# *Red Hat Virtualization Security Target*

| | |
|---|---|
| ***Version:*** | ***2.3*** |
| ***Status:*** | ***RELEASED*** |
| ***Last Update:*** | ***2021-12-08*** |
| ***Classification:*** | ***Public*** |

# Trademarks

# Legal Notices

# Table of Contents

# List of Tables



  

# List of Figures



  

# 1 Introduction

## 1.1 Security Target Identification

| | |
|---|---|
| Title: | Red Hat Virtualization Security Target |
| Version: | 2.3 |
| Status: | RELEASED |
| Date: | 2021-12-08 |
| Sponsor: | Red Hat |
| Developer: | Red Hat, Inc. |
| Keywords: | Security Target, Common Criteria |

## 1.2 TOE Identification

The TOE is Red Hat Virtualization Version 4.3.

## 1.3 TOE Type

The TOE type is Linux-based virtualization environment.

## 1.4 TOE Overview

This security target documents the security characteristics of the Red Hat Virtualization distribution (abbreviated as RHV throughout this document).

### 1.4.1 Description

Red Hat Virtualization is an enterprise-grade virtualization platform built on Red Hat Enterprise Linux. Virtualization allows users to provision new virtual servers and workstations and provides more efficient use of physical server resources.

Red Hat Enterprise Linux (RHEL) 7.9 provides virtualization primitives used by Red Hat Virtualization. The Red Hat Virtualization Host is derived from RHEL 7.9. The following components are part of the TOE:

| ***Name*** | ***Description*** |
|---|---|
| Red Hat Virtualization Host (RHVH) | A minimal installation of the Red Hat Enterprise Linux (RHEL) environment establishes the Red Hat Virtualization Host. It contains the Linux kernel providing the KVM virtualization environment. In addition, the user space QEMU framework is provided which utilizes the KVM environment to instantiate virtual machines. The libvirtd management system controls the life-cycle of virtual machines. |
| Red Hat Virtualization Manager | A service that provides a graphical user interface and a REST API to manage the resources in the environment using virtualization primitives provided by the virtualization host as described above. The Manager is installed on a physical or virtual machine running Red Hat Enterprise Linux. |

***Table 1: TOE Components***

## 1.4.2 Security functionality

### 1.4.2.1 Audit

The TOE implements its audit functionality using the Linux Audit Framework (LAF) provided by RHEL. 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.

Events are logged in an audit trail, ASCII files stored locally. The TOE provides tools to manage, view, and perform post-processing on the audit trail. Audit files are accessible by root only. The TOE notifies the administrator when the audit trail reaches a certain threshold and can halt the system to guarantee no audit data is lost.

### 1.4.2.2 User Data Protection

The TOE uses hardware-based isolation mechanisms and physical platform resource controls to protect user data. Many are implemented using components of the virtual machine environment.

Virtual machine functionality is implemented by the Kernel-based Virtual Machine (KVM) Linux kernel module, the Quick Emulator (QEMU) virtual machine monitor used for hardware emulation, and libvirtd which serves as a management daemon to control resources assigned to virtual machines. Both are provided by the underlying RHEL system. To the host system, a virtual machine is just another running process. KVM implements memory management and virtual machine maintenance functionality. QEMU provides I/O virtualization.

The TOE uses Linux kernel mechanisms to constrain VM access to physical devices. Some of these mechanisms depend on the following underlying hardware-based mechanisms.

- Processor virtualization support
- Shadow page table support
- IOMMU virtualization support

The TOE implements access control restrictions to limit virtual machine access to only their resources. These restrictions are implemented using both Linux and hardware-based capabilities.

- Security-enhanced Linux (SELinux), which is part of RHEL
- the hardware I/O memory management unit (IOMMU)
- Linux Control groups (cgroups)

Separation of VMs is also achieved since each virtual machine runs in a separate Linux process.

The TOE supports the following types of interfaces or Virtual Devices:

- hypercalls used to access para-virtualized host services
- exceptions used to signal the host kernel
- VNC providing access to the VM console

Access to the guest VM console is possible through QEMU using either the SPICE or VNC graphics protocols.

The TOE provides residual information protection for data in memory and data on disk using mechanisms provided by the Linux kernel.

### 1.4.2.3 Identification and Authentication

The TOE provides multiple authentication mechanisms, authentication failure handling, and password management, leveraged from RHEL.

#### 1.4.2.4 Security Management

The TOE supports two types of users, a regular user and an administrative user. The administrator creates virtual machines and manages their resources. The regular user starts, stops, and interacts with deployed virtual machines.

The TOE provides ways for the administrator of the virtualization environment to manage:

- physical resources
- system resources (memory, disk)
- objects and properties
- virtual machines

The TOE also provides the capability for the administrator to perform resource administration on:

- hosts
- storage
- the network

Separation of management and operational networks is achieved by establishing a separate administrative LAN for the TOE.

The TOE is administered using the Red Hat Virtualization Manager which provides an easy-to-use graphical user interface. The management framework is implemented by libvirtd in RHEL. All management operations performed in the GUI are handed to the libvirtd management system which enforces configurations. The GUI is SFR-supporting only, not SFR-enforcing. Management-related security claims are implemented by libvirtd unless explicitly noted.

#### 1.4.2.5 Protection of the TSF

The TOE protects itself from removable (both connected and disconnected) and non-existent virtual devices and isolates itself from guest VMs. Numerous hardware assists and execution environment mitigation mechanisms are also employed.

The TOE provides several mechanisms to protect against exploitation of common buffer overrun attacks.

- a guard variable (stack canary) to validate stack data
- address space layout randomization (ASLR)
- making runtime memory segments read-only (RELRO)
- using FORTIFY_SOURCE macro to identify buffer overflows

The TOE will also take advantage of Intel chips' SMEP or SMAP features if available.

#### 1.4.2.6 Trusted Path/Channels

The TOE identifies which resources are being accessed via a trusted channel.

### 1.4.3 IT environment support

The TOE may be run on several systems on a network. Each TOE system implements its own security policy. If other systems are connected to the network, they must 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 connections between this network and untrusted networks (e. g. the Internet) must 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.

The TOE requires at least one Intel x64 platform upon which it will be installed.

TOE functionality depends on no software or firmware external to the TOE.

## 1.5 TOE Description

The TOE is a Virtualization System. A simplified view of a generic Virtualization System and platform, provided in [BVPPv1.0], is reproduced in Figure 1. In this example, TOE components are shaded red and non-TOE components are shaded blue. The platform is the hardware, firmware, and software onto which the VS is installed.

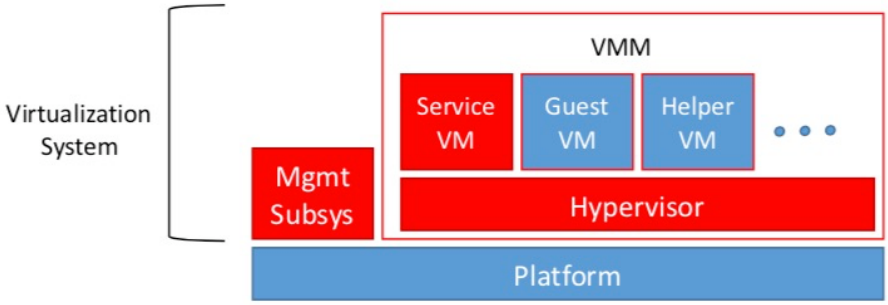


***Figure 1: Virtualization System and Platform***

### 1.5.1 Architecture

Red Hat Virtualization can be deployed as a self-hosted engine or as a standalone Manager. In the evaluated configuration, the TOE is deployed as a self-hosted engine.

In complete Red Hat Virtualization deployment is created by installing the TOE in cooperation with other non-TOE components:

- storage service
- data warehouse
- metrics store

#### 1.5.1.1 Self-Hosted Engine Architecture

In the evaluated configuration, the Red Hat Virtualization Manager runs as a virtual machine on self-hosted engine nodes (specialized hosts) in the same environment it manages. A self-hosted engine environment requires one less physical server, but more administrative overhead to deploy and manage. The Manager is highly available without external HA management.

The minimum setup of a self-hosted engine environment includes:

- One Red Hat Virtualization Manager virtual machine hosted on one of the self-hosted engine nodes. The RHV-M Appliance is used to automate the installation of a Red Hat Enterprise Linux 7 virtual machine and the Manager on that virtual machine.
- A minimum of two self-hosted engine nodes for virtual machine high availability. This can be achieved using Red Hat Enterprise Linux hosts or Red Hat Virtualization Hosts (RHVH). VDSM (the host agent) runs on all hosts to facilitate communication with the Red Hat Virtualization Manager. The HA services run on all self-hosted engine nodes to manage the high availability of the Manager virtual machine.
- One storage service which can be hosted locally or on a remote server depending on the storage type used. The storage service must be accessible to all hosts.

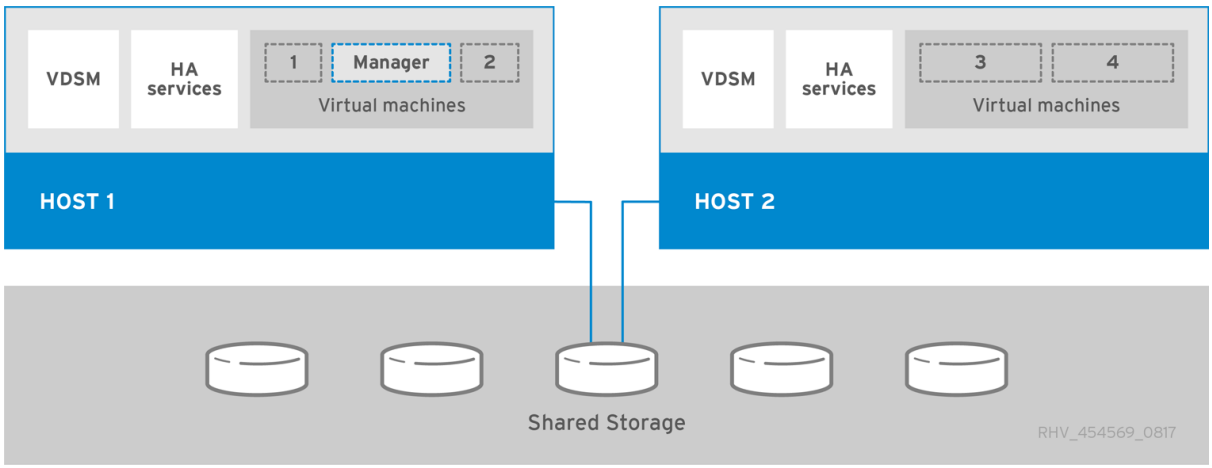


***Figure 2: Self-Hosted Engine Red Hat Virtualization Architecture***

### *1.5.1.2 Red Hat Virtualization Manager*

The following components are part of the TOE, but do not provide any security functionality. The ST covers the separation of virtual machines from each other, as well as separation between the virtual machines and the host. The installation and preparation of the resources required for virtual machines are part of the TOE, but outside of the security claims.

#### *Storage*

Setting up storage and attaching it to the Red Hat Virtualization environment is a prerequisite before creating end-user virtual machines. Red Hat Virtualization can use three types of storage domains, however only the data domain is now fully supported.

- The data domain contains all the data associated with virtual machines. The data domain supports all storage types that are supported for use with Red Hat Virtualization.
- The ISO domain is a deprecated storage domain type that was used to store ISO files for installing a virtual machine operating system or additional applications, such as the Windows guest agents and drivers. Virtual machine images can now be uploaded to data domains instead.
- The export domain is a deprecated storage domain type that was used as a temporary storage repository for moving images between data centers and Red Hat Virtualization environments. This is now accomplished by importing data storage domains.

The ISO and export domains only support file-based storage types (NFS, POSIX, or GlusterFS). The ISO domain supports local storage when used in a local storage data center.

#### *Data Warehouse*

The Red Hat Virtualization Manager includes a data warehouse that collects monitoring data about hosts, virtual machines, and storage. Data Warehouse, which includes a database and a service, must be installed and configured along with the Manager, either on the same machine or on a separate server.

The Red Hat Virtualization installation creates two databases.

- The Manager database (engine) is the primary data store used by the Red Hat Virtualization Manager. Information about the virtualization environment, like its state, configuration, and performance, are stored in this database.
- The Data Warehouse database (ovirt_engine_history) contains configuration information and statistical data which is collated over time from the Manager

database. The configuration data in the Manager database is examined every minute and changes are replicated to the Data Warehouse database. Tracking the changes to the database provides information on the objects in the database. This enables analysis and enhancement of the performance of the Red Hat Virtualization environment.

#### *Metrics Store*

The Metrics Store architecture is based on the OpenShift EFK logging stack and runs on the OpenShift Container Platform. This includes Elasticsearch, a Metric Store virtual machine, used to index data. Another Metric Store virtual machine called Kibana provides dashboards, charts, and data analysis.

Metrics Store collects logs and metrics from Red Hat Virtualization. The data is transferred from Red Hat Virtualization to OpenShift where it is stored and aggregated in Elasticsearch and saved in indexes. The data can then be analyzed and visualized in Kibana.

- Elasticsearch is a distributed, RESTful search and analytics engine providing many types of searches.
- Kibana is an open source analytics and visualization platform designed to work with Elasticsearch. One can easily perform advanced data analysis and visualize data in a variety of charts and tables.

### *1.5.1.3 Red Hat Virtualization*

Operations such as storage, host management, user connections, and virtual machine connectivity all rely on a well-planned and well-configured network to deliver optimal performance. Setting up networking is a vital prerequisite for a Red Hat Virtualization environment. Planning for projected networking requirements and implementing the network accordingly is much simpler than discovering networking requirements through use and altering the network configuration retroactively.

Red Hat Virtualization separates network traffic by defining logical networks. Logical networks define the path that a selected network traffic type must take through the network. They are created to isolate network traffic by functionality or to virtualize a physical topology.

The ovirtmgmt logical network is created by default and labeled as the Management network. The ovirtmgmt logical network is intended for management traffic between the Red Hat Virtualization Manager and hosts. Additional logical networks may be defined to segregate:

- General virtual machine traffic
- Storage-related traffic (such as NFS or iSCSI)
- Virtual machine migration traffic
- Virtual machine display traffic
- Gluster storage traffic

### *1.5.1.4 Segregation of TOE Components from non-TOE Components*

The TOE includes all components outlined in Table 1. The TOE includes the Red Hat Virtualization Manager which is provided via the oVirt management framework. oVirt allows the configuration of complex resources like Gluster, NFS or iSCSI storage. Any resource that is not local to the TOE instance is considered to be a non-TOE component. For example, the NFS server that may be accessed by the TOE and the guest operating systems managed by the TOE are not part of the TOE.

In addition, the TOE allows the configuration of external authentication providers like LDAP or Active Directory. All authentication providers external to the TOE are non-TOE

components. Only the authentication of users using the local user database is part of the TOE and subject to security claims in this Security Target.

## 1.5.2 TOE boundaries

### 1.5.2.1 Physical

The Target of Evaluation is Red Hat Virtualization 4.3. The TOE is supplied as ISO images containing installation executables distributed via the Red Hat Network. The TOE includes a package containing the user and administrator documentation for the TOE.

The general TOE documentation is also available online at the Red Hat Network.

Along with the installation media files, the following documentation is provided as part of the TOE:

- Evaluated Configuration Guide [ECG]
- Product Guide [RHVPG]
- Technical Reference [RHVTR]
- Administration Guide [RHVAG]
- Planning and prerequisites Guide [RHVPPG]

The Evaluated Configuration Guide as well as the associated guides can be obtained from https://access.redhat.com/articles/2918071.

### 1.5.2.2 Logical

The TOE includes all components outlined in Table 1. The TOE includes the Red Hat Virtualization Manager, which is provided via the oVirt management framework. oVirt allows the configuration of complex resources like Gluster, NFS or iSCSI storage. Any resource that is not local to the TOE instance is considered to be a non-TOE component. For example, the NFS server that may be accessed by the TOE and the guest operating systems managed by the TOE are not part of the TOE.

The TOE provides the following security features:

***Auditing***

The TOE implements auditing using the Linux Audit Framework (LAF) provided by RHEL. LAF gathers audit events from system calls and audit log entries of user and system applications. The TOE can also be deployed as an audit server and receive audit logs from other TOE instances.

***Virtual machine environments***

The TOE implements the host system for virtual machines, providing separation between resources. It acts as a hypervisor providing an environment in which other operating systems may execute concurrently.

***Security Management***

The security management facilities provided by the TOE are usable by authorized administrators to modify the configuration of TSF. The TOE allows the configuration of external authentication providers like LDAP or Active Directory. All authentication providers external to the TOE are non-TOE components, but only the authentication of users using the local user database is part of the TOE and subject to security claims in this ST.

***Runtime Protection Mechanisms***

The TOE leverages mechanisms supported by the underlying Linux system to prevent or significantly increase the complexity of an exploitation of common buffer overflow and similar attacks. These mechanisms are used for the TSF and are available to untrusted code. It also takes advantage of hardware-provided support on Intel CPUs to prevent the kernel from dereferencing user space memory (except in well-defined cases) and prevent the execution of code residing in user space memory.

#### *1.5.2.3 Evaluated configuration*

The evaluated configuration is defined as follows:

- The CC evaluated package set must be selected at install time and installed and configured in accordance with the descriptions provided in the Evaluated Configuration Guide ([ECG]).
- The TOE is configured self-hosted as shown by Host 1 in Figure 2.
- The TOE is configured with a dedicated administrative network (either a separate physical LAN or an isolated VLAN) for separation of operational and administrative functions.

Any deviation from the configurations and settings specified in [ECG] take the TOE out of its Evaluated Configuration.

### *1.5.3 Security Policy Model*

The security policy for RHV is defined by the security functional requirements in section 6.1 and refined into a security policy model by the TSF in section 7.1 . The following is a list of the subjects and objects participating in the policy.

***Subjects:***

- unprivileged users
- administrative users

***Objects:***

- data objects
- physical devices (CPU, RAM, PCI devices, Block devices, TPM, Watchdog)
- removable devices and media (USB, CD/DVD, ISO image)
- virtual machines

***TSF data:***

- user accounts, including the following security attributes:
  - user ID
  - default access group
  - password
  - superuser attribute
  - association with access groups and permission groups
- workflow definitions (e.g., assignment of required permissions and parameters)
- SELinux labels (for identification of each virtual machine)
- audit records
- VM configuration data

***User data:***

- user data is mainly constituted by attributes of the data objects that are not relevant for the TSP enforcements, such as parameters for remote management of the instance represented by a data object

# 2 CC Conformance Claim

This Security Target is CC Part 2 extended and CC Part 3 conformant, with a claimed Evaluation Assurance Level of EAL2, augmented by ALC_FLR.3.

This Security Target does not claim conformance to any Protection Profile.

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

# 3 Security Problem Definition

## 3.1 Threat Environment

This section describes the threat model for the TOE and identifies the individual threats that are assumed to exist in the TOE environment.

The ***assets*** to be protected by the TOE are

- virtual machine images
- storage objects used to store user and TSF data
- TSF functions

The resources and metadata, such as management attributes, used by the TSF to store and manage these assets must be protected from unauthorized read access, modification, deletion or creation of new illegitimate data by the TOE

The ***threat agents*** having an interest in manipulating the data model can be categorized as either:

- Unauthorized individuals or malware (“attackers”) which are unknown to the TOE and its runtime environment.
- Authorized users or administrators of the TOE who try to manipulate data that they are not authorized to access.

Threat agents originate from a well-managed user community within an organizations internal network. Hence, only inadvertent or casual attempts to breach system security are expected from this community.

TOE administrators, including administrators of the TOE environment, are assumed to be trustworthy, trained and to follow the instructions provided to them with respect to the secure configuration and operation of the systems under their responsibility. Hence, only inadvertent attempts to manipulate the safe operation of the TOE are expected from this community.

### 3.1.1 Threats countered by the TOE

***T.UNAUTHORIZED_MODIFICATION***

Malware running in a VM is able to modify the underlying VS or another VM or VS components outside of its own VM.

***T.PLATFORM_COMPROMISE***

An attacker accesses the underlying platform in a manner not controlled by the VMM to modify system firmware or software compromising both the Virtualization System and the underlying platform.

***T.UNAUTHORIZED_ACCESS***

An adversary with access to an open management network could undetected perform management functions or the TOE by injecting commands into the management infrastructure.

## 3.2 Assumptions

***A.PLATFORM_INTEGRITY***

It is assumed that the TOE platform works as specified, has no undocumented security critical side effects and has not been compromised prior to installation of the TOE.

***A.PHYSICAL***

It is assumed that the TOE is operated in an environment preventing unauthorized physical access.

***A.TRUSTED_ADMIN***

It is assumed that the administrators are trained, trusted and follow their guidance.

***A.NON_MALICIOUS_USER***

It is assumed that the users of the VS are not willfully negligent or hostile, and follow their guidance.

## *3.3 Organizational Security Policies*

***P.DENIAL_OF_SERVICE***

No VM shall via a resource exhaustion of shared resources be able to block other VMs from using system resources provided by the VS (e.g., system memory, persistent storage, and processing time).

***P.DATA_LEAKAGE***

The domains encapsulated by different VMs must remain separate and prohibits data transfer between VMs unless explicitly allowed by the security policy.

***P.CONFIGURATION***

The policies of the VS for information flow control between the VMs and the access control to resources provide by the VS platform must be configurable.

# 4 Security Objectives

## 4.1 Objectives for the TOE

***O.VM_ISOLATION***

The TOE must support the mechanisms to isolate all resources associated with virtual networks and to limit a VM's access to only those virtual networks for which it has been configured. The TOE must also support the mechanisms to control the configurations of virtual networks according to the SSP.

***O.VMM_INTEGRITY***

The integrity of each VMM component in the VS must be established and maintained.

***O.PLATFORM_INTEGRITY***

The VS should provide no capability for a user or any hosted software to undermine the integrity of the platform.

***O.MANAGEMENT_ACCESS***

Only administrators are allowed to exercise VMM management functions. VMM management functions include VM configuration, virtualized network configuration, allocation of physical resources, and reporting. Only certain authorized system users (administrators) are allowed to exercise management functions.

***O.AUDIT***

The TOE must provide accountability of any management functions performed by the administrators.

***O.CORRECTLY_APPLIED_CONFIGURATION***

The TOE must correctly apply changes to configurations and policies to guest VMs as specified by the administrator and within the existing security policy.

***O.RESOURCE_ALLOCATION***

The TOE will provide mechanisms that enforce constraints on the allocation of system resources in accordance with existing security policy.

## 4.2 Objectives for the Operational Environment

***OE.PHYSICAL***

The TOE is operated in an environment preventing unauthorized physical access.

***OE.TRUSTED_ADMIN***

The administrators are trained, trusted and follow their guidance.

***OE.NON_MALICIOUS_USER***

The users of the VS are not willfully negligent or hostile, and follow their guidance.

***OE.PLATFORM_INTEGRITY***

The TOE platform works as specified, has no undocumented security critical side effects and has not been compromised prior to installation of the TOE.

## 4.3 Security Objectives Rationale

### 4.3.1 Coverage

The following table provides a mapping of TOE objectives to threats and policies, showing that each objective counters or enforces at least one threat or policy, respectively.

| ***Objective*** | ***Threats / OSPs*** |
|---|---|
| O.VM_ISOLATION | P.DATA_LEAKAGE |

| *Objective* | *Threats / OSPs* |
|---|---|
| O.VMM_INTEGRITY | T.UNAUTHORIZED_MODIFICATION |
| O.PLATFORM_INTEGRITY | T.PLATFORM_COMPROMISE |
| O.MANAGEMENT_ACCESS | T.UNAUTHORIZED_ACCESS<br>P.CONFIGURATION |
| O.AUDIT | T.UNAUTHORIZED_MODIFICATION |
| O.CORRECTLY_APPLIED_CONFIGURATION | P.CONFIGURATION |
| O.RESOURCE_ALLOCATION | P.DENIAL_OF_SERVICE |

***Table 2: Mapping of security objectives to threats and policies***

The following table provides a mapping of the objectives for the Operational Environment to assumptions, threats and policies, showing that each objective holds, counters or enforces at least one assumption, threat or policy, respectively.

| *Objective* | *Assumptions / Threats / OSPs* |
|---|---|
| OE.PHYSICAL | A.PHYSICAL |
| OE.TRUSTED_ADMIN | A.TRUSTED_ADMIN |
| OE.NON_MALICIOUS_USER | A.NON_MALICIOUS_USER |
| OE.PLATFORM_INTEGRITY | A.PLATFORM_INTEGRITY |

***Table 3: Mapping of security objectives for the environment to the SPD***

### *4.3.2 Sufficiency*

The following rationale provides justification that the security objectives are suitable to counter each individual threat and that each security objective tracing back to a threat, when achieved, actually contributes to the removal, diminishing or mitigation of that threat.

| *Threat* | *Rationale for security objectives* |
|---|---|
| T.UNAUTHORIZED_MODIFICATION | O.VMM_INTEGRITY provides for enforcement of VMM integrity preventing the bypass of enforcement mechanisms. O.AUDIT ensures that abuse of legitimate authority can be detected. |
| T.PLATFORM_COMPROMISE | O.PLATFORM_INTEGRITY ensures that users and hosted software will not have any capabilities to break out of a VM and affect the platform on which the VS is running. This will prevent them from undermining the integrity of the platform. |
| T.UNAUTHORIZED_ACCESS | O.MANAGEMENT_ACCESS ensures TSF management functions cannot be executed without authorization prevents untrusted subjects from modifying the behavior of the TOE in an unanticipated manner. |

***Table 4: Sufficiency of objectives countering threats***

The following rationale provides justification that the security objectives for the environment are suitable to cover each individual assumption, that each security objective for the environment that traces back to an assumption about the environment of use of the TOE, when achieved, actually contributes to the environment achieving consistency with the assumption, and that if all security objectives for the environment that trace back to an assumption are achieved, the intended usage is supported.

| ***Assumption*** | ***Rationale for security objectives*** |
|---|---|
| A.PLATFORM_INTEGRITY | The security objective OE.PLATFORM_INTEGRITY is literally the same as the assumption and therefore directly upholds the assumption. |
| A.PHYSICAL | The security objective OE.PHYSICAL is literally the same as the assumption and therefore directly upholds the assumption. |
| A.TRUSTED_ADMIN | The security objective OE.TRUSTED_ADMIN is literally the same as the assumption and therefore directly upholds the assumption. |
| A.NON_MALICIOUS_USER | The security objective OE.NON_MALICIOUS_USER is literally the same as the assumption and therefore directly upholds the assumption. |

***Table 5: Sufficiency of objectives holding assumptions***

The following rationale provides justification that the security objectives are suitable to cover each individual organizational security policy (OSP), that each security objective that traces back to an OSP, when achieved, actually contributes to the implementation of the OSP, and that if all security objectives that trace back to an OSP are achieved, the OSP is implemented.

| ***OSP*** | ***Rationale for security objectives*** |
|---|---|
| P.DENIAL_OF_SERVICE | The policy to provide mechanisms to enforce constraints on the allocation of system resources in accordance with existing security policy is implemented by the objective O.RESOURCE_ALLOCATION. |
| P.DATA_LEAKAGE | O.VM_ISOLATION provides for separation of VMs and enforcement of domain integrity prevent unauthorized transmission of data from one VM to another. |
| P.CONFIGURATION | O.CORRECTLY_APPLIED_CONFIGURATION and O.MANAGEMENT_ACCESS provide mechanisms to prevent the application of configurations that violate the security policy help prevent misconfigurations. |

***Table 6: Sufficiency of objectives enforcing Organizational Security Policies***

# 5 Extended Components Definition

This ST defines the following extended components based on the virtualization SFRs defined in [BVPPv1.0] (all except FMT_MOF_EXT.1) and [SVEPv1.0] (FMT_MOF_EXT.1 only). Most of these SFRs are not from CC Part 2, but were created specifically to address virtualization functionality.

## 5.1 Class FDP: User data protection

### 5.1.1 Hardware-Based Isolation Mechanisms (FDP_HBI_EXT)

Family behaviour

Enumerate the hardware-based isolation mechanisms used by the TOE and the physical devices to which they constrain a Guest VM's access.

Management: FDP_HBI_EXT.1

There are no management activities foreseen.

Audit: FDP_HBI_EXT.1

There are no audit events foreseen.

#### 5.1.1.1 FDP_HBI_EXT.1 - Extended component for Hardware-Based Isolation Mechanisms

Hierarchical to: No other components.

Dependencies: No dependencies.

***FDP_HBI_EXT.1.1*** ***The TSF shall use [selection: no mechanism, [assignment: list of platform-provided, hardware-based mechanisms]] to constrain a Guest VM's direct access to the following physical devices: [selection: no devices, [assignment: physical devices to which the VMM allows Guest VMs physical access]].***

### 5.1.2 Physical Platform Resource Controls (FDP_PPR_EXT)

Family behaviour

Enumerate the physical platform resources to which the administrator may control Guest VM access.

Management: FDP_PPR_EXT.1

There are no management activities foreseen.

Audit: FDP_PPR_EXT.1

There are no audit events foreseen.

#### 5.1.2.1 FDP_PPR_EXT.1 - Extended component for Physical Platform Resource Controls

Hierarchical to: No other components.

Dependencies: No dependencies.

***FDP_PPR_EXT.1.1*** ***The TSF shall allow an authorized administrator to control Guest VM access to the following physical platform resources:***

***[assignment: list of physical platform resources the VMM is able to control access to].***

***FDP_PPR_EXT.1.2*** ***The TSF shall explicitly deny all Guest VMs access to the following physical platform resources: [selection: no physical platform resources, [assignment: list of physical platform resources to which access is explicitly denied]].***

***FDP_PPR_EXT.1.3*** ***The TSF shall explicitly allow all Guest VMs access to the following physical platform resources: [selection: no physical platform resources, [assignment: list of physical platform resources to which access is always allowed]].***

## *5.1.3 Residual information protection (FDP_RIP)*

Management: FDP_RIP_EXT.1

There are no management activities foreseen.

Management: FDP_RIP_EXT.2

There are no management activities foreseen.

Audit: FDP_RIP_EXT.1

There are no audit events foreseen.

Audit: FDP_RIP_EXT.2

There are no audit events foreseen.

### *5.1.3.1 FDP_RIP_EXT.1 - Extended component for Residual Information in Memory*

Hierarchical to: No other components.

Dependencies: No dependencies.

***FDP_RIP_EXT.1.1*** ***The TSF shall ensure that any previous information content of physical memory is cleared prior to allocation to a Guest VM.***

### *5.1.3.2 FDP_RIP_EXT.2 - Extended component for Residual Information on Disk*

Hierarchical to: No other components.

Dependencies: No dependencies.

***FDP_RIP_EXT.2.1*** ***The TSF should ensure that any previous information content of physical disk storage is cleared prior to allocation to a Guest VM or the TSS shall describe the conditions of exception.***

## *5.1.4 VM Separation (FDP_VMS_EXT)*

Family behaviour

Enumerate the mechanisms provided by the TOE for data transfer between Guest VMs and specify the administrator's ability to configure them.

Management: FDP_VMS_EXT.1

There are no management activities foreseen.

Audit: FDP_VMS_EXT.1

There are no audit events foreseen.

#### *5.1.4.1 FDP_VMS_EXT.1 - Extended component for VM Separation*

Hierarchical to: No other components.

Dependencies: No dependencies.

***FDP_VMS_EXT.1.1*** ***The VS shall provide the following mechanisms for transferring data between Guest VMs: [selection: no mechanism, virtual networking, [assignment: other inter-VM data sharing mechanisms]].***

***FDP_VMS_EXT.1.2*** ***The TSF shall allow Administrators to configure these mechanisms to [selection: enable, disable] the transfer of data between Guest VMs.***

***FDP_VMS_EXT.1.3*** ***The VS shall ensure that no Guest VM is able to read or transfer data to or from another Guest VM except through the mechanisms listed in FDP_VMS_EXT.1.1.***

### *5.1.5 Virtual Networking Components (FDP_VNC_EXT)*

Family behaviour

Require the administrator be able to configure virtual networking components to connect VMs to each other or to physical networks and ensure and that the traffic is only visible to the Guest VMs intended.

Management: FDP_VNC_EXT.1

There are no management activities foreseen.

Audit: FDP_VNC_EXT.1

There are no audit events foreseen.

#### *5.1.5.1 FDP_VNC_EXT.1 - Extended component for Virtual Networking Components*

Hierarchical to: No other components.

Dependencies: No dependencies.

***FDP_VNC_EXT.1.1*** ***The TSF shall allow Administrators to configure virtual networking components to connect VMs to each other, and to physical networks.***

***FDP_VNC_EXT.1.2*** ***The TSF shall ensure that network traffic visible to a Guest VM on a virtual network - or virtual segment of a physical network - is visible only to Guest VMs configured to be on that virtual***

***network or segment.***

## 5.2 Class FIA: Identification and authentication

### 5.2.1 Authentication failures (FIA_AFL)

Management: FIA_AFL_EXT.1

The following actions could be considered for the management functions in FMT:

a) management of the threshold for unsuccessful authentication attempts;

b) management of actions to be taken in the event of an authentication failure.

Audit: FIA_AFL_EXT.1

The following actions should be auditable if FAU_GEN Security audit data generation is included in the PP/ST:

c) Minimal: the reaching of the threshold for the unsuccessful authentication attempts and the actions (e.g. disabling of a terminal) taken and the subsequent, if appropriate, restoration to the normal state (e.g. re-enabling of a terminal).

#### 5.2.1.1 FIA_AFL_EXT.1 - Extended component for Authentication Failure Handling

Hierarchical to: No other components.

Dependencies: No dependencies.

***FIA_AFL_EXT.1.1*** ***The TSF shall detect when [selection:***

- ***[assignment: a positive integer number]***
- ***an administrator configurable positive integer within a [assignment: range of acceptable values]***

***] unsuccessful authentication attempts occur related to administrators attempting to authenticate remotely using a [selection: password, PIN].***

***FIA_AFL_EXT.1.2*** ***When the defined number of unsuccessful authentication attempts for an account has been met, the TSF shall: [selection: Account Lockout, Account Disablement, Mandatory Credential Reset, [assignment: list of actions]].***

### 5.2.2 Password Management (FIA_PMG_EXT)

Family behaviour

Require specific password management capabilities for administrative passwords.

Management: FIA_PMG_EXT.1

The following actions could be considered for the management functions in FMT:

d) management function for password management.

Audit: FIA_PMG_EXT.1

There are no audit events foreseen.

#### *5.2.2.1 FIA_PMG_EXT.1 - Extended component for Password Management*

Hierarchical to: No other components.

Dependencies: No dependencies.

***FIA_PMG_EXT.1.1*** ***The TSF shall provide the following password management capabilities for administrative passwords:***

- ***a) Passwords shall be able to be composed of any combination of upper and lower case characters, digits, and the following special characters: [selection: !, @, #, $, %, ^, &, *, (, ), [assignment: other characters]]***
- ***b) Minimum password length shall be configurable;***
- ***c) Passwords of at least 15 characters in length shall be supported.***

### *5.2.3 Administrator Identification and Authentication (FIA_UIA_EXT)*

Family behaviour

Require the administrator be successfully identified and authenticated before allowing any TSF-medicated management function to be performed.

Management: FIA_UIA_EXT.1

There are no management activities foreseen.

Audit: FIA_UIA_EXT.1

There are no audit events foreseen.

#### *5.2.3.1 FIA_UIA_EXT.1 - Extended component for Administrator Identification and Authentication*

Hierarchical to: No other components.

Dependencies: No dependencies.

***FIA_UIA_EXT.1.1*** ***The TSF shall require Administrators to be successfully identified and authenticated using one of the methods in FIA_UAU.5 before allowing any TSF-mediated management function to be performed by that Administrator.***

## *5.3 Class FMT: Security management*

### *5.3.1 Management of functions in TSF (FMT_MOF)*

Management: FMT_MOF_EXT.1

There are no management activities foreseen.

Audit: FMT_MOF_EXT.1

There are no audit events foreseen.

#### *5.3.1.1 FMT_MOF_EXT.1 - Extended component for Management of Security Functions Behavior*

Hierarchical to: No other components.

Dependencies: FMT_MSA_EXT.1 Extended component for Default Data Sharing Configuration

***FMT_MOF_EXT.1.1*** ***The TSF shall be capable of supporting [selection: local, remote] administration.***

### *5.3.2 Management of security attributes (FMT_MSA)*

Management: FMT_MSA_EXT.1

There are no management activities foreseen.

Audit: FMT_MSA_EXT.1

There are no audit events foreseen.

#### *5.3.2.1 FMT_MSA_EXT.1 - Extended component for Default Data Sharing Configuration*

Hierarchical to: No other components.

Dependencies: No dependencies.

***FMT_MSA_EXT.1.1*** ***The TSF shall by default enforce a policy prohibiting sharing of data between Guest VMs using [selection: no mechanism, virtual networking, [assignment: other inter-VM data sharing mechanisms]].***

***FMT_MSA_EXT.1.2*** ***The TSF shall allow Administrators to specify alternative initial configuration values to override the default values when a Guest VM is created.***

### *5.3.3 Separation of Management and Operational Networks (FMT_SMO_EXT)*

Family behaviour

Enumerate the means by which separate management and operational networks are configured.

Management: FMT_SMO_EXT.1

There are no management activities foreseen.

Audit: FMT_SMO_EXT.1

There are no audit events foreseen.

#### *5.3.3.1 FMT_SMO_EXT.1 - Extended component for Separation of Management and Operational Networks*

Hierarchical to: No other components.

Dependencies: No dependencies.

***FMT_SMO_EXT.1.1*** ***The TSF shall support the configuration of separate management and operational networks through [selection: physical means, logical means, trusted channel].***

## *5.4 Class FPT: Protection of the TSF*

### *5.4.1 Non-Existence of Disconnected Virtual Devices (FPT_DVD_EXT)*

Family behaviour

Require the TSF limit a Guest VM's access to virtual devices to those present in the VM's virtual hardware configuration.

Management: FPT_DVD_EXT.1

There are no management activities foreseen.

Audit: FPT_DVD_EXT.1

There are no audit events foreseen.

#### *5.4.1.1 FPT_DVD_EXT.1 - Extended component for Non-Existence of Disconnected Virtual Devices*

Hierarchical to: No other components.

Dependencies: No dependencies.

***FPT_DVD_EXT.1.1*** ***The TSF shall limit a Guest VM’s access to virtual devices to those that are present in the VM’s current virtual hardware configuration.***

### *5.4.2 Execution Environment Mitigations (FPT_EEM_EXT)*

Family behaviour

Enumerate the vulnerability mitigation mechanisms used by the TOE.

Management: FPT_EEM_EXT.1

There are no management activities foreseen.

Audit: FPT_EEM_EXT.1

There are no audit events foreseen.

#### *5.4.2.1 FPT_EEM_EXT.1 - Extended component for Execution Environment Mitigations*

Hierarchical to: No other components.

Dependencies: No dependencies.

***FPT_EEM_EXT.1.1*** ***The TSF shall take advantage of execution environment-based vulnerability mitigation mechanisms supported by the Platform such as: [selection:***

1. ***Address space randomization***
2. ***Memory execution protection (e.g., DEP)***
3. ***Stack buffer overflow protection***

4. ***Heap corruption detection***
5. ***[assignment: other mechanisms]***
6. ***No mechanisms***

***].***

## *5.4.3 Hardware Assists (FPT_HAS_EXT)*

Family behaviour

Enumerate the hardware assists used by the TOE to reduce or eliminate the need for binary translation and shadow page tables.

Management: FPT_HAS_EXT.1

There are no management activities foreseen.

Audit: FPT_HAS_EXT.1

There are no audit events foreseen.

### *5.4.3.1 FPT_HAS_EXT.1 - Extended component for Hardware Assists*

Hierarchical to: No other components.

Dependencies: No dependencies.

***FPT_HAS_EXT.1.1*** ***The VMM shall use [assignment: list of hardware-based virtualization assists] to reduce or eliminate the need for binary translation.***

***FPT_HAS_EXT.1.2*** ***The VMM shall use [assignment: list of hardware-based virtualization memory-handling assists] to reduce or eliminate the need for shadow page tables.***

## *5.4.4 Removable Devices and Media (FPT_RDM_EXT)*

Family behaviour

Require the TSF implement controls for the transfer of virtual and physical removable media or devices and specify the rules enforced upon specified devices.

Management: FPT_RDM_EXT.1

There are no management activities foreseen.

Audit: FPT_RDM_EXT.1

There are no audit events foreseen.

### *5.4.4.1 FPT_RDM_EXT.1 - Extended component for Removable Devices and Media*

Hierarchical to: No other components.

Dependencies: No dependencies.

***FPT_RDM_EXT.1.1*** ***The TSF shall implement controls for handling the transfer of virtual and physical removable media and virtual and physical removable media devices between information domains.***

***FPT_RDM_EXT.1.2*** ***The TSF shall enforce the following rules when [assignment: virtual or physical removable media and virtual or physical removable media devices ] are switched between information domains, then [selection:***

1. ***the Administrator has granted explicit access for the media or device to be connected to the receiving domain,***
2. ***the media in a device that is being transferred is ejected prior to the receiving domain being allowed access to the device,***
3. ***the user of the receiving domain expressly authorizes the connection,***
4. ***the device or media that is being transferred is prevented from being accessed by the receiving domain***

***].***

## *5.4.5 Virtual Device Parameters (FPT_VDP_EXT)*

Family behaviour

Require the TSF to provide interfaces for virtual devices as part of the virtual hardware abstraction and that the TSF validate parameters passed to the interface before executing the functionality provided.

Management: FPT_VDP_EXT.1

There are no management activities foreseen.

Audit: FPT_VDP_EXT.1

There are no audit events foreseen.

### *5.4.5.1 FPT_VDP_EXT.1 - Extended component for Virtual Device Parameters*

Hierarchical to: No other components.

Dependencies: No dependencies.

***FPT_VDP_EXT.1.1*** ***The TSF shall provide interfaces for virtual devices implemented by the VMM as part of the virtual hardware abstraction.***

***FPT_VDP_EXT.1.2*** ***The TSF shall validate the parameters passed to the virtual device interface prior to execution of the VMM functionality exposed by those interfaces.***

## *5.4.6 VMM Isolation from VMs (FPT_VIV_EXT)*

Family behaviour

Require that the TSF ensure the software running in a VM is not able to degrade or disrupt other VMs, the VMM, or the platform and that a Guest VM is not able to invoke platform code running at a privileged level at or above that of the VMM without involvement of the VMM.

Management: FPT_VIV_EXT.1

There are no management activities foreseen.

Audit: FPT_VIV_EXT.1

There are no audit events foreseen.

#### *5.4.6.1 FPT_VIV_EXT.1 - Extended component for VMM Isolation from VMs*

Hierarchical to: No other components.

Dependencies: No dependencies.

***FPT_VIV_EXT.1.1*** ***The TSF must ensure that software running in a VM is not able to degrade or disrupt the functioning of other VMs, the VMM, or the Platform.***

***FPT_VIV_EXT.1.2*** ***The TSF must ensure that a Guest VM is unable to invoke platform code that runs at a privilege level equal to or exceeding that of the VMM without involvement of the VMM.***

## *5.5 Class FTP: Trusted path/channels*

### *5.5.1 User Interface: I/O Focus (FTP_UIF_EXT)*

Family behaviour

Require the TSF to indicate to the user which VM has current input focus.

Management: FTP_UIF_EXT.1

There are no management activities foreseen.

Management: FTP_UIF_EXT.2

There are no management activities foreseen.

Audit: FTP_UIF_EXT.1

There are no audit events foreseen.

Audit: FTP_UIF_EXT.2

There are no audit events foreseen.

#### *5.5.1.1 FTP_UIF_EXT.1 - Extended component for User Interface: I/O Focus*

Hierarchical to: No other components.

Dependencies: No dependencies.

***FTP_UIF_EXT.1.1*** ***The TSF shall indicate to users which VM, if any, has the current input focus.***

#### *5.5.1.2 FTP_UIF_EXT.2 - Extended component for User Interface: Identification of VM*

Hierarchical to: No other components.

Dependencies: No dependencies.

***FTP_UIF_EXT.2.1*** ***The TSF shall support the unique identification of a VM's output display to users.***

# 6 Security Requirements

The Security Functional Requirements (SFRs) included in this section are derived from [CC] Part 2 with additional extended functional components.

The CC defines operations on Security Functional Requirements. This document uses the following font conventions to identify the operations performed:

- Assignments and selections are indicated with ***bold*** text;
- Refinement deletions are indicated by ~~strikethrough~~, refinement additions are indicated by *italicized* text;
- No iterations of SFRs are performed.

Extended SFRs are identified by the text "_EXT" appended to the name.

## 6.1 TOE Security Functional Requirements

The following table shows the SFRs for the TOE, and the operations performed on the components according to CC part 1: iteration (Iter.), refinement (Ref.), assignment (Ass.) and selection (Sel.).

| ***Security functional class*** | ***Security functional requirement*** | ***Source*** | ***Operations*** | | | |
|---|---|---|---|---|---|---|
| | | | ***Iter.*** | ***Ref.*** | ***Ass.*** | ***Sel.*** |
| FAU - Security audit | FAU_GEN.1 Audit data generation | CC Part 2 | No | No | Yes | Yes |
| | FAU_SAR.1 Audit review | CC Part 2 | No | No | Yes | No |
| | FAU_STG.1 Protected audit trail storage | CC Part 2 | No | No | No | Yes |
| FDP - User data protection | FDP_HBI_EXT.1 Extended component for Hardware-Based Isolation Mechanisms | ECD | No | No | Yes | Yes |
| | FDP_PPR_EXT.1 Extended component for Physical Platform Resource Controls | ECD | No | No | Yes | Yes |
| | FDP_RIP_EXT.1 Extended component for Residual Information in Memory | ECD | No | No | No | No |
| | FDP_RIP_EXT.2 Extended component for Residual Information on Disk | ECD | No | No | No | No |
| | FDP_VMS_EXT.1 Extended component for VM Separation | ECD | No | No | No | Yes |
| | FDP_VNC_EXT.1 Extended component for Virtual Networking Components | ECD | No | No | No | No |
| FIA - Identification and authentication | FIA_AFL_EXT.1 Extended component for Authentication Failure Handling | ECD | No | Yes | Yes | Yes |
| | FIA_PMG_EXT.1 Extended component for Password Management | ECD | No | Yes | No | Yes |
| | FIA_UAU.5 Multiple | CC Part 2 | No | Yes | Yes | Yes |

| ***Security functional class*** | ***Security functional requirement*** | ***Source*** | ***Operations*** | | | |
|---|---|---|---|---|---|---|
| | | | ***Iter.*** | ***Ref.*** | ***Ass.*** | ***Sel.*** |
| | authentication mechanisms | | | | | |
| | FIA_UIA_EXT.1 Extended component for Administrator Identification and Authentication | ECD | No | No | No | No |
| FMT - Security management | FMT_MOF_EXT.1 Extended component for Management of Security Functions Behavior | ECD | No | No | No | Yes |
| | FMT_MSA_EXT.1 Extended component for Default Data Sharing Configuration | ECD | No | No | No | Yes |
| | FMT_SMO_EXT.1 Extended component for Separation of Management and Operational Networks | ECD | No | No | No | Yes |
| FPT - Protection of the TSF | FPT_DVD_EXT.1 Extended component for Non-Existence of Disconnected Virtual Devices | ECD | No | No | No | No |
| | FPT_EEM_EXT.1 Extended component for Execution Environment Mitigations | ECD | No | No | Yes | Yes |
| | FPT_HAS_EXT.1 Extended component for Hardware Assists | ECD | No | No | Yes | No |
| | FPT_RDM_EXT.1 Extended component for Removable Devices and Media | ECD | No | No | Yes | Yes |
| | FPT_STM.1 Reliable time stamps | CC Part 2 | No | No | No | No |
| | FPT_VDP_EXT.1 Extended component for Virtual Device Parameters | ECD | No | No | No | No |
| | FPT_VIV_EXT.1 Extended component for VMM Isolation from VMs | ECD | No | No | No | No |
| FTP - Trusted path/channels | FTP_UIF_EXT.1 Extended component for User Interface: I/O Focus | ECD | No | No | No | No |
| | FTP_UIF_EXT.2 Extended component for User Interface: Identification of VM | ECD | No | No | No | No |

***Table 7: SFRs for the TOE***

## *6.1.1 Security audit (FAU)*

### *6.1.1.1 Audit data generation (FAU_GEN.1)*

***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) additional information defined in Table 8: Auditable Events.***

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

d) Date and time of the event, type of event, subject identity (if applicable), and the outcome (success or failure) of the event; and

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

***Application Note:*** The table entry for FDP_VNC_EXT.1 refers to configuration settings that attach VMs to virtualized network components. Changes to these configurations can be made during VM execution or when VMs are not running. Audit records must be generated for either case.

The intent of the audit requirement for FDP_PPR_EXT.1 is to log that the VM is connected to a physical device (when the device becomes part of the VM's hardware view), not to log every time that the device is accessed. Generally, this is only once at VM startup. However, some devices can be connected and disconnected during operation (e.g., virtual USB devices such as CD-ROMs). All such connection/disconnection events must be logged.

| ***Requirement*** | ***Auditable Events*** | ***Additional Audit Record Contents*** |
|---|---|---|
| FAU_GEN.1 | None. | None. |
| FAU_SAR.1 | Basic: Reading of information from the audit records. | None. |
| FAU_STG.1 | Failure of audit data capture due to lack of disk space or pre-defined limit. On failure of logging function, capture record of failure and record upon restart of logging function. | None. |
| FDP_HBI_EXT.1 | None. | None. |
| FDP_PPR_EXT.1 | Successful and failed VM connections to physical devices where connection is governed by configurable policy. Security policy violations. The intent of this audit requirement is to log that the VM is connected to a physical device (when the device becomes part of the VM's hardware view), not to log every time that the device is accessed. Generally, this is only once at VM startup. However, some devices can be connected and disconnected during operation (e.g., | VM and physical device identifiers. Identifier for the security policy that was violated. |

| ***Requirement*** | ***Auditable Events*** | ***Additional Audit Record Contents*** |
|---|---|---|
| | virtual USB devices such as CD-ROMs). All such connection/disconnection events must be logged. | |
| FDP_RIP_EXT.1 | None. | None. |
| FDP_RIP_EXT.2 | None. | None. |
| FDP_VMS_EXT.1 | None. | None. |
| FDP_VNC_EXT.1 | Successful and failed attempts to connect VMs to virtual and physical networking components. Security policy violations. Administrator configuration of inter-VM communications channels between VMs. Changes to these configurations can be made during VM execution or when VMs are not running. Audit records must be generated for either case. | VM and virtual or physical networking component identifiers. Identifier for the security policy that was violated. |
| FIA_AFL_EXT.1 | Minimal: the reaching of the threshold for the unsuccessful authentication attempts and the actions (e.g., disabling of a terminal) taken and the subsequent, if appropriate, restoration to the normal state (e.g., re-enabling of a terminal). | None. |
| FIA_PMG_EXT.1 | None. | None. |
| FIA_UAU.5 | Minimal: The final decision on authentication. | None. |
| FIA_UIA_EXT.1 | Administrator authentication attempts. All use of the identification and authentication mechanism. | Provided user identity, origin of the attempt (e.g., console, remote IP address). |
| FMT_MOF_EXT.1 | None. | None. |
| FMT_MSA_EXT.1 | None. | None. |
| FMT_SMO_EXT.1 | None. | None. |
| FPT_DVD_EXT.1 | None. | None. |
| FPT_EEM_EXT.1 | None. | None. |
| FPT_HAS_EXT.1 | None. | None. |
| FPT_RDM_EXT.1 | Connection/disconnection of removable media or device to/from a VM. Ejection/insertion of removable media or device from/to an already connected VM. | VM Identifier, removable media/device identifier, event description or identifier (connect/disconnect, ejection/insertion, etc.). |
| FPT_STM.1 | Minimal: changes to the time. | None. |
| FPT_VDP_EXT.1 | None. | None. |
| FPT_VIV_EXT.1 | None. | None. |
| FTP_UIF_EXT.1 | None. | None. |
| FTP_UIF_EXT.2 | None. | None. |

***Table 8: Auditable Events***

#### *6.1.1.2 Audit review (FAU_SAR.1)*

***FAU_SAR.1.1*** The TSF shall provide ***the root user*** with the capability to read ***all audit information*** from the audit records.

***FAU_SAR.1.2*** The TSF shall provide the audit records in a manner suitable for the user to interpret the information.

***Application Note:*** The audit records are stored in ASCII format and can therefore be read with a normal editor or pager. In addition, the TOE provides specific tools that support the interpretation of the audit trail.

***Application Note:*** The audit trail is stored in a file that is readable to the users with the above-mentioned capabilities only.

#### *6.1.1.3 Protected audit trail storage (FAU_STG.1)*

***FAU_STG.1.1*** The TSF shall protect the stored audit records in the audit trail from unauthorized deletion.

***FAU_STG.1.2*** The TSF shall be able to ***prevent*** unauthorized modifications to the stored audit records in the audit trail.

### *6.1.2 User data protection (FDP)*

#### *6.1.2.1 Extended component for Hardware-Based Isolation Mechanisms (FDP_HBI_EXT.1)*

***FDP_HBI_EXT.1.1*** The TSF shall use

1. ***Intel VT-x***
2. ***Intel EPT***
3. ***Intel IOMMU***

to constrain a Guest VM’s direct access to the following physical devices:

1. ***CPU***
2. ***RAM***
3. ***PCI devices***

***Application Note:*** The TSF must use available hardware-based isolation mechanisms to constrain VMs when VMs have direct access to physical devices. “Direct access” in this context means that the VM can read or write device memory or access device I/O ports without the VMM being able to intercept and validate every transaction.

#### *6.1.2.2 Extended component for Physical Platform Resource Controls (FDP_PPR_EXT.1)*

***FDP_PPR_EXT.1.1*** The TSF shall allow an authorized administrator to control Guest VM access to the following physical platform resources:

a) ***CPU***
b) ***RAM***

***c) PCI devices***
***d) USB devices***
***e) CD-ROM***
***f) Block devices***
***g) TPM***
***h) Watchdog***

***FDP_PPR_EXT.1.2*** The TSF shall explicitly deny all Guest VMs access to the following physical platform resources: ***no physical platform resources***.

***FDP_PPR_EXT.1.3*** The TSF shall explicitly allow all Guest VMs access to the following physical platform resources: ***no physical platform resources***.

***Application Note:*** This requirement specifies that the VMM controls access to physical platform resources, and indicates that it must be configurable, but does not specify the means by which that is done.

### *6.1.2.3 Extended component for Residual Information in Memory (FDP_RIP_EXT.1)*

***FDP_RIP_EXT.1.1*** The TSF shall ensure that any previous information content of physical memory is cleared prior to allocation to a Guest VM.

***Application Note:*** Physical memory must be zeroed before it is made accessible to a VM for general use by a Guest OS.

The purpose of this requirement is to ensure that a VM does not receive memory containing data previously used by another VM or the host.

For general use means for use by the Guest OS in its page tables for running applications or system software.

This does not apply to pages shared by design or policy between VMs or between the VMMs and VMs, such as read-only OS pages or pages used for virtual device buffers.

### *6.1.2.4 Extended component for Residual Information on Disk (FDP_RIP_EXT.2)*

***FDP_RIP_EXT.2.1*** The TSF should ensure that any previous information content of physical disk storage is cleared prior to allocation to a Guest VM or the TSS shall describe the conditions of exception.

***Application Note:*** Disk storage should be zeroed before it is made accessible to a VM for use by a Guest OS or the TSS must document the conditions under which physical disk storage is not cleared prior to allocation to a Guest VM.

The purpose of this requirement is to ensure that a VM does not receive disk storage containing data previously used by another VM or the host.

This does not apply to disk-resident files shared by design or policy between VMs or between the VMMs and VMs, such as read-only data files or files used for inter-VM data transfers permitted by policy.

### *6.1.2.5 Extended component for VM Separation (FDP_VMS_EXT.1)*

***FDP_VMS_EXT.1.1*** The VS shall provide the following mechanisms for transferring data between Guest VMs: ***virtual networking*** .

***FDP_VMS_EXT.1.2*** The TSF shall allow Administrators to configure these mechanisms to ***enable, disable*** the transfer of data between Guest VMs.

***FDP_VMS_EXT.1.3*** The VS shall ensure that no Guest VM is able to read or transfer data to or from another Guest VM except through the mechanisms listed in FDP_VMS_EXT.1.1.

***Vendor attestation:*** A Guest VM cannot access the data of another Guest VM, or transfer data to another Guest VM other than through the mechanisms described in FDP_VMS_EXT.1.1 when expressly enabled by an authorized Administrator. There are no design or implementation flaws that permit the above mechanisms to be bypassed or defeated, or for data to be transferred through undocumented mechanisms. This claim does not apply to covert channels or architectural side-channels.

***Application Note:*** The fundamental requirement of a Virtualization System is the ability to enforce separation between information domains implemented as Virtual Machines and Virtual Networks. The intent of this requirement is to ensure that VMs, VMMs, and the Virtualization System as a whole is implemented with this fundamental requirement in mind.

For data transfer mechanisms that use virtual networking, FDP_VMS_EXT.1.2 is met if FDP_VNC_EXT.1.1 is met (VM access to virtual networks is configurable).

FDP_VMS_EXT.1.3 is an attestation requirement. The vendor must attest that data cannot be transferred between Guest VMs except through the configurable mechanisms documented in FDP_VMS_EXT.1.1. The vendor must attest that there are no design or implementation flaws that permit the above mechanisms to be bypassed or defeated, or for data to be transferred through a different, undocumented mechanism.

#### *6.1.2.6 Extended component for Virtual Networking Components (FDP_VNC_EXT.1)*

***FDP_VNC_EXT.1.1*** The TSF shall allow Administrators to configure virtual networking components to connect VMs to each other, and to physical networks.

***FDP_VNC_EXT.1.2*** The TSF shall ensure that network traffic visible to a Guest VM on a virtual network—or virtual segment of a physical network—is visible only to Guest VMs configured to be on that virtual network or segment.

***Vendor attestation:*** Traffic traversing a virtual network is visible only to Guest VMs that are configured by an Administrator to be members of that virtual network. There are no design or implementation flaws that permit the virtual networking configuration to be bypassed or defeated, or for data to be transferred through undocumented mechanisms. This claim does not apply to covert channels or architectural side-channels.

***Application Note:*** Virtual networks must be isolated from one another to provide assurance commensurate with that provided by physically separate networks. It must not be possible for data to cross between properly configured virtual networks regardless of whether the traffic originated from a local Guest VM or a remote host.

Unprivileged users must not be able to connect VMs to each other or to external networks.

FDP_VNC_EXT.1.2 is an attestation requirement. The vendor must attest that traffic traversing a virtual network is visible only to Guest VMs that are configured by an Administrator to be members of that virtual network, and that there are no design or implementation flaws that permit the virtual networking configuration to be bypassed or defeated, or for data to be transferred through undocumented mechanisms.

## 6.1.3 Identification and authentication (FIA)

### 6.1.3.1 Extended component for Authentication Failure Handling (FIA_AFL_EXT.1)

***FIA_AFL_EXT.1.1*** The TSF shall detect when ***an administrator configurable positive integer within a range of 1 to 10*** unsuccessful authentication attempts occur related to administrators attempting to authenticate remotely using a ***password*** *when authenticating with the local user database* .

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

### 6.1.3.2 Extended component for Password Management (FIA_PMG_EXT.1)

***FIA_PMG_EXT.1.1*** The TSF shall provide the following password management capabilities for administrative passwords *when authenticating with the local user database* :

a) Passwords shall be able to be composed of any combination of upper and lower case characters, digits, and the following special characters: ***!, @, #, $, %, ^, &, *, (, )***

b) Minimum password length shall be configurable;

c) Passwords of at least 15 characters in length shall be supported.

### 6.1.3.3 Multiple authentication mechanisms (FIA_UAU.5)

***FIA_UAU.5.1*** The TSF shall provide *the following authentication mechanisms:*

- ***authentication based on username and password*** *using local user database*
- ***authentication based on username and password using remote authentication providers***
- ***authentication based on SSH keys***

to support

user authentication *(SSH keys) and Administrator authentication (passwords)*

***FIA_UAU.5.2*** The TSF shall authenticate any user's *or Administrator's* claimed identity according to the ***following rules:***

- ***password-based authentication via the local password database provided by the TOE (for Administrator)***
- ***password-based authentication via an authentication backend (such as LDAP or single sign-on mechanism) configured by the administrator and provided by the Operational Environment (for Administrator)***
- ***SSH key-based authentication (for user)***

#### 6.1.3.4 Extended component for Administrator Identification and Authentication (FIA_UIA_EXT.1)

***FIA_UIA_EXT.1.1*** The TSF shall require Administrators to be successfully identified and authenticated using one of the methods in FIA_UAU.5 before allowing any TSF-mediated management function to be performed by that Administrator.

### 6.1.4 Security management (FMT)

#### 6.1.4.1 Extended component for Management of Security Functions Behavior (FMT_MOF_EXT.1)

***FMT_MOF_EXT.1.1*** The TSF shall be capable of supporting ***remote*** administration.

#### 6.1.4.2 Extended component for Default Data Sharing Configuration (FMT_MSA_EXT.1)

***FMT_MSA_EXT.1.1*** The TSF shall by default enforce a policy prohibiting sharing of data between Guest VMs using ***virtual networking***.

***FMT_MSA_EXT.1.2*** The TSF shall allow Administrators to specify alternative initial configuration values to override the default values when a Guest VM is created.

***Application Note:*** By default, the VMM must enforce a policy prohibiting sharing of data between VMs. The default policy applies to all mechanisms for sharing data between VMs, including inter-VM communication channels, shared physical devices, shared virtual devices, and virtual networks. The default policy does not apply to covert channels and architectural side-channels.

#### 6.1.4.3 Extended component for Separation of Management and Operational Networks (FMT_SMO_EXT.1)

***FMT_SMO_EXT.1.1*** The TSF shall support the configuration of separate management and operational networks through ***physical means, logical means***.

***Application Note:*** Management communications must be separate from user workloads. Administrative communications including communications between physical hosts concerning load balancing, audit data, VM startup and shutdown must be separate from guest operational networks.

Physical means refers to using separate physical networks for management and operational networks. For example, the machines in the management network are connected by separate cables plugged into separate and dedicated physical ports on each physical host.

Logical means refers to using separate network cables to connect physical hosts together using general-purpose networking ports. The management and operational networks are kept separate within the hosts using separate virtualized networking components.

## 6.1.5 Protection of the TSF (FPT)

### 6.1.5.1 Extended component for Non-Existence of Disconnected Virtual Devices (FPT_DVD_EXT.1)

***FPT_DVD_EXT.1.1*** The TSF shall limit a Guest VM's access to virtual devices to those that are present in the VM's current virtual hardware configuration.

***Application Note:*** The virtualized hardware abstraction implemented by a particular VS might include the virtualized interfaces for many different devices. Sometimes these devices are not present in a particular instantiation of a VM. The interface for devices not present must not accessible by the VM.

Such interfaces include memory buffers and processor I/O ports.

The purpose of this requirement is to reduce the attack surface of the VMM by closing unused interfaces.

### 6.1.5.2 Extended component for Execution Environment Mitigations (FPT_EEM_EXT.1)

***FPT_EEM_EXT.1.1*** The TSF shall take advantage of execution environment-based vulnerability mitigation mechanisms supported by the Platform such as:

1. ***Address space randomization***
2. ***Stack buffer overflow protection (Stack Canary): Modification of a function return address on the process' or thread's stack to jump to previously known processor instructions by misusing the following C programming language constructs (also known as Stack Protector Strong):***
   i. ***Functions with stack buffers larger than 8 bytes;***
   ii. ***Functions using alloca();***
   iii. ***Functions with local array definitions;***
   iv. ***Functions having references to local frame addresses;***
3. ***Read-Only Relocation (RELRO)***
4. ***FORTIFY_SOURCE set to 2***
5. ***Intel SMEP***
6. ***Intel SMAP***

***Application Note:*** Processor manufacturers, compiler developers, and operating system vendors have developed execution environment-based mitigations that increase the cost to attackers by adding complexity to the task of compromising systems. Software can often take advantage of these mechanisms by using APIs provided by the operating system or by enabling the mechanism through compiler or linker options.

This requirement does not mandate that these protections be enabled throughout the Virtualization System—only that they be enabled where they have likely impact. For example, code that receives and processes user input should take advantage of these mechanisms.

### 6.1.5.3 Extended component for Hardware Assists (FPT_HAS_EXT.1)

***FPT_HAS_EXT.1.1*** The VMM shall use ***Intel VT-x*** to reduce or eliminate the need for binary translation.

***FPT_HAS_EXT.1.2*** The VMM shall use ***Intel EPT*** to reduce or eliminate the need for shadow page tables.

***Application Note:*** These hardware-assists help reduce the size and complexity of the VMM, and thus, of the trusted computing base, by eliminating or reducing the need for paravirtualization or binary translation. Paravirtualization involves modifying guest software so that instructions that cannot be properly virtualized are never executed on the physical processor.

### *6.1.5.4 Extended component for Removable Devices and Media (FPT_RDM_EXT.1)*

***FPT_RDM_EXT.1.1*** The TSF shall implement controls for handling the transfer of virtual and physical removable media and virtual and physical removable media devices between information domains.

***FPT_RDM_EXT.1.2*** The TSF shall enforce the following rules when ***CD/DVD, ISO images acting as CD/DVD backend, USB drives*** are switched between information domains, then

1. ***the Administrator has granted explicit access for the media or device to be connected to the receiving domain***

***Application Note:*** The purpose of these requirements is to ensure that VMs are not given inadvertent access to information from different domains because of media or removable media devices left connected to physical machines.

Removable media is media that can be ejected from a device, such as a compact disc, floppy disk, SD, or compact flash memory card.

Removable media devices are removable devices that include media, such as USB flash drives and USB hard drives. Removable media devices can themselves contain removable media (e.g., USB CDROM drives).

For purposes of this requirement, an Information Domain is:

a. A VM or collection of VMs,
b. The Virtualization System,
c. Host OS, or
d. Management Subsystem.

These requirements also apply to virtualized removable media such as virtual CD drives that connect to ISO images as well as physical media such as CDROMs and USB flash drives. In the case of virtual CDROMs, virtual ejection of the virtual media is sufficient.

### *6.1.5.5 Reliable time stamps (FPT_STM.1)*

***FPT_STM.1.1*** The TSF shall be able to provide reliable time stamps.

### *6.1.5.6 Extended component for Virtual Device Parameters (FPT_VDP_EXT.1)*

***FPT_VDP_EXT.1.1*** The TSF shall provide interfaces for virtual devices implemented by the VMM as part of the virtual hardware abstraction.

***FPT_VDP_EXT.1.2*** The TSF shall validate the parameters passed to the virtual device interface prior to execution of the VMM functionality exposed by those interfaces.

***Vendor attestation:*** Parameters passed from Guest VMs to virtual device interfaces are thoroughly validated and all illegal values (as specified in the TSS) are rejected. Additionally, parameters passed from Guest VMs to virtual device interfaces are not able to degrade or disrupt the functioning of other VMs, the VMM, or the Platform. Thorough testing and architectural design reviews have been conducted to ensure the accuracy of these claims, and there are no design or implementation flaws that bypass or defeat the security of the virtual device interfaces.

***Application Note:*** The purpose of this requirement is to ensure that the VMM is not vulnerable to compromise through the processing of malformed data passed to the virtual device interface from a Guest OS. The VMM cannot assume that any data coming from a VM is well-formed even if the virtual device interface is unique to the Virtualization System and the data comes from a virtual device driver supplied by the Virtualization Vendor.

FPT_VDP_EXT.1.2 is an attestation requirement. The vendor must attest that parameters passed from a VM to a virtual device interface are not able to degrade or disrupt the functioning of other VMs, the VMM, or the Platform. The vendor must attest that there are no design or implementation flaws that permit the above.

#### *6.1.5.7 Extended component for VMM Isolation from VMs (FPT_VIV_EXT.1)*

***FPT_VIV_EXT.1.1*** The TSF must ensure that software running in a VM is not able to degrade or disrupt the functioning of other VMs, the VMM, or the Platform.

***FPT_VIV_EXT.1.2*** The TSF must ensure that a Guest VM is unable to invoke platform code that runs at a privilege level equal to or exceeding that of the VMM without involvement of the VMM.

***Vendor attestation:*** Software running in a VM is not able to degrade or disrupt the functioning of other VMs, the VMM, or the Platform. There are no design or implementation flaws that bypass or defeat VM isolation.

***Application Note:*** This requirement is intended to ensure that software running within a Guest VM cannot compromise other VMs, the VMM, or the platform. This requirement is not met if Guest VM software whatever its privilege level can crash the Virtualization System or the Platform, or breakout of its virtual hardware abstraction to gain execution on the platform, within or outside of the context of the VMM.

This requirement is not violated if software running within a VM can crash the Guest OS and there is no way for an attacker to gain execution in the VMM or outside of the virtualized domain.

FPT_VIV_EXT.1.2 addresses several specific mechanisms that must not be permitted to bypass the VMM and invoke privileged code on the Platform.

At a minimum, the TSF should enforce the following:

a) On the x64 platform, a virtual System Management Interrupt (SMI) cannot invoke platform System Management Mode (SMM).
b) An attempt to update virtual firmware or virtual BIOS cannot cause physical platform firmware or physical platform BIOS to be modified.
c) An attempt to update virtual firmware or virtual BIOS cannot cause the VMM to be modified.

Of the above, (a) does not apply to platforms that do not support SMM. The rationale behind activity (c) is that a firmware update of a single VM must not affect other VMs. So if multiple VMs share the same firmware image as part of a common hardware abstraction, then the update of a single machines BIOS must not be allowed to change the common abstraction. The virtual hardware abstraction is part of the VMM.

This is an attestation requirement. The vendor must attest that software running in a VM is not able to degrade or disrupt the functioning of other VMs, the VMM, or the Platform.

The vendor must attest that there are no design or implementation flaws that permit the above.

### 6.1.6 Trusted path/channels (FTP)

#### 6.1.6.1 Extended component for User Interface: I/O Focus (FTP_UIF_EXT.1)

***FTP_UIF_EXT.1.1*** The TSF shall indicate to users which VM, if any, has the current input focus.

***Application Note:*** This requirement applies to all users—whether User or Administrator.

In environments where multiple VMs run at the same time, the user must have a way of knowing which VM user input is directed to at any given moment. This is especially important in multiple-domain environments.

In the case of a human user, this is usually a visual indicator. In the case of headless VMs, the user is considered to be a program, but this program still needs to know which VM it is sending input to; this would typically be accomplished through programmatic means.

The TOE provides access to virtual machines' consoles and serial consoles. For consoles, the TOE provides a different TCP port number for each console. For serial consoles, the TOE only connects a user to the serial console of the virtual machine guest the user was configured for. If the user is allowed to access the serial consoles of multiple virtual machines, the user has to select the intended serial console during login.

#### 6.1.6.2 Extended component for User Interface: Identification of VM (FTP_UIF_EXT.2)

***FTP_UIF_EXT.2.1*** The TSF shall support the unique identification of a VM's output display to users.

***Application Note:*** In environments where a user has access to more than one VM at the same time, the user must be able to determine the identity of each VM displayed in order to avoid inadvertent cross-domain data entry.

There must be a mechanism for associating an identifier with a VM so that an application or program displaying the VM can identify the VM to users. This is generally indicated visually for human users (e.g., a border around a VMs screen display) and programmatically for headless VMs (e.g., an API function). The identification must be unique to the VS, but does not need to be universally unique.

## 6.2 Security Functional Requirements Rationale

### 6.2.1 Coverage

The following table provides a mapping of SFR to the security objectives, showing that each security functional requirement addresses at least one security objective.

| ***Security functional requirements*** | ***Objectives*** |
|---|---|
| FAU_GEN.1 | O.AUDIT |
| FAU_SAR.1 | O.AUDIT |
| FAU_STG.1 | O.AUDIT |
| FDP_HBI_EXT.1 | O.PLATFORM_INTEGRITY |

| ***Security functional requirements*** | ***Objectives*** |
|---|---|
| FDP_PPR_EXT.1 | O.PLATFORM_INTEGRITY,<br>O.VM_ISOLATION,<br>O.VMM_INTEGRITY |
| FDP_RIP_EXT.1 | O.RESOURCE_ALLOCATION,<br>O.VM_ISOLATION |
| FDP_RIP_EXT.2 | O.RESOURCE_ALLOCATION,<br>O.VM_ISOLATION |
| FDP_VMS_EXT.1 | O.PLATFORM_INTEGRITY,<br>O.VM_ISOLATION,<br>O.VMM_INTEGRITY |
| FDP_VNC_EXT.1 | O.PLATFORM_INTEGRITY,<br>O.VM_ISOLATION,<br>O.VMM_INTEGRITY |
| FIA_AFL_EXT.1 | O.MANAGEMENT_ACCESS |
| FIA_PMG_EXT.1 | O.MANAGEMENT_ACCESS |
| FIA_UAU.5 | O.MANAGEMENT_ACCESS |
| FIA_UIA_EXT.1 | O.MANAGEMENT_ACCESS |
| FMT_MOF_EXT.1 | O.MANAGEMENT_ACCESS,<br>O.VMM_INTEGRITY |
| FMT_MSA_EXT.1 | O.CORRECTLY_APPLIED_CONFIGURATION,<br>O.MANAGEMENT_ACCESS,<br>O.VMM_INTEGRITY |
| FMT_SMO_EXT.1 | O.MANAGEMENT_ACCESS |
| FPT_DVD_EXT.1 | O.PLATFORM_INTEGRITY,<br>O.VM_ISOLATION,<br>O.VMM_INTEGRITY |
| FPT_EEM_EXT.1 | O.PLATFORM_INTEGRITY,<br>O.VM_ISOLATION,<br>O.VMM_INTEGRITY |
| FPT_HAS_EXT.1 | O.PLATFORM_INTEGRITY,<br>O.VM_ISOLATION,<br>O.VMM_INTEGRITY |
| FPT_RDM_EXT.1 | O.PLATFORM_INTEGRITY |
| FPT_STM.1 | O.AUDIT |
| FPT_VDP_EXT.1 | O.PLATFORM_INTEGRITY,<br>O.VM_ISOLATION,<br>O.VMM_INTEGRITY |
| FPT_VIV_EXT.1 | O.PLATFORM_INTEGRITY,<br>O.VM_ISOLATION,<br>O.VMM_INTEGRITY |
| FTP_UIF_EXT.1 | O.VM_ISOLATION |
| FTP_UIF_EXT.2 | O.VM_ISOLATION |

***Table 9: Mapping of security functional requirements to security objectives***

## *6.2.2 Sufficiency*

The following rationale provides justification for each security objective for the TOE, showing that the security functional requirements are suitable to meet and achieve the security objectives.

| ***Security objectives*** | ***Rationale*** |
|---|---|
| O.VM_ISOLATION | Met by SFRs for physical platform resource controls and VM separation and isolation: FDP_PPR_EXT.1, FDP_RIP_EXT.1, FDP_RIP_EXT.2, FDP_VMS_EXT.1, FDP_VNC_EXT.1, FPT_DVD_EXT.1, FPT_EEM_EXT.1, FPT_HAS_EXT.1, FPT_VDP_EXT.1, FPT_VIV_EXT.1, FTP_UIF_EXT.1, and FTP_UIF_EXT.2. |
| O.VMM_INTEGRITY | Met by SFRs for VMM management and VM separation and isolation: FMT_MOF_EXT.1, FMT_MSA_EXT.1, FDP_PPR_EXT.1, FDP_VMS_EXT.1, FDP_VNC_EXT.1, FPT_DVD_EXT.1, FPT_EEM_EXT.1, FPT_HAS_EXT.1, FPT_VDP_EXT.1, and FPT_VIV_EXT.1. |
| O.PLATFORM_INTEGRITY | Met by SFRs for physical platform resource controls: FDP_HBI_EXT.1, FDP_PPR_EXT.1, FDP_VMS_EXT.1, FDP_VNC_EXT.1, FPT_DVD_EXT.1, FPT_EEM_EXT.1, FPT_HAS_EXT.1, FPT_RDM_EXT.1, FPT_VDP_EXT.1, and FPT_VIV_EXT.1. |
| O.MANAGEMENT_ACCESS | Met by SFRs for I&A and VMM management: FIA_AFL_EXT.1, FIA_PMG_EXT.1, FIA_UAU.5, FIA_UIA_EXT.1, FMT_MOF_EXT.1, FMT_MSA_EXT.1, and FMT_SMO_EXT.1. |
| O.AUDIT | Met by SFRs for audit generation, review, and storage: FAU_GEN.1, FAU_SAR.1, FAU_STG.1, and FPT_STM.1. |
| O.CORRECTLY_APPLIED_CONFIGURATION | Met by SFR for VMM management: FMT_MSA_EXT.1. |
| O.RESOURCE_ALLOCATION | Met by SFRs for residual data protection: FDP_RIP_EXT.1, and FDP_RIP_EXT.2. |

***Table 10: Security objectives for the TOE rationale***

## *6.2.3 Security Requirements Dependency Analysis*

The security functional requirements in this Security Target do not introduce dependencies on any security assurance requirement; neither do the security assurance requirements in this Security Target introduce dependencies on any security functional requirement.

The following table demonstrates the dependencies of the SFRs modeled in CC Part 2 and the extended component definition in this Security Target, and how the SFRs for the TOE resolve those dependencies.

| *Security functional requirement* | *Dependencies* | *Resolution* |
|---|---|---|
| FAU_GEN.1 | FPT_STM.1 | FPT_STM.1 |
| FAU_SAR.1 | FAU_GEN.1 | FAU_GEN.1 |
| FAU_STG.1 | FAU_GEN.1 | FAU_GEN.1 |
| FDP_HBI_EXT.1 | No dependencies | |
| FDP_PPR_EXT.1 | No dependencies | |
| FDP_RIP_EXT.1 | No dependencies | |
| FDP_RIP_EXT.2 | No dependencies | |
| FDP_VMS_EXT.1 | No dependencies | |
| FDP_VNC_EXT.1 | No dependencies | |
| FIA_AFL_EXT.1 | No dependencies | |
| FIA_PMG_EXT.1 | No dependencies | |
| FIA_UAU.5 | No dependencies | |
| FIA_UIA_EXT.1 | No dependencies | |
| FMT_MOF_EXT.1 | FMT_MSA_EXT.1 | FMT_MSA_EXT.1 |
| FMT_MSA_EXT.1 | No dependencies | |
| FMT_SMO_EXT.1 | No dependencies | |
| FPT_DVD_EXT.1 | No dependencies | |
| FPT_EEM_EXT.1 | No dependencies | |
| FPT_HAS_EXT.1 | No dependencies | |
| FPT_RDM_EXT.1 | No dependencies | |
| FPT_STM.1 | No dependencies | |
| FPT_VDP_EXT.1 | No dependencies | |
| FPT_VIV_EXT.1 | No dependencies | |
| FTP_UIF_EXT.1 | No dependencies | |
| FTP_UIF_EXT.2 | No dependencies | |

***Table 11: SFR dependency analysis***

## *6.3 Security Assurance Requirements*

The security assurance requirements (SARs) for the TOE are defined in [CC] part 3 for the Evaluation Assurance Level 2, augmented by ALC_FLR.3.

The following table shows the SARs, and the operations performed on the components according to CC part 3: iteration (Iter.), refinement (Ref.), assignment (Ass.) and selection (Sel.).

| *Security assurance class* | *Security assurance requirement* | *Source* | *Operations* | | | |
|---|---|---|---|---|---|---|
| | | | *Iter.* | *Ref.* | *Ass.* | *Sel.* |
| ADV Development | ADV_ARC.1 Security architecture description | CC Part 3 | No | No | No | No |
| | ADV_FSP.2 Security-enforcing functional specification | CC Part 3 | No | No | No | No |

| ***Security assurance class*** | ***Security assurance requirement*** | ***Source*** | ***Operations*** | | | |
|---|---|---|---|---|---|---|
| | | | ***Iter.*** | ***Ref.*** | ***Ass.*** | ***Sel.*** |
| | ADV_TDS.1 Basic design | CC Part 3 | No | No | No | No |
| AGD Guidance documents | AGD_OPE.1 Operational user guidance | CC Part 3 | No | No | No | No |
| | AGD_PRE.1 Preparative procedures | CC Part 3 | No | No | No | No |
| ALC Life-cycle support | ALC_CMC.2 Use of a CM system | CC Part 3 | No | No | No | No |
| | ALC_CMS.2 Parts of the TOE CM coverage | CC Part 3 | No | No | No | No |
| | ALC_DEL.1 Delivery procedures | CC Part 3 | No | No | No | No |
| | ALC_FLR.3 Systematic flaw remediation | CC Part 3 | No | No | No | No |
| ASE Security Target evaluation | ASE_INT.1 ST introduction | CC Part 3 | No | No | No | No |
| | ASE_CCL.1 Conformance claims | CC Part 3 | No | No | No | No |
| | ASE_SPD.1 Security problem definition | CC Part 3 | No | No | No | No |
| | ASE_OBJ.2 Security objectives | CC Part 3 | No | No | No | No |
| | ASE_ECD.1 Extended components definition | CC Part 3 | No | No | No | No |
| | ASE_REQ.2 Derived security requirements | CC Part 3 | No | No | No | No |
| | ASE_TSS.1 TOE summary specification | CC Part 3 | No | No | No | No |
| ATE Tests | ATE_COV.1 Evidence of coverage | CC Part 3 | No | No | No | No |
| | ATE_FUN.1 Functional testing | CC Part 3 | No | No | No | No |
| | ATE_IND.2 Independent testing - sample | CC Part 3 | No | No | No | No |
| AVA Vulnerability assessment | AVA_VAN.2 Vulnerability analysis | CC Part 3 | No | No | No | No |

***Table 12: Security Assurance Requirements***

## *6.4 Security Assurance Requirements Rationale*

Dependencies within the EAL package selected (EAL2) for the security assurance requirements have been considered by the authors of CC Part 3 and are not analysed here again. The augmentation by flaw remediation, ALC_FLR.3, has no dependencies on other requirements. The security functional requirements in this Security Target do not introduce dependencies on any security assurance requirement; neither do the security assurance requirements in this Security Target introduce dependencies on any security functional requirement.

The EAL2 level was also deemed sufficient because this will provide a necessary assurance for a product that is operated in an environment that is not directly exposed to external attackers, but still able to resist attacker with basic attack potential.

The assurance requirements of the EAL2 package provides a full Security Target and requires an analysis using a functional and interface specification and a basic description of the architecture of the TOE, which would give sufficient confidence in the design and architecture and for the evaluator to perform an analysis of the design and architecture for the vulnerability analysis.

The augmentation of EAL2 with ALC_FLR.3 is seen useful for software products in a security critical market where the flaw remediation procedures requiring timely response and the automatic distribution of security flaw reports and the associated corrections to registered users who might be affected by the security flaw.

# 7 TOE Summary Specification

## 7.1 TOE Security Functionality

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

The primary security features of the TOE are:

- Audit
- User Data Protection
- Identification and Authentication
- Security Management
- Protection of the TSF
- Trusted Path/Channels

Many security functions in the TOE are implemented in the virtualization environment provided by Red Hat Enterprise Linux (RHEL).

KVM is implemented as part of the Linux kernel supported by user space code. It consists of two essential components that implement VMM functionality: the KVM Linux kernel module and QEMU for hardware emulation. The use of QEMU implies that KVM provides full virtualization to its guests and can, therefore, execute unaltered guest operating systems.

The KVM Linux kernel module implements memory management and virtual machine maintenance functionality. This kernel extension makes the entire Linux kernel the hypervisor. Virtual machines are treated by the Linux kernel as normal applications. The kernel schedules them like applications, and they can be handled like applications. As such, the process implementing a virtual machine can be seen in process listings and it can be sent signals, like SIGTERM.

From the Linux kernel perspective, the virtual machine is just another process. However, the virtual machine process has a special layout. The process image is split into two parts. The first part hosts a regular application logic executing in user mode – this is used to maintain the QEMU I/O virtualization and some other small KVM-related software components. The second part contains the image of the guest code, usually an operating system, where the software may execute either in supervisor or user mode of the processor. This implies that the entire memory used for the guest operating system is allocated by the QEMU application. The kernel keeps track of which parts of the application belong to the guest operating system and which parts to the regular application.

When the kernel releases control of the CPU to the virtual machine process, it sets the processor state of the CPU to the user state when calling the regular application logic in user mode. However, when returning control of the CPU to the guest code, the CPU can be set either to supervisor state or user state, depending on the state of the CPU when the Linux kernel initially obtained control.

The overall logic flow between the Linux kernel, the regular application logic, and the guest operating system is described below. This logic flow is an endless loop which is characterized as follows:

1) The regular application logic executing in user mode sets up the virtual machine configuration by instructing the kernel to allocate memory, CPU, and other resources for the application. The kernel sets up these resources and assigns them to the calling process. After setup is complete, the kernel is instructed to execute the guest. This phase starts the loop and is not executed again during the loop.
2) The kernel now causes the processor to enter guest mode. If the processor exits guest mode due to an event, such as an external interrupt or a shadow page table fault (see section 7.1.2.1 ), the kernel performs the necessary handling and

resumes guest execution. If the exit is due to an I/O instruction or a signal queued to the process, then the kernel exits to the regular application logic in user mode.

3) The processor executes guest code until it encounters an instruction that needs assistance, a fault, or an external interrupt. The processor then returns control to the host kernel.
4) If the host kernel detects an exit of the guest code due to an I/O instruction or a signal, or until an external event such as arrival of a network packet or a timeout occurs, the kernel invokes the user mode component of the virtual machine process. The processed I/O instructions cover programmed I/O (PIO) whose implementation is not as complex as the second set of processed I/O instructions, the memory mapped I/O (MMIO). QEMU, with a small extension for making QEMU KVM-aware, is used to implement the I/O handling, implementing a number of emulated devices and mediate access to real resources when a device is accessed via the I/O instruction. QEMU may alter the virtual processor state to reflect the emulated I/O instruction result to the calling guest code. The modification of the virtual processor state is done with IOCTLs to a file descriptor that QEMU has allocated when setting up the virtual machine. This file descriptor is used to store all virtual machine and virtual CPU data relevant for executing the virtual machine. As the file descriptor is bound to one process only, the kernel implicitly ensures that a QEMU instance can only operate on its virtual machine and virtual CPUs. Once QEMU completes the I/O operation, it signals the kernel that the guest code can resume execution which is implemented by step 2 above.

In this architecture, regular applications (i.e., applications executing only in user state of the CPU and having full access to all services of the Linux kernel and, therefore, other parts of the operating system) coexist with applications that host virtual machines.

The libvirtd management daemon sets up virtual machines and controls the resources assigned to virtual machines. To support the separation of virtual machines, libvirtd uses the following capabilities:

- Every virtual machine process executes with the normal, unprivileged user ID of “qemu” and the group ID of “qemu”. This implies that these processes do not possess any Linux kernel capability.
- Libvirtd sets up the unique SELinux label for a virtual machine and assigns it to the virtual machine and its resources. The resource access control functionality is defined as an independent security functionality.
- Libvirtd can instruct the kernel to set up the IOMMU in a way to exclusively assign hardware resources to a virtual machine process as an independent security functionality.
- Every virtual machine process will be placed in a dedicated control group or cgroup. Cgroup is a mechanism of the Linux kernel to mark processes and assign certain properties to these processes. Every process spawned by an already-marked process will bear the same identifier. Using the device whitelist controller with the cgroup mechanism, ACLs on devices are implemented. Libvirtd uses cgroups to restrict access of each virtual machine process to only the devices assigned to this virtual machine, even though ordinary UNIX permission bits would have granted access to these devices. Please note that this mechanism only applies when the disk resource granted to a virtual machine is based on iSCSI, LVM or SANs. It does not apply to backends like regular files, NFS or others.

Virtual machines are associated with one or more unique IP addresses that can be used to communicate with other virtual machines on the same host or with other external entities. The kernel ensures that the configured IP addresses are used by the virtual machines for any network-related communication.

The oVirt management framework supports, but does not enforce, the SFRs in this Security Target. Figure 3, Virtualization System and Platform shows the relationship between oVirt and the previously described Linux components.

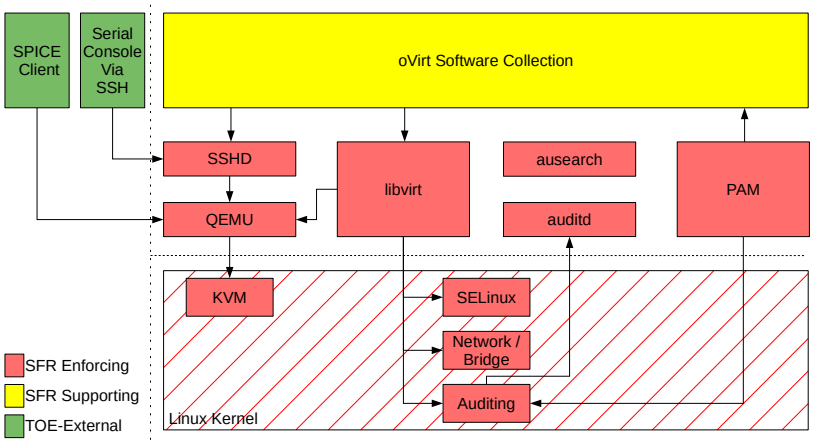


***Figure 3, Virtualization System and Platform***

Once QEMU is spawned, it interacts with the Linux kernel KVM framework to provide one virtual machine instance. Together with the KVM framework, QEMU provides the execution environment for the guest operating system. This includes the virtualization and para-virtualization of all resources that are referenced via the command line parameters. For example, a storage resource may be presented to the guest as a VirtIO storage device by QEMU.

QEMU implements a SPICE daemon that may be enabled if configured via the oVirt management framework. SPICE is the protocol used to access the virtual machine console. An external SPICE client connects to the QEMU SPICE daemon to access this virtual machine console. When oVirt triggers the instantiation of a virtual machine via libvirt, it also provides the network port to be used for the SPICE communication of this QEMU instance. In order for the SPICE client to know which port to access, oVirt provides the connection details to the SPICE client.

In addition to the virtual machine console, QEMU externalizes the serial console of the guest operating system if configured by the oVirt management framework. The serial console is accessed by a client via SSH. oVirt instantiates a separate SSH daemon which connects to the QEMU serial console interface using TLS. The SSH daemon is configured to only allow SSH-key-based authentication. oVirt provides the following data to the SSH daemon:

- oVirt manages the SSH keys used for authentication. The SSH daemon obtains the keys from oVirt to implement and enforce the key-based authentication.
- oVirt knows which QEMU instances are available on the system, thus it provides the information to SSHD to ensure that a user authenticated by SSHD is only connected to the serial console of the intended QEMU instance.

### *7.1.1 Audit*

Auditing support is implemented using the Linux Auditing Framework (LAF). libvirt generates audit entries which are stored on disk by the audit daemon. The audit data can be reviewed using the ausearch utility.

The Linux 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. 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.

#### *7.1.1.1 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 need to be passed 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 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 sent to the user space audit daemon for storage in the audit trail. The kernel sends the message to the audit daemon 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 kernel switches into single-user mode and 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 are not lost due to a 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 kernel, 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 Linux Audit Framework offers 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
- SELinux label components

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.

This security functionality supports FAU_GEN.1.

#### 7.1.1.2 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)
- Success or failure (where appropriate)
- SELinux label of the subject that caused the event
- Process ID of the subject that caused the event (PID)
- Hostname or terminal the subject used 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 ausearch application allows selective extraction of records from the audit trail using defined selection criteria. Using the ausearch application, 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 reaches a warning threshold, the administrator is notified. If the audit trail is full, the audit daemon rejects new audit logs from the kernel to store and 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, preventing additional auditable events. In addition, the audit daemon can inform the administrator of the full audit trail, can switch to single-user mode, or can halt the system, depending on the configuration.

This security functionality supports FAU_SAR.1 and FAU_STG.1.

### 7.1.2 User Data Protection

The TOE uses hardware-based isolation mechanisms and physical platform resource controls to protect user data.

#### 7.1.2.1 Required hardware support

The TOE uses Linux kernel mechanisms, which in turn use hardware-based mechanisms, to constrain VMs when VMs have direct access to physical devices.

The Linux kernel uses various hardware mechanisms to provide the virtualization support and ensure proper separation of virtual machines. The following enumerates the hardware support and indicates whether the underlying hardware must provide the respective functionality to achieve full separation.

***IOMMU virtualization support***

If PCI device assignment is configured, the hardware must provide the IOMMU mechanism located on the North Bridge chip set. Intel chip sets with the VT-d support implement the IOMMU support. The KVM Linux kernel support can only implement the PCI device assignment functionality when the chip set implements the mentioned IOMMU mechanisms and they are enabled.

This security functionality supports FDP_HBI_EXT.1.

#### *7.1.2.2 Network resources*

The TOE provides a central location for users to perform logical network-related operations and search for logical networks based on each network's property or association with other resources. The following operations are allowed:

- Attaching or detaching the networks to hosts
- Removing network interfaces from virtual machines and templates

The TOE does not provide any other means for inter-virtual-machine-communication apart from these network administration functions.

The TOE allows configuration of virtual switches and network interfaces for establishing a communication between virtual machines within a host. In addition, the TOE can be configured to link a virtual switch or a virtual interface with a physical interface to allow a virtual machine to communicate outside of the host.

This security functionality supports FDP_VMS_EXT.1 and FDP_VNC_EXT.1.

#### *7.1.2.3 Residual Information Protection*

The TOE protects against residual information being left behind in both memory and on disk after use by a virtual machine.

##### *In Memory*

The TOE utilizes the underlying Linux kernel to manage memory. Each time a process is given new memory, the kernel allocates the requested amount of memory pages which are all zeroized before the process can access them.

This security functionality supports FDP_RIP_EXT.1.

##### *On Disk*

The TOE utilizes the underlying Linux kernel to manage information on disk. The Linux kernel ensures that when a new sparse file is created, accessing any block for the first time within that file will return a zero page. When pre-allocating file-based storage, the TOE ensures that the entire file contains zeros. Only when mapping a physical disk device directly from the host would existing data be available to a guest VM.

This security functionality supports FDP_RIP_EXT.2.

#### *7.1.2.4 Physical Platform Resource Controls*

The TOE relies on the fact that each virtual machine is represented by a separate Linux process. Each process has a separate instance of the QEMU application previously described to provide the virtualized environment.

This security functionality supports FDP_PPR_EXT.1.

### *7.1.3 Identification and Authentication*

The TOE uses RHEL authentication and password management functions. All administrative users must always authenticate when accessing consoles. The TOE utilizes the underlying Linux authentication mechanisms. The TOE (oVirt) authenticates users by using the PAM library offered by the basic Linux operating system. Using PAM authentication of users with the local user database is achieved. To access remote authentication providers like LDAP or Active Directory, oVirt uses appropriate PAM configurations. Remote authentication providers implement the authentication of users and return the information about authentication decisions to PAM which forwards the result to oVirt for enforcement. PAM also links with the Linux Auditing Framework to provide the capability to audit authentication requests.

Users and their credentials are allowed to be managed via an external directory server. A variety of directory server products are supported by the TOE.

This security functionality supports FIA_AFL_EXT.1, FIA_PMG_EXT.1, FIA_UAU.5, and FIA_UIA_EXT.1.

### *7.1.4 Security Management*

Hosts, also known as hypervisors, are the physical servers on which virtual machines run. Full virtualization is provided by using a loadable Linux kernel module called Kernel-based Virtual Machine (KVM).

KVM can concurrently host multiple virtual machines running either Windows or Linux operating systems. Virtual machines run as individual Linux processes and threads on the host machine and are managed remotely by the Red Hat Virtualization Manager. A Red Hat Virtualization environment has one or more hosts attached to it.

The security management of covers the entire life-cycle of Virtual Machines from the creation, starting stopping, restarting and deletion of VMs.

The oVirt management framework allowing users – considered to be an administrator in terms of the evaluation – to manage resources assigned to virtual machines as well as virtual machines. Management operations that are applied to resources like networking options, storage options, etc. are applied to objects that are not yet part of virtual machines. Such operations may include the preparation of storage objects like creating file system objects that eventually will become backends for virtual machine disk images. All these operations are outside of the security claims of this ST. At one point, however, a user may perform an operation on virtual machines. Such operations include start or stop of virtual machines or other operations.

The oVirt translates all operations into XML requests that are forwarded to the libvirt API. These are the same as requests that can be sent via virsh or any other invocation of the libvirt API. Note the API is reachable via the network since libvirt is a daemon. These XML requests received by the libvirt API are transformed into command line parameters with which the QEMU application is invoked. The invocation of the QEMU application assigns the resources managed by oVirt to QEMU. All resources the QEMU application shall use for virtualization, such as storage backends, are referenced as command line parameters to QEMU. As part of the invocation of the QEMU application, libvirt configures the Linux kernel infrastructure, like SELinux labels for the QEMU resources and QEMU, configures the networking options, like bridges and assignment of QEMU networking interfaces to bridges. In addition, libvirt contains auditing support which may cause audit entries to be generated for administrative actions.

Once QEMU is spawned, it interacts with the Linux kernel KVM framework to provide one virtual machine instance. Together with the KVM framework, QEMU provides the execution environment for the guest operating system. This includes the virtualization and para-virtualization of all resources that are referenced via the command line parameters. For example, a storage resource may be presented to the guest as a VirtIO storage device by QEMU.

This security functionality supports FMT_MOF_EXT.1, and FMT_MSA_EXT.1.

To ensure that the management and operational networks maintain separation, [ECG] outlines how a separate, dedicated administrative LAN is established with the TOE in the evaluated configuration.

This security functionality supports FMT_SMO_EXT.1.

### *7.1.5 Protection of the TSF*

The TOE provides functionality to mitigate the effects of buffer overrun vulnerabilities in applications and to protect against residual information remaining in memory or on disk between uses by different virtual machine instances.

#### *7.1.5.1 Buffer Overrun Protection*

The TOE provides mechanisms to prevent or significantly increase the complexity of exploitation of common buffer overflow and similar attacks. These mechanisms are used for the TSF and are available to untrusted code.

Runtime protection against programming errors is provided by implementing multiple countermeasures against exploitation of such errors. Standard programming errors, such as buffer overflows, are exploitable using known techniques. The TOE blocks or significantly increases the challenge to use these techniques with the following different approaches:

- Prevention of code execution on the process' or thread's stack. This prevents standard buffer overflow attacks which write executable code (e.g. the shellcode) into a stack variable and causes the CPU to execute it. A guard variable is added to functions with vulnerable objects and is checked for correctness after a function ends and before the function pointed to by the return address is evaluated. This guard variable is also known as Stack Canary. The variable is a random number which is generated during startup of an application and used throughout the lifetime of the process.
- Performing address space randomization which affects position independent code (PIC) as well as position independent executables (PIE). All shared libraries are necessarily PIC, whereas only dedicated TSF applications are PIE. Users may compile their applications as PIE to allow address space randomization protect their applications. Using Address Space Layout Randomization (ASLR) configured by the Linux kernel, the layout of the address space covering the memory pages as well as the stack, the location of application and library code segments with their symbols differs for each instantiation of a process. This means that even when the same binary is started in a second process, its address space is completely different than the layout of the first instance. This approach makes is very hard for attackers to modify the return address stored on the stack during a buffer overrun exploit. Therefore, using code present in the binaries to mount an attack is rendered difficult as the attacker does not know the address of the memory segments he wants the CPU to execute. Note, this approach does not technically prevent the exploitation, but adds significant barriers to doing so successfully.
- Marking the runtime memory of all parts of a binary as read-only apart from heap data before the loaded application gains control. This support is enabled for dedicated TSF applications and specifically compiled user applications. Partial protection is also possible which implies that also the .got.plt section is marked read/writable. The technique called read-only relocation (RELRO) is implemented by the Linux loader and linker to mark memory as read only. This memory contains the ELF segments holding the global object table (GOT) and procedure linking table (PLT) after the resolution by the linker but before the main() function of the application is called. These sections store offset tables required for the dynamic linking mechanism and, if abused, allow attackers to modify the jump addresses of object accesses. Marking these memory segments read-only requires the dynamic linker to resolve all library symbols of shared libraries during load time of the application. Marking the PLT as read only incurs a significant performance penalty. Therefore, the TOE implements two types of RELRO: partial and full.
    - Full RELRO support means that for an application and all dependent shared libraries, the PLT is set read only in addition to all ELF header sections other than the data segments. If at least one dependent library is not compiled with full RELRO, the entire application cannot be claimed to have full RELRO. Note, the TOE code does not use full RELRO.
    - Partial RELRO means that for an application and all dependent libraries, all ELF sections except the data segments are marked read only. The PLT is not marked read only in either the application or at least one dependent

shared library. If at least one dependent library is compiled without RELRO, the entire application cannot be claimed to have partial RELRO.

- The FORTIFY_SOURCE macro provides lightweight support for detecting buffer overflows in various functions that perform operations on memory and strings. Not all types of buffer overflows can be detected with this macro, but it does provide an extra level of validation for some functions that are potentially a source of buffer overflow flaws. It protects both C and C++ code. FORTIFY_SOURCE works by computing the number of bytes that are going to be copied from a source to the destination. All packages are compiled with -D_FORTIFY_SOURCE=2.

On Intel CPUs starting with Ivy Bridge, the CPU feature SMEP is employed which prevents the kernel from executing code located in user space memory.

Intel CPUs starting with Haswell CPUs offers the feature SMAP. This feature is employed by the Linux kernel which ensures that the Linux kernel cannot read from or write to user space memory.

This security functionality supports FPT_EEM_EXT.1.

### *7.1.5.2 Storage Management*

A storage domain is a collection of images that have a common storage interface. A storage domain contains complete images of templates and virtual machines (including snapshots), or ISO files. A storage domain can be made of block devices (SAN - iSCSI or FCP) or a file system (NAS - NFS, GlusterFS, or other POSIX compliant file systems).

On NFS, all virtual disks, templates, and snapshots are files.

On SAN (iSCSI/FCP), each virtual disk, template or snapshot is a logical volume. Block devices are aggregated into a logical entity called a volume group, and then divided by LVM (Logical Volume Manager) into logical volumes for use as virtual hard disks. See Red Hat Enterprise Linux Logical Volume Manager Administration Guide for more information on LVM.

Virtual disks can have one of two formats, either QCOW2 or raw. The type of storage can be sparse or preallocated. Snapshots are always sparse but can be taken for disks of either format.

Virtual machines that share the same storage domain can be migrated between hosts that belong to the same cluster.

By default, in an NFS, local, or POSIX compliant data center, the SPM creates the virtual disk using a thin provisioned format as a file in a file system.

In iSCSI and other block-based data centers, the SPM creates a volume group on top of the Logical Unit Numbers (LUNs) provided and makes logical volumes to use as virtual disks. Virtual disks on block-based storage are preallocated by default.

Reassignment of removable media is only allowed for read-only media like CD/DVD or ISO images presented as virtual CD/DVD to a guest.

This security functionality supports FPT_RDM_EXT.1.

### *7.1.5.3 Required hardware support*

The TOE uses Linux kernel mechanisms, which in turn use hardware-based mechanisms, to constrain VMs when VMs have direct access to physical devices.

The Linux kernel uses various hardware mechanisms to provide the virtualization support and ensure proper separation of virtual machines. The following enumerates the hardware support and indicates whether the underlying hardware must provide the respective functionality to achieve full separation.

***Processor virtualization support***

The Linux kernel uses the Intel VT-x processor support to allow untrusted software to execute in user mode and supervisor mode. The KVM mechanism of the Linux kernel is only operational if these support mechanisms are implemented. This evaluation applies only when the processor virtualization support is present and

enabled. The kernel marks the enabled virtualization support in /proc/cpuinfo as the CPU flag vmx (Intel).

***Shadow page table support***

The Linux kernel uses the shadow page table support provided with the CPU (Intel refers to this mechanism as EPT - extended page table support). The shadow page table support ensures that the processor handles guest software access to the paging configuration and the associated page-table walks. The evaluation applies only if the underlying processor has the shadow page table support enabled and ready for use. For x86 CPUs, the kernel marks the enabled shadow page table support in /proc/cpuinfo as the CPU flag ept (Intel) and npt (AMD).

This security functionality supports FPT_HAS_EXT.1.

### *7.1.5.4 Resource access control for virtual machines*

The TOE implements the following types of access control restrictions to limit access of virtual machines to only their resources:

- SELinux-based: each virtual machine and its resources are assigned a unique SELinux label which prevents other virtual machines with a different label from accessing either the virtual machine process or its resources.
- IOMMU-based: The TOE is able to configure the IOMMU of the underlying machine to bind the DMA address space of a particular physical resource to be only visible in the address space of the virtual machine process configured to access that resource.
- Cgroup-based: each virtual machine is granted access to a whitelist of device files. Access to other device files is prevented using the cgroup device ACL mechanism.

The SELinux mechanism is implemented as a Linux Security Module (LSM) that is invoked for each and every operation of a subject on an object or resource. The SELinux policy enforced for virtual machines therefore prevent virtual machines from accessing:

- Resources assigned to other virtual machines,
- Resources owned by other users on the system,
- Resources belonging to the host operating system.

This security functionality supports FPT_DVD_EXT.1.

### *7.1.5.5 Virtual Device Parameters*

The TOE supports the following types of interfaces:

- Hypercalls: A guest can issue hypercalls to access para-virtualized host services implemented in the kernel.
- Exceptions: A guest can trigger exceptions that must be caught by the host kernel. Those exceptions may be handled by the kernel (such as interrupts, most instruction completion operations) or by QEMU associated with the guest for more complex exceptions (such as interaction with fully virtualized devices or para-virtualized devices provided by QEMU). In case of para-virtualized devices, the VirtIO framework is used for communication between the guest and the host.
- VNC: The QEMU application associated with a guest allows access to a guest's console via VNC. That VNC communication channel may be tunneled through cryptographic channels if configured by the administrator.

This security functionality supports FPT_VDP_EXT.1 and FPT_VIV_EXT.1.

### *7.1.5.6 Reliable Time Stamps*

Audit logs depend on reliable time stamps. The RHEL operating system, which is part of the TOE, provides an accurate time function. The time is set by the administrator.

This security functionality supports FPT_STM.1.

### 7.1.6 Trusted Path/Channels

The TOE implements trusted paths and trusted channels using components of the RHEL virtual machine environment.

#### 7.1.6.1 User Interface

##### I/O Focus

The user or administrator can connect to the VM console using either the SPICE or VNC graphics protocols which communicate through QEMU. From the perspective of the virtual machine, focus is always on this connection. Focus for the user's keyboard and mouse is managed by the system and/or window manager they use to make their connection to the TOE or a VM.

This security functionality supports FTP_UIF_EXT.1.

##### Identification of VM

Access to virtual machines is only possible via network access: either network access of the guest operating system or via VNC-offered QEMU for the guest. Each virtual machine has one or more unique IP addresses for network access allowing an unambiguous identification. When using VNC, each VNC server endpoint maintained by a QEMU instance has a unique identifier which allows unique identification of the accessed virtual machine.

This security functionality supports FTP_UIF_EXT.2.

# 8 Abbreviations, Terminology and References

## 8.1 Abbreviations

## 8.2 Terminology

This section contains definitions of technical terms that are used with a meaning specific to this document. Terms defined in the [CC] are not reiterated here, unless stated otherwise.

***Administrator***

Administrators perform management activities on the VS. These management functions do not include administration of software running within Guest VMs, such as the Guest OS. Administrators need not be human as in the case of embedded or headless VMs. Administrators are often nothing more than software entities that operate within the VM.

***Auditor***

Auditors are responsible for managing the audit capabilities of the TOE. An Auditor may also be an Administrator. It is not a requirement that the TOE be capable of supporting an Auditor role that is separate from that of an Administrator.

***Domain***

A Domain or Information Domain is a policy construct that groups together execution environments and networks by sensitivity of information and access control policy. For example, classification levels represent information domains. Within classification levels, there might be other domains representing communities of interest or coalitions. In the context of a VS, information domains are generally implemented as collections of VMs connected by virtual networks. The VS itself can be considered an Information Domain, as can its Management Subsystem.

***EPT***

Extended Page Table

***FCP***

Fibre Channel Protocol (FCP) is a protocol that transports SCSI commands over Fibre Channel networks.

***Guest Network***

See Operational Network.

***Guest Operating System (OS)***

An operating system that runs within a Guest VM.

***Guest VM***

A Guest VM is a VM that contains a virtual environment for the execution of an independent computing system. Virtual environments execute mission workloads and implement customer-specific client or server functionality in Guest VMs, such as a web server or desktop productivity applications.

***Helper VM***

A Helper VM is a VM that performs services on behalf of one or more Guest VMs, but does not qualify as a Service VM—and therefore is not part of the VMM. Helper VMs implement functions or services that are particular to the workloads of Guest VMs. For example, a VM that provides a virus scanning service for a Guest VM would be considered a Helper VM. For the purposes of this document, Helper VMs are considered a type of Guest VM, and are therefore subject to all the same requirements, unless specifically stated otherwise.

***Host Operating System (OS)***

An operating system onto which a VS is installed. Relative to the VS, the Host OS is part of the Platform.

***Hypercall***

An API function that allows VM-aware software running within a VM to invoke VMM functionality.

***Hypervisor***

The Hypervisor is part of the VMM. It is the software executive of the physical platform of a VS. A Hypervisor's primary function is to mediate access to all CPU and memory resources, but it is also responsible for either the direct management or the delegation of the management of all other hardware devices on the hardware platform.

***Information Domain***

See Domain.

***Introspection***

A capability that allows a specially designated and privileged domain to have visibility into another domain for purposes of anomaly detection or monitoring.

***iSCSI***

Internet Small Computer Systems Interface is an IP-based storage networking standard providing block-level access to storage devices by carrying SCSI commands over a TCP/IP network.

***KVM***

Kernel-based Virtual Machine (KVM) is a virtualization module in the Linux kernel that allows the kernel to function as a hypervisor.

***LAF***

Linux Audit Framework (LAF) is designed to be an audit system making Linux compliant with the requirements from Common Criteria.

***LSM***

Linux Security Modules is a framework that allows the Linux kernel to support a variety of computer security models and is a standard part of the Linux kernel.

***LVM***

Logical Volume Manager (LVM) is a device mapper target that provides logical volume management for the Linux kernel.

***Management Network***

A network, which may have both physical and virtualized components, used to manage and administer a VS. Management networks include networks used by VS Administrators to communicate with management components of the VS, and networks used by the VS for communications between VS components. For purposes of this document, networks that connect physical hosts for purposes of VM transfer or coordinate, and back-end storage networks are considered management networks.

***Management Subsystem***

Components of the VS that allow VS Administrators to configure and manage the VMM, as well as configure Guest VMs. VMM management functions include VM configuration, virtualized network configuration, and allocation of physical resources.

***NFS***

Network File System (NFS) is a distributed file system protocol allowing a user on a client computer to access files over a computer network as if it resided on a local storage device.

***Operational Network***

An Operational Network is a network, which may have both physical and virtualized components, used to connect Guest VMs to each other and potentially to other

entities outside of the VS. Operational Networks support mission workloads and customer-specific client or server functionality. Also called a “Guest Network.”

***Physical Platform***

The hardware environment on which a VS executes. Physical platform resources include processors, memory, devices, and associated firmware.

***Platform***

The hardware, firmware, and software environment into which a VS is installed and executes.

***QEMU***

QEMU is an open source machine emulator and virtualizer.

***RELRO***

Read-only relocation of runtime memory segments.

***SAN***

Storage Area Network is a computer network which provides access to consolidated, block-level data storage.

***Service VM***

A Service VM is a VM whose purpose is to support the Hypervisor in providing the resources or services necessary to support Guest VMs. Service VMs may implement some portion of Hypervisor functionality, but also may contain important system functionality that is not necessary for Hypervisor operation. As with any VM, Service VMs necessarily execute without full Hypervisor privileges — only the privileges required to perform its designed functionality. Examples of Service VMs include device driver VMs that manage access to a physical device, and name-service VMs that help establish communication paths between VMs.

***SPICE***

Simple Protocol for Independent Computing Environments remote display system for virtual environments.

***SPM***

Storage Pool Manager (SPM) is a role given to one of the hosts in the data center enabling it to manage the storage domains of the data center.

***System Security Policy (SSP)***

The overall policy enforced by the VS defining constraints on the behavior of VMs and users.

***User***

Users operate Guest VMs and are subject to configuration policies applied to the VS by Administrators. Users need not be human as in the case of embedded or headless VMs, users are often nothing more than software entities that operate within the VM.

***Virtualization System (VS)***

A software product that enables multiple independent computing systems to execute on the same physical hardware platform without interference from one other. For the purposes of this document, the VS consists of a Virtual Machine Manager (VMM), Virtual Machine (VM) abstractions, a management subsystem, and other components.

***Virtual Machine (VM)***

A Virtual Machine is a virtualized hardware environment in which an operating system may execute.

***Virtual Machine Manager (VMM)***

A VMM is a collection of software components responsible for enabling VMs to function as expected by the software executing within them. Generally, the VMM consists of a Hypervisor, Service VMs, and other components of the VS, such as virtual devices, binary translation systems, and physical device drivers. It manages concurrent execution of all VMs and virtualizes platform resources as needed.

***VNC***

Virtual Network Computing graphical desktop sharing system.

## *8.3 References*


***BVPPv1.0*** ***Protection Profile for Virtualization Version 1.0***
Version 1.0, 2016-11-22
https://www.niap-ccevs.org/MMO/PP/pp_base_virtualization_v1.0.pdf

***CC*** ***Common Criteria for Information Technology Security Evaluation***
Version 3.1R5, April 2017
http://www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R5.pdf
http://www.commoncriteriaportal.org/files/ccfiles/CCPART2V3.1R5.pdf
http://www.commoncriteriaportal.org/files/ccfiles/CCPART3V3.1R5.pdf

***ECG*** ***EAL2 Evaluated Configuration Guide for Red Hat Virtualization 4.3***
Version 0.9, 2021-11-08

***RHVAG*** ***Red Hat Virtualization 4.3 Administration Guide***
Red Hat, Inc., 2020-10-06
https://access.redhat.com/documentation/en-us/red_hat_virtualization/4.3/html/administration_guide/index

***RHVPG*** ***Red Hat Virtualization 4.3 Product Guide***
Red Hat, Inc., 2020-04-20
https://access.redhat.com/documentation/en-us/red_hat_virtualization/4.3/html-single/product_guide/index

***RHVPPG*** ***Red Hat Virtualization 4.3 Planning and Prerequisites Guide***
Red Hat, Inc., 2020-03-27
https://access.redhat.com/documentation/en-us/red_hat_virtualization/4.3/html/planning_and_prerequisites_guide/index

***RHVTR*** ***Red Hat Virtualization 4.3 Technical Reference***
Red Hat, Inc., 2020-04-20
https://access.redhat.com/documentation/en-us/red_hat_virtualization/4.3/html/technical_reference/index

***SVEPv1.0*** ***Extended Package for Server Virtualization Version 1.0***
Version 1.0, 2016-11-22
https://www.niap-ccevs.org/MMO/PP/ep_sv_v1.0.pdf