a suite of libraries called the "Pluggable Authentication Modules" (PAM) that allow an administrative user to choose how PAM-aware applications authenticate users. The TOE provides PAM modules that implement all the security functionality to:

- Provides login control and establishing all UIDs, GIDs and login ID for a subject
- Ensure the quality of passwords
- Enforce limits for accounts (such as the number of maximum concurrent sessions allowed for a user)
- Enforce the change of passwords after a configured time including the password quality enforcement
- Enforcement of locking of accounts after failed login attempts.
- Restriction of the use of the root account to certain terminals
- Restriction of the use of the su and sudo commands

The login processing sets the real, file system effective and login UID as well as the real, effective, file system GID and the set of supplemental GIDs of the subject that is created. It is of course up to the client application usually provided by a remote system to protect the user's entry of a password correctly (e. g. provide only obscured feedback).

During login processing, the user is shown a banner. After successful authentication, the login time is recorded.

When configuring the OpenSSH server, the administrator is allowed to enable SSH key-based authentication in addition or instead of the username/password based authentication. When a user can successfully authenticate using the SSH key-based authentication based on a private SSH key in his possession, the TOE grants the user access.

After a successful identification and authentication, the TOE initiates a session for the user and spawns the initial login shell as the first process the user can interact with. The TOE provides a mechanism to lock a session either automatically after a configurable period of inactivity for that session or upon the user's request.

The TOE ensures that the memory used for the authentication operation is cleared before the authentication takes place. This ensures that previously entered credentials are not re-used for a new authentication operation.

When a new user is created, a complete new entry is added to the credential database. This ensures that previously existing credentials are not reused for the newly added user.

After successful authentication, a new process is spawned where the spawned process is identified by the "shell" entry in the credential store (/etc/passwd). This new process is spawned with the UID associated to the user in the credential store, In addition, the new process is spawned with the primary GID as well as supplemental GIDs defined by the credential store for the user (/etc/group). The capabilities are initially set as follows: if the UID of the user is 0, all capabilities are assigned to the newly spawned process. Otherwise no capabilities are assigned.

7.6.1.1 Pluggable Authentication Module

PAM is responsible for the identification and authentication subsystem. PAM provides a centralized mechanism for authenticating all services. PAM allows for limits on access to applications and alternate, configurable authentication methods. For more detailed information about PAM, see the PAM project Web site at http://www.kernel.org/pub/linux/libs/pam.

PAM consists of a set of shared library modules, which provide appropriate authentication and audit services to an application. Applications are updated to offload their authentication and audit code to PAM, which allows the system to enforce a consistent identification and authentication policy, as well as to generate appropriate audit records. The following programs are enhanced to use PAM:

- login
- passwd
- su, sudo
- useradd, usermod, userdel
- groupadd, groupmod, groupdel
- sshd
- chage
- chfn
- chsh

A PAM-aware application generally goes through the following steps:

1. The application makes a call to PAM to initialize certain data structures. With the initialization, the calling application provides a name to PAM which ultimately is used to find the configuration file of the authentication stack configuration in /etc/pam.d/. Usually, that name equals the application name.
2. The PAM module locates the configuration file for that application from /etc/pam.d/application_name and obtains a list of PAM modules necessary for servicing that application. If it cannot find an application-specific configuration file, then it uses /etc/pam.d/other.
3. Depending on the order specified in the configuration file, PAM loads the appropriate modules for the PAM operation requested by the calling application (i.e. PAM provides one call back for each module type – the module type is consistent with the “auth”, “session”, “password” and “account” sections in the PAM configuration files.

4. The authentication module code performs the requested operation depending on the module type. The module may require input from the user. Note: a module may perform operations which hardly have anything to do with authentication, but whose operations are necessary to set up the user environment.
5. Each authentication module performs its action and relays the result back to the application.
6. The PAM library is modified to create a USER_AUTH type of audit record to note the success or failure from the authentication module.
7. The application takes appropriate action based on the aggregate results from all authentication modules.

7.6.1.2 PAM Modules

Linux is configured to use the following PAM modules – each PAM module used in the evaluated configuration is accompanied by a man page that provides additional information:

- pam_unix.so Supports all four module types, and provides standard password-based authentication. pam_unix.so uses standard calls from the system libraries to retrieve and set account information as well as to perform authentication. Authentication information about Linux is obtained from the /etc/passwd and /etc/shadow files. To perform the authentication, the pam_unix module calls the unix_chkpwd helper program. This application hashes the user-provided password with the hash algorithm specified for that user in /etc/shadow, uses the salt stored for the user in /etc/shadow and compares the generated hash with the hash stored in /etc/shadow for the user. If both hashes match, the user is authenticated. Otherwise, the user is denied access.
- pam_stack.so pam_stack.so module performs normal password authentication through recursive stacking of modules. For example, the argument service=system-auth passed to the pam_stack.so module indicates that the user must pass through the PAM configuration for system authentication, found in /etc/pam.d/system-auth.
- pam_passwdqc.so Performs additional password strength checks. For example, it rejects passwords such as “1qaz2wsx” that follow a pattern on the keyboard. In addition to checking regular passwords it offers support for passphrases and can provide randomly generated passwords.
- pam_env.so Loads a configurable list of environment variables, and is configured with the /etc/security/pam_env.conf file.
- pam_shells.so Authentication is granted if the user’s shell is listed in /etc/shells. If no shell is in /etc/passwd (empty), then /bin/sh is used. It also checks to make sure that /etc/shells is a plain file and not world-writable.
- pam_limits.so This module imposes user limits on login. It is configured using the /etc/security/limits.conf file. No limits are imposed on UID 0 accounts.
- pam_rootok.so This module is an authentication module that performs one task: if the id of the user is 0, then it returns PAM_SUCCESS. With the “sufficient” control flag, it can be used to allow password-free access to some service for root.

- pam_xauth.so This module forwards xauth cookies from user to user. Primitive access control is provided by ~/.xauth/export in the invoking user's home directory, and ~/.xauth/import in the target user's home directory.
- pam_wheel.so Returns successful if the user to be authenticated is part of the wheel group. First, the module checks for the existence of a wheel group. Otherwise, the module defines the group with group ID 0 to be the wheel group.
- pam_nologin.so Provides standard UNIX nologin authentication. If the /etc/nologin file exists, only root is allowed to log in; other users are turned away with an error message (and the module returns PAM_AUTH_ERR or PAM_USER_UNKNOWN). All users (root or otherwise) are shown the contents of /etc/nologin.
- pam_loginuid.so Sets the audit uid for the process that was authenticated.
- pam_securetty.so Provides standard UNIX securetty checking, which causes authentication for root to fail unless the calling program has set PAM_TTY to a string listed in the /etc/securetty file. For all other users, pam_securetty.so succeeds.
- pam_faillock.so Keeps track of the number of login attempts made and denies access based on the number of failed attempts, which is specified as an argument to pam_faillock.so module. This is addressed at the “account” module type. The pam_faillock program allows administrative users to examine and control the pam_faillock PAM module's tally file, such as reset.
- pam_tally2.so Keeps track of the number of login attempts made and denies access based on the number of failed attempts, which is specified as an argument to pam_tally2.so module. This is addressed at the “account” module type. The pam_tally2 program allows administrative users to examine and control the pam_tally2 PAM module's tally file such as reset.
- pam_listfile.so Allows the use of ACLs based on users, ttys, remote hosts, groups, and shells.
- pam_deny.so Always returns a failure.
- pam_cracklib.so The action of this module is to prompt the user for a password and check its strength against a system dictionary and a set of rules for identifying poor choices. The first action is to prompt for a single password, check its strength and then, if it is considered strong, prompt for the password a second time (to verify that it was typed correctly on the first occasion). All being well, the password is passed on to subsequent modules to be installed as the new authentication token.
- pam_systemd.so pam_systemd registers user sessions with the systemd login manager systemd-logind.service(8), and hence the systemd control group hierarchy.

7.6.1.3 User Identity Changing

When switching identities, the real, file system and effective user ID and real, file system and effective group ID are changed to the one of the user specified in the command (after successful authentication as this user).

Note: The login ID is not retained for the following special case:

1. User A logs into the system.

2. User A uses su to change to user B.
3. User B now edits the cron or at job queue to add new jobs. This operation is appropriately audited with the proper login ID.
4. Now when the new jobs are executed as user B, the system does not provide the audit information that the jobs are created by user A.

The su command invokes the common authentication mechanism to validate the supplied authentication.

Users can change their identity (i.e., switch to another identity) using one of the following commands provided with the TOE.

7.6.1.3.1 su command

The su command is intended for a switch to a another identity that establishes a new login session and spawns a new shell with the new identity. When invoking su, the user must provide the credentials associated with the target identity - i.e. when the user wants to switch to another user ID, it has to provide the password protecting the account of the target user.

The primary use of the su command within the TOE is to allow appropriately authorized individuals the ability to assume the root identity to perform administrative actions. In this system the capability to login as the root identity has been restricted to defined terminals only. In addition the use of the su command to switch to root has been restricted to users belonging to a special group. Users that don’t have access to a terminal where root login is allowed and are not member of that special group will not be able to switch their real, file system and effective user ID to root even if they would know the authentication information for root. Note that when a user executes a program that has the setuid bit set, only the effective user ID and file system ID are changed to that of the owner of the file containing the program while the real user ID remains that of the caller. The login ID is neither changed by the su command nor by executing a program that has the setuid or setgid bit set as it is used for auditing purposes.

7.6.1.3.2 sudo command

The sudo command is intended for giving users permissions to execute commands with another user identity. When invoking sudo, the user has to authenticate with this credentials.

Sudo is associated with sophisticated ruleset that can be engaged to specify which:

- source user ID
- originating from which host
- can access a command, a command with specific configuration flags, or all commands within a directory
- with which new user identity.

7.6.2 Authentication Data Management

Each TOE instance maintains its own set of users with their passwords and attributes. Although the same human user may have accounts on different servers interconnected by a network and running an instantiation of the TOE, those accounts and their parameter are not synchronized on different TOE instances. As a result the same user may have different user names, different user Ids, different

passwords and different attributes on different machines within the networked environment. Existing mechanism for synchronizing this within the whole networked system are not subject to this evaluation.

Each TOE instance within the network maintains its own administrative database by making all administrative changes on the local TOE instance. System administration has to ensure that all machines within the network are configured in accordance with the requirements defined in this Security Target.

The file /etc/passwd contains for each user the user’s name, the id of the user, an indicator whether the password of the user is valid, the principal group id of the user and other (not security relevant) information. The file /etc/shadow contains for each user a hash of the user's password, the userid, the time the password was last changed, the expiration time as well as the validity period of the password and some other information that are not subject to the security functions as defined in this Security Target. Users are allowed to change their passwords by using the passwd command. This application is able to read and modify the contents of /etc/shadow for the user’s password entry, which would ordinarily be inaccessible to a non-privileged user process. Users are also warned to change their passwords at login time if the password will expire soon, and are prevented from logging in if the password has expired.

The time of the last successful logins is recorded in the directory /var/log/faillock where one file per user is kept.

The TOE displays informative banners before or during the login to users. The banners can be specified with the files /etc/issue for log ins via the physical console or /etc/issue.net for remote log ins, such as via SSH. When performing a log in on the physical console, the banner is displayed above the username and password prompt. For log ins via SSH, the banner is displayed to the remote peer during the SSH-session handshake takes place. The remote SSH client will display the banner to the user. When using the provided OpenSSH client, the banner is displayed when the user instructs the OpenSSH client to log into the remote system.

Users can change their own password. Only administrators can add or delete users or change their properties.

7.6.3 SSH Key-Based Authentication

In addition to the PAM-based authentication outlined above, the OpenSSH server is able to perform a key-based authentication. When a user wants to log on, instead of providing a password, the user applies his SSH key. After a successful verification, the OpenSSH server considers the user as authenticated and performs the PAM-based operations as outlined above.

To establish a key-based authentication, a user first has to generate an RSA, or ECDSA key pair. The private part of the key pair remains on the client side. The public part is copied to the server into the file .ssh/authorized_keys which resides in the home directory of the user he wants to log on as. When the login operation is performed the SSHv2 protocol tries to perform the "publickey" authentication using the private key on the client side and the public key found on the server side. The operations performed during the publickey authentication is defined in RFC 4252 chapter 7.

Users have to protect their private key part the same way as protecting a password. Appropriate permission settings on the file holding the private key is necessary. To strengthen the protection of the private key, the user can encrypt the key where a password serves as key for the encryption operation. See ssh-keygen(1) for more information.

7.6.4 Session Locking

The TOE uses the screen(1) application which locks the current session of the user either after an administrator-specified time of inactivity or upon the user's request.

To unlock the session, the user must supply his password. Screen uses PAM to validate the password and allows the user to access his session after a successful validation.

7.6.5 X.509 Certificate Validation

X.509 certificate validation is implemented by NSS supporting the TLS protocol.

RFC 5280 defines the X.509 certificate and validation mechanism. With the following list, functions that are either defined optional or that are implemented differently than specified in RFC 5280 are listed. The list uses the section numbers out of RFC 5280.

RFC 5280 Reference	Description	Implementation Details
4.1.1.2	Signature algorithms	RSA, and ECDSA.
4.1.2.1	X.509 Version to be supported	X.509 version 3 when generating certificates. It also supports version 2 certificate for validation.
4.1.2.4	Supported attributes	All attributes listed in the RFC are supported.
4.1.2.6	Subject field format	All recommended attributes allowed for the subject field can be processed. Comparison rules for unfamiliar attribute types are implemented. TelexString, BMPString, and UniversalString are not supported.
4.1.2.8	Unique ID	Support for the handling of unique identifiers.
4.2	Certificate extensions	Handling of authority, subject key, and policy mapping extensions is supported.
4.2.1.1	Public key ID	The keyIdentifier field is derived from the public key.
4.2.1.2	CA subject key identifiers	The keyIdentifier field is derived from the public key which is processed by a SHA-1 hash.
4.2.1.10	Name constraints	Capable of processing rfc822Name, uniformResourceIdentifier, dNSName, and iPAddress name forms.
4.2.1.11	Policy constraints	Processing of the inhibitPolicyMapping field is supported.
5.2.4	Delta CRL update	Use of the latest value of thisUpdate when multiple delta CRLs are received.

Table 8: X.509 Implementation Details

7.6.5.1 TLS Key-Based Authentication

When applying the bi-directional certificate validation for TLS, the TOE can be configured to verify the certificate's distinguished name (DN).

The verification of the certificate's DN is performed after the certificate validation part of the bi-directional certificate validation which ensures that the client certificate is trustworthy.

7.7 Trusted Path / Channel

The TOE offers different cryptographic services to protect user data. The following subsections cover the different types of cryptographic services analyzed as part of the evaluation. Additional cryptographic mechanisms are active in the TOE which, however, are not subject to the assessments of this evaluation.

The TOE provides cryptographically secured network communication channels to allow remote users to interact with the TOE. Using one of the following cryptographically secured network channels, a user can request the following services:

- OpenSSH: The OpenSSH application provides access to the command line interface of the TOE. Users may employ OpenSSH for interactive sessions as well as for non-interactive sessions. The console provided via OpenSSH provides the same environment as a local console. OpenSSH implements the SSHv2 protocol.
- NSS: The NSS library offers generic TLS communication services to protect user data using TLS v1.1 or TLS v1.2 for protecting the communication link.

7.7.1 TLS Protocol

The TOE provides TLSv1.1 and TLSv1.2 to allow users from a remote host to establish a secure channel to the TOE. The TOE can be configured using a bi-directional certificate verification where the client side validates the server certificate.

The following RFCs are supported for implementing the TLS protocol: RFC 4346 (TLS 1.1) and RFC5246 (TLS 1.2).

The TOE supports the generation of the RSA key pair used by the client. The key generation mechanism uses the OpenSSL random number generator. The evaluated configuration also allows the use of an externally-generated certificate. A widely accepted Certification Authority might be used to generate and/or sign such a certificate (allowing a wide community trusting this CA to validate the certificate). In a closed community it might also be sufficient to have one server within the community to act as a CA. The OpenSSL library provides the functions to set up such a CA, but those functions are not subject of this Security Target.

7.8 Secure Shell

The TOE provides the Secure Shell Protocol Version 2 (SSH v2.0) to allow users from a remote host to establish a secure connection and perform a logon to the TOE.

The TOE supports the generation of RSA, ECDSA key pairs. These key pairs are used by OpenSSH for the host keys as well as for the per-user keys. When a user registers his public key with the user he wants to access on the server side, a key-based authentication can be performed instead of a

password-based authentication. The key generation mechanism uses the Linux kernel random number generator. The evaluated configuration permits the import of externally-generated key pairs.

The TOE supports the following security functions of the SSH v2.0 protocol:

- Establishing a secure communication channel using the following cryptographic functions provided by the SSH v2.0 protocol:
  - Encryption as defined in section 4.3 of RFC4253 - the keys are generated using the random number generator of the underlying cryptographic library;
  - Diffie-Hellman key exchange as defined in section 6.1 of RFC4253
  - The keyed hash function for integrity protection as defined in section 4.4 of RFC4253

  Note: The protocol supports more cryptographic algorithms than the ones listed above. Those other algorithms are not covered by this evaluation and should be disabled or not used when running the evaluated configuration.
- Performing user authentication using the standard password-based authentication method the TOE provides for users (password authentication method as defined in chapter 5 of RFC4252).
- Performing user authentication using a RSA or ECDSA key-based authentication method (public key authentication method as defined in chapter 5 of RFC4252).
- Checking the integrity of the messages exchanged and close down the connection in case an integrity error is detected.

The OpenSSH applications of sshd, ssh and ssh-keygen use the OpenSSL random number generator seeded by /dev/random or /dev/urandom to generate cryptographic keys. OpenSSL provides different DRNGs depending whether the FIPS 140-2 mode is enabled in the system.

7.8.1 OpenSSH Implementation Details

Secure Shell (SSH) is a network protocol that provides a replacement for insecure remote login and command execution facilities such as telnet, rlogin, and Remote Shell (RSH). SSH encrypts traffic, preventing traffic sniffing and password theft.

On a local system, the user starts the SSH client to open a connection to a remote server running the sshd daemon. If the user is authenticated successfully, an interactive session is initiated, allowing the user to run commands on the remote system. SSH is not a shell in the sense of a command interpreter, but it permits the use of a shell on the remote system.

In addition to interactive logins, the user can tunnel TCP network connections through the existing channel, allowing the use of X11 and other network-based applications, and copy files through the use of the scp and sftp tools. OpenSSH is configured to use the PAM framework for authentication, authorization, account maintenance, and session maintenance. Password expiration and locking are handled through the appropriate PAM functions.

Communication between the SSH client and SSH server uses the SSH protocol, version 2.0. The SSH protocol requires that each host have a host-specific key. When the SSH client initiates a connection, the keys are exchanged using the Diffie-Hellman protocol. A session key is generated, and all traffic is encrypted using this session key and the agreed-upon algorithm.

Default encryption algorithms supported by SSH are 3DES (triple DES) and blowfish. The default can be overridden by providing the list in the server configuration file with the “Ciphers” keyword. The evaluated configuration will define ciphers compliant to this Security Target.

The default message authentication code algorithms supported by SSH are SHA-1 and MD5. The default can be overridden by providing the list in the server configuration file with the keyword MACs. The evaluated configuration will define ciphers compliant to this Security Target.

Encryption is provided by the OpenSSL package, which is a separate software package. The following briefly describes the default SSH setup with respect to encryption, integrity check, certificate format, and key exchange protocol.

- Encryption: A number of ciphers and block chaining modes are available with OpenSSH. A subset is allowed in the evaluated configuration.
- Integrity check: Data integrity is protected by including a message authentication code (MAC) with each packet that is computed from a shared secret, packet sequence number, and the contents of the packet. The message authentication algorithm and key are negotiated during key exchange. Initially, no MAC will be in effect, and its length must be zero. After key exchange, the selected MAC will be computed before encryption from the concatenation of packet data mac = MAC (key, sequence_number || unencrypted_packet) where unencrypted_packet is the entire packet without MAC (the length fields, payload and padding), and sequence_number is an implicit packet sequence number represented as uint32. The sequence number is initialized to zero for the first packet, and is incremented after every packet, regardless of whether encryption or MAC is in use. It is never reset, even if keys or algorithms are renegotiated later. It wraps around to zero after every 2^{32} packets. The packet sequence number itself is not included in the packet sent over the wire.

  The MAC algorithms for each direction must run independently, and implementations must allow choosing the algorithm independently for both directions. The MAC bytes resulting from the MAC algorithm must be transmitted without encryption as the last part of the packet. The number of MAC bytes depends on the algorithm chosen. The default MAC algorithm defined is the hmac-sha1 (with digest length = key length = 20 bytes).
- Certificate format: RSA (2048 bits and higher) and ECDSA (NIST P-256, NIST P-384) are available and used as specified in FIPS 186-4. These certificates can be used for host authentication as well as the key-based authentication of users.
- Key exchange protocol: The following key agreement protocols are available:
  - diffie-hellman-group14-sha1: Diffie-Hellman key agreement with SHA-1 and domain parameters of size 2048 bits defined in RFC3526
  - diffie-hellman-group-exchange-sha1: Diffie-Hellman key agreement with SHA-1 and domain parameters generated as defined in RFC4419 - OpenSSH provides a set of pre-computed Diffie-Hellman domain parameters in /etc/ssh/moduli. During the SSH protocol handshake, the client and server negotiate the domain parameter set where both must agree on a set that is located in /etc/ssh/moduli on both sides.
  - diffie-hellman-group-exchange-sha256: This option is identical to diffie-hellman-group-exchange-sha1 except that it requires SHA-256 to be used.
  - ecdh-sha2-nistp256: Elliptic Curve Diffie-Hellman key agreement with SHA-256 using the NIST curve P-256 as defined in RFC5656

- ecdh-sha2-nistp384: Elliptic Curve Diffie-Hellman key agreement with SHA-384 using the NIST curve P-384 as defined in RFC5656
- ecdh-sha2-nistp521: Elliptic Curve Diffie-Hellman key agreement with SHA-384 using the NIST curve P-521 as defined in RFC5656

The following subsections briefly describe the implementation of SSH client and SSH server. For detailed information about the SSH Transport Layer Protocol, SSH Authentication Protocol, SSH Connection Protocol, and SSH Protocol Architecture, refer to the corresponding protocol specifications in RFC 4250ff.

7.8.1.1 SSH Client

The SSH client first parses arguments and reads the configuration (readconf.c), then calls ssh_connect (in sshconnect*.c) to open a connection to the server, and performs authentication (ssh_login in sshconnect.c). Terminal echo is turned off while users type their passwords, which prevents the password from being displayed on the terminal as it is being typed. The SSH client then makes requests such as allocating a pseudo-tty, forwarding X11 connections, forwarding TCP-IP connections and so on, and might call code in ttymodes.c to encode current tty modes. Finally, the SSH client calls client_loop in clientloop.c.

The client is typically installed with suid as root. The client temporarily gives up this right while reading the configuration data. The root privileges are used to make the connection from a privileged socket, which is required for host-based authentication and to read the host key for host-based authentication using protocol version 1. Any extra privileges are dropped before calling ssh_login. Because .rhosts support is not included in the TSF, the SSH client is not suid root on the system.

Any SSH packet larger than 2^{18} bytes is discarded.

7.8.1.2 SSH Server Daemon

The sshd daemon starts by processing arguments and reading the /etc/ssh/sshd_config configuration file. The configuration file contains keyword-argument pairs, one per line. Refer to the sshd_config man page for available configuration options. The daemon then reads the host key, starts listening for connections, and generates the server key.

When the server receives a connection, it forks a process and re-executes the sshd binary, disables the regeneration alarm, and starts communicating with the client. The server and client first perform identification string exchange, and then negotiate encryption and perform authentication. If authentication is successful, the forked process sets the effective user ID to that of the authenticated user, performs preparatory operations, and enters the normal session mode by calling server_loop in serverloop.c.

When the server accepts a new connection, it prints the contents of the file pointed to by the configuration variable “Banner” before any authentication takes place.

Any SSH packet larger than 2^{18} bytes is discarded.

7.8.1.2.1 Password-based authentication

The password based authentication utilizes the PAM library if the configuration option UsePAM is set in sshd_config. The SSH daemon receives the user name and password after setting up the SSH

tunnel and feeds it into the PAM library. The following sequence is used by the SSH daemon to access the PAM library:

1. Initializing the interaction with the PAM library using the pam_start. The PAM configuration name is set to the file name of the SSH daemon which is “sshd”.
2. Establishing a thread that is used for the authentication conversation. That thread uses pam_authenticate to authenticate the user. If the PAM library requires a change of the authentication token, pam_chauthtok is called.
3. If the authentication returns PAM_SUCCESS, pam_open_session is used to set up the user session.

7.8.1.2.2 Key-based authentication

If the key-based authentication is enabled, the SSH daemon allows the use of RSA or ECDSA keys as authentication token.

The following steps are performed by the SSH daemon:

1. Verify that the user name is defined on the local system. If not, the authentication attempt is terminated.
2. The key-based authentication is performed as defined by RFC 4252. The public key for the key-based authentication must reside in the home directory of the target user in the file .ssh/authorized_keys. As this file may contain multiple key, each key is tried whether it is appropriate as a public key for the authentication attempt (i.e. whether the public key can decrypt the data sent by the client encrypted with the client's private key). The first key that is found to match the private key indicates a successful authentication.
3. If the authentication was successful, pam_open_session is used to set up the user session. The session part of the PAM configuration file for the SSH daemon is applied.

7.9 SFR to TSS References

The Protection Profile mandates various specific information to be supplied in the TSS to cover aspects of the SFRs. The following table enumerates the SFRs from the Protection Profile and the extended packages and adds pointers into the TSS documenting the respective aspects.

SFR	Coverage in TSS
FCS_CKM.1(1)	The key generation supported by the TOE is documented in section 7.1. References to CAVS certificates are provided in the application notes to the SFR.
FCS_CKM.2(1)	The key establishment supported by the TOE is documented in section 7.1. References to CAVS certificates are provided in the application notes to the SFR.
FCS_CKM.4	The key destruction supported by the TOE is documented in section 7.1.
FCS_COP.1(1)	The cryptographic mechanisms provided by the TOE are documented in section 7.1.

SFR	Coverage in TSS
	References to CAVS certificates are provided in the application notes to the SFR.
FCS_COP.1(2)	The cryptographic mechanisms provided by the TOE are documented in section 7.1. References to CAVS certificates are provided in the application notes to the SFR.
FCS_COP.1(3)	The cryptographic mechanisms provided by the TOE are documented in section 7.1. References to CAVS certificates are provided in the application notes to the SFR.
FCS_COP.1(4)	The cryptographic mechanisms provided by the TOE are documented in section 7.1. References to CAVS certificates are provided in the application notes to the SFR.
FCS_RBG_EXT.1	The cryptographic mechanisms provided by the TOE are documented in section 7.1. References to CAVS certificates are provided in the application notes to the SFR.
FCS_STO_EXT.1	The block device encryption support is documented in section 7.1.5.
FCS_TLSC_EXT.1	The supported TLS cipher suites and TLS options are listed in section 7.1.2.
FCS_TLSC_EXT.2	The supported curves are listed in section 7.1.2.
FDP_ACF_EXT.1	The DAC support is explained in section 7.2.
FDP_IFC_EXT.1	The TOE provides different VPN interface to allow VPN clients to implement a VPN: the TOE provides the XFRM framework with the XFRM netlink interface. In addition, it provides the TUN/TAP interface for supporting user-space VPN clients operating at ISO/OSI level 2 or 3.
FMT_MOF_EXT.1	Security management is documented in section 7.4.
FMT_SMF_EXT.1	Security management is documented in section 7.4.
FPT_ACF_EXT.1	The TSF protection is documented in section 7.3.
FPT_ASLR_EXT.1	The TOE implements address space layout randomization for all user space code.
FPT_SBOP_EXT.1	The use of stack canaries is documented in section 7.3.1.
FPT_TST_EXT.1	Secure boot is documented in section 7.3.3.

SFR	Coverage in TSS
FPT_TUD_EXT.1	The TOE allows automated check for the availability of OS updates as documented in section 7.3.4.
FPT_TUD_EXT.2	The TOE allows automated check for the availability of application updates for applications delivered as part of the Oracle Linux distribution as documented in section 7.3.4.
FAU_GEN.1	The audit support and auditable events are documented in section 7.5.
FIA_AFL.1	The identification and authentication support is documented in section 7.6.
FIA_UAU.5	The identification and authentication support is documented in section 7.6.
FIA_X509_EXT.1	X.509 support is documented in section 7.6.5.
FIA_X509_EXT.2	X.509 support is documented in section 7.6.5.
FTP_ITC_EXT.1	TLS support is documented in section 7.7.1. SSH support is documented in section 7.8.
FTP_TRP.1	Remote OS administration is offered via SSH. The administrator logs into the TOE via SSH to perform system-local administration operations.
FPT_TST_EXT.1	The executed self tests are documented in section 7.1.6.
FCS_SSH_EXT.1	SSH support is documented in section 7.8.
FCS_SSHC_EXT.1	SSH support including the supported cryptographic mechanisms is documented in section 7.8.
FCS_SSHS_EXT.1	SSH support including the supported cryptographic mechanisms is documented in section 7.8.

Table 9: SFR to TSS References

8 Abbreviations

Abbreviation	Description
AH	Authentication Header
CC	Common Criteria
DAC	Discretionary Access Control
EAL	Evaluation Assurance Level
ESP	Encapsulating Security Payload
IKE	Internet Key Exchange
IPSEC	IP Security Protocol
MAC	Mandatory Access Control
OSPP	Operating System Protection Profile
PP	Protection Profile
SAR	Security Assurance Requirement
SFP	Security Function Policy
SFR	Security Functional Requirement
SSH	Secure Shell
ST	Security Target
TOE	Target of Evaluation
TLS	Transport Layer Security
TSF	TOE security function
TSFI	TSF Interface
TSP	TOE security policy