Chapter 1 Introduction

This section identifies the PP-Module as well as the Base PP and provides a Module overview for potential users.

1.1 PP Module Reference

Title: MILS Platform Protection Profile Management Module
Sponsor: certMILS Consortium
CC Version: 3.1 (Revision 5)
Assurance Level: see the Base PP.
Version: draft
Keywords: Base-PP, PP-module, Operating System, Separation Kernel, MILS

1.2 Base PP Identification

Base MILS Platform Protection Profile, Version: 1.0

1.3 PP Module Overview

This PP module supplements the Base PP by adding services for management of APIs and and/or objects. An API is a set of functions. An object is a logical abstraction of a resource (for examples of objects see Section 8.3).

Chapter 2 Consistency Rationale

This section states the correspondence between the PP-Module and its Base PP.

2.1 TOE type consistency

The TOE type for which both the Base PP and this PP Module are designed is “a special kind of operating system, namely an SK.”

An SK is a special kind of operating system that allows to effectively separate different containers called “partitions” from each other. Applications themselves are hosted in those partitions. They can also be entire operating systems. The SK is installed and runs on a hardware platform (e.g. embedded systems, desktop class hardware).

The PP module extends the Base PP by specifying management services for the TOE. Such management services can be optionally provided by an SK that is compliant to the certMILS base PP.

2.2 Security Problem Definition consistency

2.2.1 Assets

The section 3.1 of the Base PP describes the assets to be protected:

- Memory
- CPU time

This PP Module adds the following assets:

- API
- Objects

The new assets are independent and compatible with the assets defined in the Base PP as it does not interfere with the protection of the **Memory** or **CPU time**. It adds protection to the API that acts on the entire TOE and to the objects that can be created in a partition.

2.2.2 Threats

The section 3.2 of the Base PP describes the threats contemplated:

- T.DISCLOSURE
- T.MODIFICATION
- T.DEPLETION

This PP Module contemplates the following additional threat:

- T.EXECUTION

This new threat is independent and compatible with the set of threats defined in the Base PP as it is focused on SKs that provide management functions not specified in the base PP. T.EXECUTION covers a new threat scenario where attackers those execute these management functions, specified in this module, without being authorized to do so.

2.2.3 *Organizational Security Policies*

Neither the Base PP nor this PP Module define organizational security policies.

2.2.4 *Assumptions*

This PP Module does not define additional assumptions. The assumptions defined in section 3.4 of the Base PP are applicable with no changes.

2.3 Security Objectives consistency

The section 4.1 of the Base PP describes the security objectives to be implemented:

- OT.CONFIDENTIALITY
- OT.INTEGRITY
- OT.AVAILABILITY

This PP Module adds the following security objective for the TOE:

- OT.API_PROTECTION

This security objective adds security functionality to the TOE regarding the API protection which is compatible to the rest of security objectives for the TOE defined in the Base PP.

2.4 Security Functional Requirements consistency

In addition to the set of SFRs included in section 6.1 of the Base PP, this PP Module defines:

- FMT_MTD.1/API Management of TSF Data–API – This SFR is compatible with the set of SFRs defined in the Base PP as it adds independent functionality regarding restrictions to access the API by trusted partitions.
- FMT_MTD.1/OBJ Management of TSF Data–Objects – This SFR is compatible with the set of SFRs defined in the Base PP as it adds independent functionality regarding restrictions to access the objects by trusted partitions.
- FMT_SMF.1 Specification of Management Functions – This SFR adds management functionality to the FMT_SMF.1 SFR included in the Base PP. ST authors may either iterate this SFR or extend the Base PP FMT_SMF.1 by adding specific management functionality for APIs and objects.
- FMT_SMR.1 Security Roles – This SFR contemplates the role management by associating roles to partitions. This SFR is compatible with the set of SFRs defined in the Base PP.

Chapter 3 Conformance claim

This protection profile module claims conformance to

- Common Criteria for Information Technology Security Evaluation. Part 1: Introduction and general model. Version 3.1, Revision 5. April 2017. CCMB-2017-04-001 [1]
- Common Criteria for Information Technology Security Evaluation. Part 2: Security Functional Components. Version 3.1, Revision 5. April 2017. CCMB-2017-04-002 [2]
- Common Criteria for Information Technology Security Evaluation. Part 3: Security Assurance Components. Version 3.1, Revision 5. April 2017. CCMB-2017-04-003 [3]

as follows

- Part 2 conformant,

The "Common Methodology for Information Technology Security Evaluation, Evaluation Methodology; Version 3.1, Revision 5, April 2017. CCMB-2017-04-004, [4]" has to be taken into account.

This protection profile module is associated with the Base MILS Platform Protection Profile Version 1.0.

3.1 Conformance Rationale

Since a PP module cannot claim conformance to any protection profile, this section is not applicable.

3.2 Conformance Statement

This Protection Profile Module requires strict conformance of any ST or PP claiming conformance to this PP Module.

Note: claiming conformance to this PP Module also requires claiming conformance to the Base MILS Platform Protection Profile.

Chapter 4 Security Problem Definition

This section describes the security aspects of the environment in which the TOE claiming conformance with the PP will be used and the manner in which the TOE is expected to be employed. It provides the statement of the TOE security environment, which identifies and explains all:

- Known and presumed threats countered by either the TOE or by the security environment.
- Organizational security policies with which the TOE must comply.
- Assumptions about the secure usage of the TOE.

4.1 Assets

Asset Name	Description
API (AS.API)	This asset is a management API that e.g. acts on the entire TOE. Application note: e.g. this API contains functions to shutdown the entire TOE.
Objects (AS.OBJ)	The asset consists of objects (e.g. tasks, threads) that can be created in a partition.

Table 1: Assets

4.2 Threats

Assets are defined in Table 1 in Section 4.1. The attackers are the defined in the Base PP.

T.EXECUTION

An attacker executes a management function without being authorized to do so.

4.3 Organizational Security Policies

This module defines no organizational security policies.

4.4 Assumptions

The assumptions are the same as in the base PP.

Chapter 5 Security Objectives

Security objectives are concise, abstract statements of the intended solution to the problem defined by the security problem definition (see previous section). The set of security objectives for a TOE form a high-level solution to the security problem. This high-level solution is divided into two part-wise solutions: the security objectives for the TOE, and the security objectives for the TOE's operational environment.

This section presents the solution to the security problem in terms of objectives for the TOE and its operational environment.

5.1 Security Objectives for the TOE

OT.API_PROTECTION

The TSF shall prevent any execution of a management function unless it is allowed to a partition by the integrator.

5.2 Security Objectives for the Operational Environment

The security objectives for the operational environment are the same as for the base TOE.

5.3 Security Objectives Rationale

	OT.API_PROTECTION
T.EXECUTION	X

Table 2: Security Objectives Rationale

T.EXECUTION

If the security objective OT.API_PROTECTION has been reached, the threat T.EXECUTION is completely eliminated.

Chapter 6 Extended Components Definition

This module does not define any extended component.

Chapter 7 Security Requirements

This section defines the Security Functional requirements (SFRs) in relationship with the set of TOE security objectives in the PP-Module and with the security functional requirements of the Base-PP. This PP Module does not introduce specific assurance requirements. The assurance requirements are defined by the Base MILS Platform Protection Profile.

7.1 Security Functional Requirements

7.1.1 FMT_MTD.1/API Management of TSF Data–API

FMT_MTD.1.1/API: The TSF shall restrict the ability to [assignment: list of operations on API] the [**API**] to [**the trusted partitions**].

7.1.2 FMT_MTD.1/OBJ Management of TSF Data–Objects

FMT_MTD.1.1/OBJ: The TSF shall restrict the ability to [assignment: list of operations on objects] the [**objects**] to [**the trusted partitions**].

7.1.3 FMT_SMF.1 Specification of Management Functions

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

- API management
- object management]

7.1.4 FMT_SMR.1 Security Roles

FMT_SMR.1.1: The TSF shall maintain the roles: [

- [*see section 8.1 for guidance*]

FMT_SMR.1.2: The TSF shall be able to associate ~~users~~ partitions with roles.

Application Note: The TSF supports roles on partition granularity.

7.2 Security Requirements Rationale

	OT.API_PROTECTION
FMT_MTD.1/API	X
FMT_MTD.1/OBJ	X
FMT_SMF.1 (in Base PP)	X
FMT_SMR.1	X

Table 3: SFR Rationale

OT.API_PROTECTION

FMT_SMF.1 (in base PP) specifies the management functions for partitions, tasks, and threads. FMT_MTD.1/API and FMT_MTD.1/OBJ specify the rules that define when a partition is allowed to perform the management activity.

7.3 Security Functional Requirements Dependencies Analysis

The following dependencies are defined for the SFRs used in this PP Module:

SFR	Dependencies	Satisfied?
FMT_MTD.1/API	FMT_SMF.1 FMT_SMR.1	yes (base PP) yes
FMT_MTD.1/OBJ	FMT_SMF.1 FMT_SMR.1	yes (base PP) yes

Table 4: SFR Functional Requirements Dependencies Analysis

Chapter 8 Application Notes

This section gives some examples how one could use and instantiate SFRs for roles (Section 8.1), API management (Section 8.2), and object management (Section 8.3).

We do not make any statement about what is the default behaviour when nothing is explicitly configured by the integrator, whether it is “deny all” management operations or “allow all” management operations. The user is encouraged to state this in the rules that define when a partition is allowed to perform the management activity or in the operational user guidance.

8.1 Roles SFR example

Some systems may come with a dichotomy of “trusted” and “untrusted” partitions. Other systems e. g. could support some partition hierarchy and restricting the functions to partitions that are hierarchically higher. Yet another type of systems could support some capability matrix that gives each partition a fine-grained set of capabilities.

Especially in capability-based TOEs some care has to be taken that the specifications do not contradict each other, e.g. a partition that can shut down the entire TOE, can implicitly also stop the threads of other partitions, even if this is not explicitly allowed. In TOEs with such capability-based models the consistency of the capability specification either could be explained in the operational user guidance, e.g. stating that capability to shut down other partitions implies the capability to stop threads in other partitions, at least where it is not obvious, or it could be argued for in the security architecture. Alternatively, a capability-based TOE could state when a partition is trusted and then use the “trusted” / “untrusted” dichotomy.

Some SK implementations allow trusted partitions. These may allow their host applications to partially or fully circumvent the system security policy (SSP), e.g. to change scheduling or reassign memory. This kind of set-up can be useful to implement custom monitoring and control functionality. The SK shall still control the untrusted partitions, and also control those parts of the SSP that the trusted partitions cannot bypass.

For instance, if there are only “trusted” and “untrusted” partitions, a possible instantiation of FMT_SMR.1 would then be

FMT_SMR.1.1: The TSF shall maintain the roles: [

- trusted partition
- untrusted partition

]

8.2 API Management SFR example

If we assume that there are only “trusted” and “untrusted” partitions, and there is an API for trusted partitions, the FMT_MTD.1 could look like the following:

FMT_MTD.1.1/API: The TSF shall restrict the ability to [invoke] the [Trusted Partition API] to [trusted partitions].

Application Note: The complete definition of the Trusted Partition API is given in the TOE User Manuals.

8.3 Object Management SFR examples

Partitions are a concept unique to SKs. However, an SK may optionally support additional interfaces mainly borrowed from general-purpose operating systems. In the following we give examples for objects: threads and tasks.

Thread: A thread is conventionally the smallest schedulable entity. That is, for a thread an OS maintains a CPU register context (i.e. the state of all CPU registers, often a one- or two-digit number), that can be stored away when the thread is scheduled away and be restored when the thread is scheduled in again. User programmable activities such as application code must be done in a thread. Some SK also might put exception handlers into threads. Also, for the virtualization of guest OS (virtual CPU), some SKs can opt to use one or many threads.

Address space/task: A virtual address space is an abstraction of physical memory, provided by a memory management unit (MMU). For SKs that are based on a memory protection unit (MPU), the concept of address spaces does not exist, unless similar concepts are provided by measures such as address rewriting at the compiler/linker level. In the following we describe the MMU case: The MMU manages chunks of physical memory, usually called pages. At the time of writing, on many systems, the size of a page is 4096 bytes. The MMU assigns on-the-fly pages to "virtual" addresses when access to any memory address is requested by application code. Address spaces make programming and deployment easier, as it is no longer needed to compile the applications into different memory regions. In the SK and embedded systems domain, also often the synonym task is used.

Application and its relation to tasks/threads: An application, in an MMU-based system is typically associated to at least one task and (where the thread abstraction is used) at least one thread. The typical use-case is that when the application is running, the SK uses the assigned quotas of threads and tasks to invoke a static configuration, that is when the application invokes a service call to start a new thread or new task it is checked that the assigned quotas are not exceeded. While in principle, other designs are technically possible, an often-encountered hierarchy is to:

- assign applications to a partition
- assign tasks to an application
- assign threads to a task

For example, in an SK that does not support tasks, but that supports threads, applications and partitions, the SK's design hierarchy would skip the task step, that is assign applications to a partition and assign threads to an application.

In an SK that does neither support tasks nor threads (e.g. real-time operating system – RTOS), applications are simply assigned to a partition, and that's all.

Of course, even in systems built on SKs that have all these abstractions (i.e. partitions, applications, tasks, and threads), a system integrator may opt not to make real use of them for a given deployment, as illustrated by Figure 1.

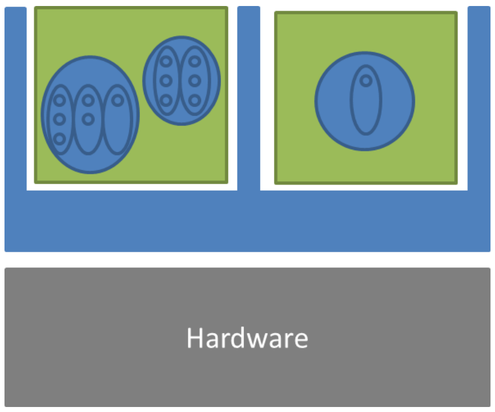

Figure 1: threads, tasks, applications, partitions

Here the left side of Figure 1 shows a partition with two applications, containing 3 and 2 tasks. The individual tasks contain 3, 2, 1, 3, and 3 threads, respectively. On the right hand side of Figure 1, there is a partition with just one application, which has just one thread that contains just one task.

Threads and tasks could refine AS.OBJ in the following way.

Tasks (AS.TASK)	An application always has at least one task. Tasks are used to structure the assigned memory into address spaces. This asset consists of this structure.
Threads (AS.THR)	An application always has at least one thread. The asset consists of all threads that can be created in a partition.

For simplicity, we will use in the examples below the dichotomy of “trusted” and “untrusted” partitions given in Section 8.1.

An assignment for FMT_MTD.1/OBJ for tasks could look like this:

FMT_MTD.1.1/TASK: The TSF shall restrict the ability to [assignment: start/stop/modify/read data from] the [**tasks**] to [**the owning partition or the trusted partitions**].

An assignment for FMT_MTD.1/OBJ for threads could look like this:

FMT_MTD.1.1/THREAD: The TSF shall restrict the ability to [assignment: start/stop/modify/read data from] the [**threads**] to [**the owning partition or the trusted partitions**].

Application note: The term “own” (for a task or a thread) is used in the following way: A partition owns a task if the task is assigned to it by the integrator in the configuration. A partition owns a thread if the thread is created by one of its applications.

Chapter 9 List of Abbreviations

Abbreviation	Translation
API	Application Programming Interface
MILS	Multiple Independent Levels of Safety / Security
PP	Protection Profile
SFR	Security Functional Requirement
SK	Separation Kernel
SSP	System Security Policy
TOE	Target of Evaluation
TSF	TOE Security Functionality

Chapter 10 Bibliography

[1] Common Criteria for Information Technology Security Evaluation. Part 1: Introduction and general model. Version 3.1, Revision 5. April 2017. CCMB-2017-04-001

[2] Common Criteria for Information Technology Security Evaluation. Part 2: Security Functional Components. Version 3.1, Revision 5. April 2017. CCMB-2017-04-002

[3] Common Criteria for Information Technology Security Evaluation. Part 3: Security Assurance Components. Version 3.1, Revision 5. April 2017. CCMB-2017-04-003

[4] Common Criteria for Information Technology Security Evaluation. Evaluation Methodology. Version 3.1, Revision 5. April 2017. CCMB-2017-04-004