SmartChip Shuniu

OS Kernel

Security Target Lite

Date: 2024/11/25

Change History

Version	Date	Author	Comment
1.0	25/11/2024	SmartChip	Initial official release

Table of contents

1 ST INTRODUCTION

1.1 ST REFERENCE

Title: SmartChip Shuniu OS Kernel - Security Target Lite

Version: 1.0

Author: SmartChip

Date of publication: 2024/11/25

1.2 TOE REFERENCE

TOE Name: Kernel layer of the Shuniu 4.0-Lite Microkernel

TOE Developer: Beijing SmartChip Microelectronics Technology Co. Ltd.

TOE Version: 2.0.12

1.3 TOE OVERVIEW

1.3.1 INTRODUCTION

The target of evaluation (TOE) is the kernel layer of a microkernel named the “Shuniu 4.0-Lite Microkernel”, henceforth abbreviated to the “Shuniu OS”, which is an embedded operating system (OS) designed to run on M4 Cortex processors with MPU support.

The TOE performs the kernel functionalities of the OS including address space management, thread management and scheduling, exception and interrupt handling, inter-thread communication, access control, and other basic microkernel services which are defined later in section 1.4.1.1.

As the TOE implements only a minimal set of services, it is considered lightweight, which is a required characteristic to run strong real-time applications in embedded systems.

For clarification, in this document microkernel is a trade name for the entire operating system, which includes the kernel layer (TOE) and the user layer (non-TOE). Therefore, when the term "microkernel" is used, it refers to the entire OS, including both layers.

The user layer of the microkernel includes several components as shown below in Figure 1:

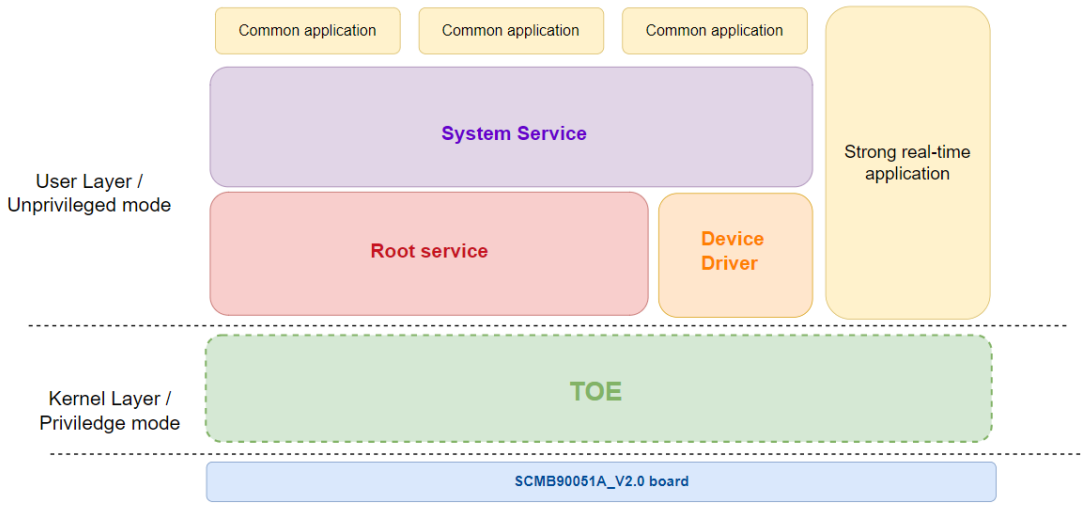


Figure 1 Microkernel's architecture

Each component within the user layer is explained in detail as follows:

- **Applications:** which encompasses common applications and strong real-time applications. The key difference between the two is that common applications do not invoke kernel services directly through system calls, unlike strong real-time applications which operate by directly requesting functionality from the kernel through invoking system calls to the TOE. Instead, common applications use the interfaces to the root service within the user layer, the root service then performs the requested operations on behalf of the requesting common application by making the appropriate system calls to the kernel (TOE).

- **System service:** this service implements the security and communication protocol, the database, the shell, and other functionalities to be used by the applications.

- **Root service:** provides several services, but the most relevant ones are POSIX API and user layer memory management. Common applications requests functionality from the kernel through this component, which acts as an intermediary by performing the necessary system calls to the kernel (TOE) on behalf of the requesting common application.

- **Device driver:** implements the driver interface and driver framework to be able to connect other devices to the microkernel.

These components are located in the user layer (unprivileged mode), as such they are not part of the scope of the TOE. The TOE itself is the kernel layer (privileged mode) of the overall microkernel, also known as the Shuniu OS, designed to run on ARM platforms that utilise Cortex M4 processors with MPU support. The final product of use for the Shuniu OS is a terminal device, containing the SCMB90051A_V2.0 board in which the TOE is run, running both common and strong real-time applications created by developers; an example of a typical scenario of its usage would be for devices controlling switches in power systems, which are also applied to real-time devices.

Since the scoping of the TOE does not include the core part of the Shuniu OS, as the evaluation focuses on the functionality provided by the kernel layer (privileged mode), the assurance provided is that the kernel layer protects its internal data structures, enforces access control on microkernel services, and additionally ensures that offending threads are blocked or killed.

1.3.2 TOE TYPE

The TOE is the kernel layer of a microkernel (whole OS), that contains the minimum number of functionalities to implement the core of an operating system. The most important functionality is the fine-grained management for threads and objects within the kernel, which it is intended to be used for the Internet of Things (IoT) and industrial control.

1.3.3 TOE USAGE & MAJOR SECURITY FEATURES

To provide fine-grained management for threads and resources, the functionalities of the TOE are divided into eight modules, that are described in further detail throughout section 1.4.1.1:

- System Invoking / System call
- Access Control
- Inter-thread communication
- Interrupt Management
- Thread management and scheduling
- Hardware abstraction layer (HAL)
- Space Isolation and Address Space
- System initialization

Access control is a major security feature of the TOE, it is implemented as a capability-based model, which checks the permissions of a subject to perform actions on an object; the users are threads, the subject is the kernel, whereas objects refer to the kernel objects. The access from the thread to the objects is controlled by capabilities, which are unforgeable tokens that represents authority; a list of capabilities is maintained for each thread, indicating the objects to which they have access.

The intended end usage of the TOE is a terminal device with real-time requirements, running the Shuniu OS; as such it is not meant to use video, multimedia, or multi-applications and therefore cannot to be integrated into devices for the intent of using these purposes (e.g. a mobile phone or in car multimedia). Due to the intended usage, it is therefore assumed that physical access to the TOE is restricted after completion of the respective duties of the trusted personnel, this being the system integrator and the system administrator.

In consequence to this, physical attacks to the TOE, or to the system in which it will be installed, are not considered to be within the scope of the evaluation.

1.3.4 NON-TOE HARDWARE/SOFTWARE/FIRMWARE

The following section details the Hardware, software and firmware components that are outside of the scope of the TOE.

1.3.4.1 NON-TOE HARDWARE

An SCMB90051A_V2.0 board is the only supported hardware platform. It provides the (non-TOE) communication interfaces and contains a SoC, which contains an ARM Cortex M4 and Memory Protection Unit (MPU); which enforces access control to the physical memory to establish isolation between different memory areas.

1.3.4.2 NON-TOE SOFTWARE

The TOE is distributed as part of a full Operating System, known as the Shuniu 4.0-Lite Microkernel. As previously shown in Figure 1 and discussed in section 1.3.1, there are components of the microkernel which reside in the user layer, not the kernel layer which is the TOE itself. All components within the user layer are therefore not part of the TOE, though it should be noted that these components do function together with the kernel in the overall operational environment. Additionally this includes the management of the communication interfaces (non-TOE hardware) as well, as this is also provided by the user layer.

1.4 TOE DESCRIPTION

1.4.1 INTRODUCTION

1.4.1.1 A GENERIC VIEW OF THE MICROKERNEL ARCHITECTURE

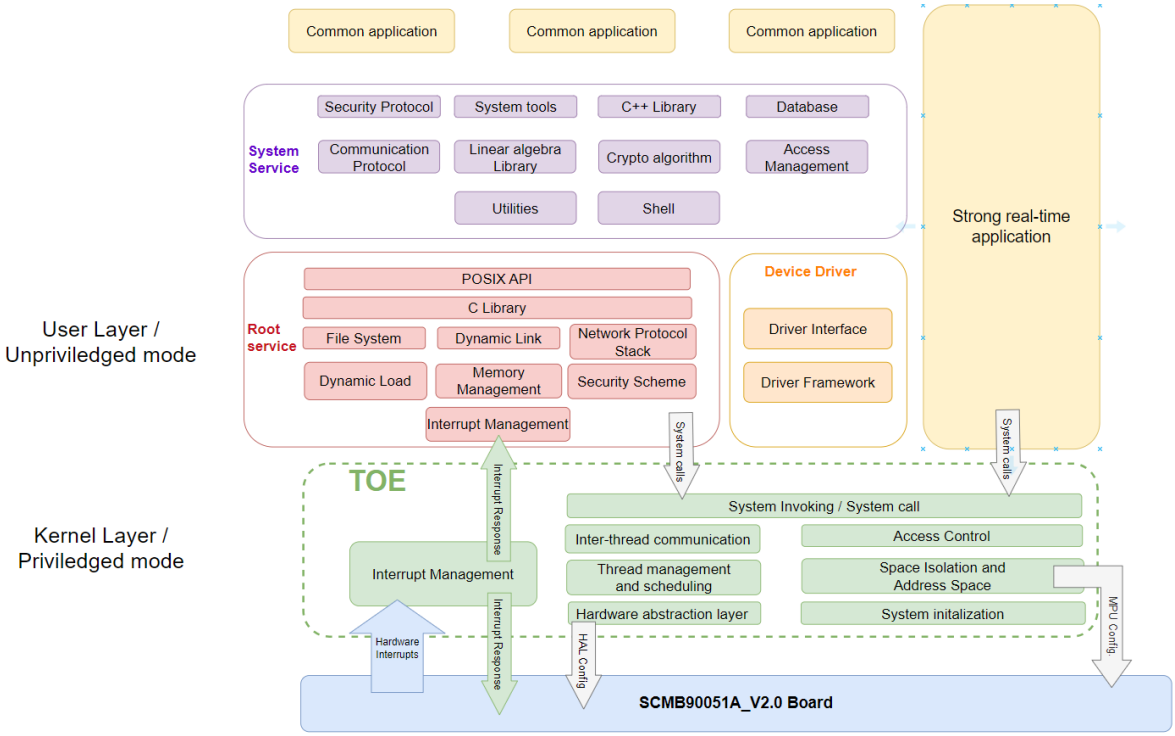


Figure 2 Detailed microkernel architecture

As shown in Figure 2, outside of the TOE, the components for the system services, root services and device drivers function as entities within the user layer. Also shown in Figure 2, are the functionalities of the kernel layer (TOE) which are spread throughout eight modules.

The functionalities of each module are described in detail throughout the following subsections.

1.4.1.1.1 SYSTEM INITIALIZATION

The system initialization module, as its name indicates, initializes the system including the kernel, CPU, root server and other components of the TOE (kernel layer). The TOE is stored in the internal flash memory of the platform and boots from it when the device is powered on, after which it then passes control over to the system initialization module to perform the actual initialization of the system. During initialization, this module will also interact with other modules to call upon a function when required; for example, it will call upon a function of the hardware abstraction layer to initialise the system clock.

1.4.1.1.2 HARDWARE ABSTRACTION LAYER

The hardware abstraction layer is the module responsible for initializing the system clock, the main frequency, and the configuration of other bus blocks.

1.4.1.1.3 THREAD MANAGEMENT AND SCHEDULING

The TOE provides a combined set of mechanisms to implement thread life cycle management and time slice management. This mechanism consists of creating threads through the root service, cloning threads, and managing their priority and state.

This module also supports the creation of IDLE thread, and is responsible for initializing the scheduler during the creation process. In addition, it also supports thread cloning as well as thread attribute configuration.

1.4.1.1.4 INTERRUPT MANAGEMENT

The TOE handles both external and internal exceptions. External exceptions refer to hardware-triggered external interrupts, such as MPU exceptions. Whereas, internal exception refers to undefined instructions or data/instruction access exceptions, SYSTICK exceptions, or software exceptions generated in the user layer.

1.4.1.1.5 INTER-THREAD COMMUNICATION

The TOE provides communication between threads through event notifications. The communication can be either by notifications or endpoints. Notifications are to perform inter-thread synchronisation, whereas endpoints are for transmitting information between threads.

1.4.1.1.6 SYSTEM INVOKING/SYSTEM CALL

This module supports calling specific kernel interface functions from the applications of the user layer, hence the threads cannot directly access the memory or objects in the kernel layer, except through system call invocation. These system calls, referred to as syscalls throughout this document, are directly invoked and processed by the kernel layer to perform operations on kernel objects when they are invoked. Invocation can occur directly when using strong real-time applications, or indirectly by common applications in the user layer (through use of the root service as an intermediary).

There are two types of syscalls: normal and capability-based. In the case of normal syscalls, no access control is exercised to invoke this type. Whereas as the name suggests, capability-based syscalls first check whether the subject actually has the permission to invoke them. Capability-based system calls are meant to perform operations on a kernel object, and in order to be able to perform those operations, the calling thread needs to have capabilities on that object either by having them in its private_cspace or in the initial_cspace (described in further detail in section 1.4.2.1.2).

1.4.1.1.7 ACCESS CONTROL

The TOE performs access control between subjects and objects by using a capability-based model, in which permissions on a kernel object mean being able to invoke capability-based system calls on those objects. In this model, there is no segregation of permissions, all operations for a matched capability are allowed. This process is described in further detail in section 1.4.2.1.2.

In general, when a system call is invoked, two separate processes can occur depending on the type of call. The two types of system call available are either normal system calls, or capability-based system calls. Figure 3 below shows the high-level flow of the aforementioned types of system calls:

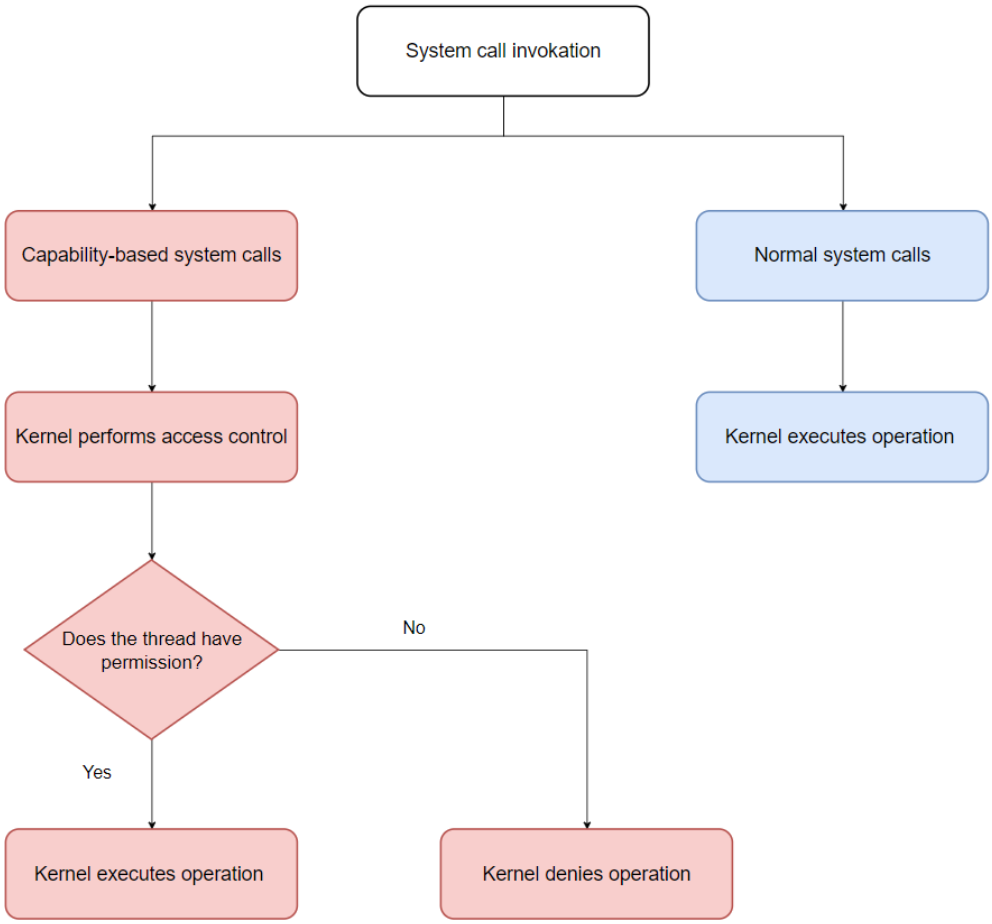


Figure 3 Process flow for system calls

As can be seen, for normal system calls no access control is performed, thus the kernel automatically performs the invoked operation. Whereas, capability-based system calls must have the correct permission else they are denied.

1.4.1.1.8 SPACE ISOLATION AND ADDRESS SPACE

The internal memory of the SoC is divided into address spaces (bin, bss, data, etc), inside those address spaces the TOE performs kernel-specific memory management to organize free blocks.

In addition, the TOE uses the MPU (non-TOE hardware) to set up address spaces (composed of one or more regions) for the user threads created by the kernel, thus creating address space isolation.

1.4.2 TOE LOGICAL SCOPE

The TOE, which is a binary that performs the kernel functions of the Operating System, provides the following security services with the modules presented in the previous section.

1.4.2.1.1 USER IDENTIFICATION

The TOE identifies all threads before allowing them to perform any TSF-mediated actions by associating them with the security attributes ID, private_cspace, and initial_cspace. Through this method of identification, the threads can be associated as either a common application or a strong real-time application; which in turn also checks if normal or capability-based system calls have been used, thus subsequently, if access control checks are required or not.

1.4.2.1.2 CAPABILITY-BASED ACCESS CONTROL

Access control is performed when a capability-based system call is invoked, this is to ensure that the control is only granted to users which are allowed to invoke the specific operation of the capability-based system call they are attempting to invoke. In this process, the thread attempting to invoke the system call passes, as a parameter, a pointer to the kernel object, which would also happen to be the same as in the cslot. These concepts are shown in below in Figure 4; a cnode is a list of the associated capabilities available to that thread, whereas the cslot is a pair of values stored in the cnode, it consists of the pointer to the kernel object and the type of kernel object:

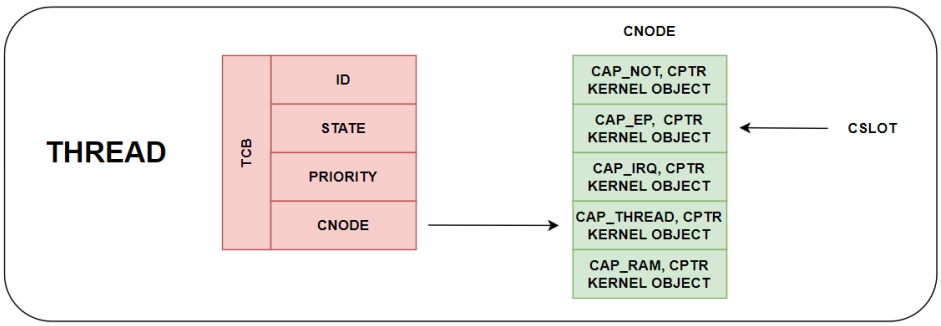


Figure 4 cnode information

Each thread has its own cnode (private_cspace) and has access to the private_cspace of the Root Service thread, which is referred as initial_cspace throughout the document.

Threads cannot create or modify their capabilities, instead they must invoke system calls to let the kernel do these operations. The access control mechanism works by performing a comparison between the pointer passed as argument, and the one stored in the cslot.

It will check if the comparison is made in either private or initial cspace:

- **Private:** used when the function to that kernel object is private, so only available to that thread.
- **Initial:** used when multiple threads require access to the function of that kernel object, so it is considered as public.

The TOE will first check the private_cspace of the calling thread to see if there is a match; if it exhausts all clsots within private_cspace, it will then move on and check for this match inside initial_cspace. As soon as a match is found, the search stops, this means if the match is found in private it will not move onto to checking for a match in initial_cspace.

Once the TOE has determined the cspace, it will then finally check that the type of object defined matches the request function. To put the process into context using an example, if the operation requested through a system call was an IPC communication to clone a thread, then that operation would be both valid and accepted if the target kernel object passed as parameter is a thread-type object, but it would be invalid and denied for other types of kernel objects.

1.4.2.1.3 MEMORY MANAGEMENT

The TOE configures and operates the MPU (non-TOE hardware) to set up address spaces (composed of one or more regions) for the threads created by the kernel, effectively establishing separation between the kernel and user layer memory regions. This means that threads executing in privileged mode can access the memory space in both layers, whereas the remaining threads executing in unprivileged mode can only access the memory space within the user layer.

1.4.2.1.4 THREAD MANAGEMENT

The Thread management and scheduling module provide a combined set of mechanisms to manage threads.

- **Thread management and scheduling initialization:** supports the creation of IDLE and root service threads, initializing the scheduler during the creation process.
- **Thread configuration:** support cloning threads and configure threads' properties.
- **Thread Lifecycle Management:** support thread state switching and thread queue management.
- **Thread scheduling:** priority-based preemptible scheduling strategies is implemented for threads with different priorities, whereas for threads with the same priority time-slice round-robin is implemented. This strategy is used for assigning CPU time to the threads.

1.4.2.1.5 INTERRUPT MANAGEMENT

The TOE handles both external and internal exceptions. External exceptions refer to hardware-triggered external interrupts, meanwhile internal exceptions refer to undefined instructions or data/instruction access exceptions, SYSTICK exceptions, or software exceptions all of them generated in the user layer. Therefore, this module provides fault tolerance, keeping a secure state in case of any type of interruption or hardware failure which could lead on unexpected behaviour of the TOE, by killing the thread or halting the system if necessary.

1.4.3 TOE PHYSICAL SCOPE

The TOE includes the following components:

Name	Type	Version	Distribution format	Description	Delivery
Kernel.bin **Hash Value:** 3eee585c67118ccad8df9b8931fda2852e866b93269c7d423b07f91bcdb389e3	TOE	2.0.12	Binary file	TOE software image. **Note:** This image shall be integrated during the secure acceptance (described in the TOE guidance) with an image containing the non-TOE parts of the full operating system. That full OS image will be installed on the non-TOE hardware platform.	Electronic delivery
SmartChip Shuniu OS Kernel - Operational Guidance. **Hash Value:** 0eefd13b39872e5190debff0cf89fd9b46f848b17eb	Guidance	1.9	PDF Document	Operational usage guidance.	Electronic delivery

f4cce86f2911be7d41a5e					
SmartChip Shuniu OS Kernel - Preparative Guidance. **Hash Value:** 003dd761a6e9f6a93c1575f60e40ca8f0526ceae63f7c2293158b70795d6e89c	Guidance	1.10	PDF Document	Installation process guidance.	Electronic delivery

Table 1 Physical Scope

2 CONFORMANCE CLAIMS

This Security Target and the TOE described are in accordance with the requirements of Common Criteria 3.1R5.

This Security Target claims conformance with the following parts of Common Criteria:

- Conformance with [CC31R5P1].
- Conformance with [CC31R5P2] extended.
- Conformance with [CC31R5P3].

The methodology to be used for the evaluation is described in the “Common Evaluation Methodology” of the Common Criteria standard of April 2017, version 3.1 revision 5 with an evaluation assurance level of EAL5 + ALC_FLR.1.

This Security Target does not claim conformance with any protection profile.

3 SECURITY PROBLEM DEFINITION

This section describes the security aspects of the operational environment and its expected use in said environment. It includes the declaration of the TOE operational environment that identifies and describes:

- The alleged known threats that will be countered by the TOE
- The organizational security policies that the TOE and the TOE environment have to adhere to.
- The TOE usage assumptions in the suggested operational environment.

We will begin defining Assets and Threat Agents.

3.1 ASSETS

This section identifies the assets that require protection by the TOE.

KERNEL_DATA: Data located in memory space of the kernel and used by the TOE, such as Kernel objects. This also includes the compiled binary code implementing the logic of the kernel.

REAL_TIME_REQUIREMENT: Thread scheduling of the microkernel implements priority based (real-time) execution for running threads. There is a limit on the maximum amount of time to which the kernel will respond to a request from a thread for execution, as part of the real-time requirements.

3.2 THREAT AGENTS

This section identifies the threat agents. Hardware attacks have been explicitly not considered or included as a possible security problem of the TOE; the TOE itself is a microkernel, as piece of software, as such the only threat agent considered to be within scope of the TOE is **SOFTWARE ATTACKER**, as the only attacks considered involve the non-TOE applications running on the user layer of the Shuniu OS.

SOFTWARE ATTACKER: Malicious actor performing software attacks on the TOE using a user-space application or service, either by abusing a legitimate application or by creating a malicious one.

3.3 THREATS TO SECURITY

This section identifies the threats to assets that require protection by the TOE. The threats are defined in terms of assets concerned, attackers and the adverse action that materializes the threat.

It is assumed that the Shuniu OS will only be deployed in the previously aforementioned intended method, and thus will be protected from unauthorised physical access. Therefore, physical attacks to the TOE, or to the system in which it will be integrated, are not considered to be within the scope of the TOE.

T.UNAUTHORIZED_ACCESS: A **SOFTWARE ATTACKER** is able to read or modify **KERNEL_DATA** without authorization.

T.QUEUE_SKIPPING: A **SOFTWARE ATTACKER** attempts to manipulate the KERNEL scheduler in an attempt to bypass the **REAL_TIME_REQUIREMENT** of the KERNEL.

T.MALFUNCTION: A **SOFTWARE ATTACKER** attempts to improperly access **KERNEL_DATA** by invoking interruptions which could provoke TOE malfunction, through use of one of the following methods:

- Attempting to request an unrecognised/illegal operation
- Attempting to access resources without the required permissions
- Intentionally causing an anomalous scenario within the TOE

3.4 ASSUMPTIONS

The assumptions when using the TOE are the following:

A.TRUSTWORTHY_PERSONNEL: The personnel installing the TOE onto the SCMB90051A_V2.0 board (System Integrator), as well as administering to the final device in which the board is to be integrated (System Administrator), are trustworthy. In particular, the roles of system integrator and system administrator must adhere to the following properties;

System Integrator:

- Responsible for installing the TOE into the board.
- Responsible for integrating the board into the final device.
- Loads applications into the TOE before release of the final device.

System Administrator:

- Purchases the final device in which the TOE is integrated.
- Loads applications into the TOE before deployment of the final device.

A.SECURE_INTEGRATION_AND_ADMINISTRATION: It is assumed that security procedures are used during the integration (System Integrator) of the TOE onto the SCMB90051A_V2.0 board and during the administration of the final device (System Administrator) up to the delivery to the end consumer to protect its integrity and confidentiality. System Integrator and System Administrator entities are expected to provide a controlled environment in which the relevant personnel interacting with the TOE do not pose a security threat to its integrity or confidentiality.

A.TRUSTED_PLATFORM: The TOE is designed to run on the trusted SCMB90051A_V2.0 board, subsequently all peripherals on this board are hence trusted.

A.PLATFORM_PROTECTION: The trusted SCMB90051A_V2.0 board, on which the TOE runs, is considered to be adequately protected from physical attacks, and does not represent a physical attack vector due to in place security measures within the operational environment that prevent both unauthorised physical and logical access to the board as well as access to the final device in which the trusted SCMB90051A_V2.0 board is integrated. This ensures that any form of physical attack to the TOE, the cortex M4 processor, or the final device in which it is integrated are prevented. In particular, the roles of system integrator and system administrator must adhere to the following properties;

System Integrator:

- Responsible for the physical protection of the board (including the TOE) during installation.

System Administrator:

- Responsible for physical protection of the device (containing the board which contains the TOE).

Application Note:

Consequently, physical attacks are not considered to be within scope of the SPD, or part of this assumption. As such, any form of physical attack to either the TOE or the system in which the TOE in integrated into, are not considered as they are not within the scope of the SPD.

3.5 ORGANISATIONAL SECURITY POLICIES

The organizational Security policy is defined as follows.

OSP.INTERRUPTS: The TOE shall deal with hardware interruptions generated because of hardware faults which could provoke TOE malfunction.

4 SECURITY OBJECTIVES

The security objectives are high level declarations, concise and abstract of the solution to the problem exposed in the former section, which counteracts the threats and fulfils the security policies and the assumptions. These consist of:

- the security objectives for the operational environment.
- the security objectives for the TOE

4.1 SECURITY OBJECTIVES FOR THE TOE

The security objectives for the TOE must determine the responsibility of the TOE in countering the threats and in enforcing the OSPs. Each objective must be traced back to aspects of identified threats to be countered by the TOE and to aspects of OSPs to be met by the TOE.

OT.PRIORITY: The TOE shall define a priority-based scheduling for the threads. The priority shall determine the given time for each thread to access the resources (a.k.a CPU time). As a real-time kernel, task scheduling will prioritise threads with the highest priority first and will guarantee a time slice for execution.

OT.SAFE_SECURE_STATE: The TOE shall be able to intercept interruptions generated outside the TOE, from hardware or from software failures and preserve a secure state by cancelling the problematic threads or halting the system.

OT.ACCESS_CONTROL: The TOE shall enforce access controls in order to grant permissions over assets only to authorized users. Specifically, only threads possessing capabilities on a kernel object shall be able to perform operations (specific per object type) over that kernel object. The TOE shall configure and operate non-TOE memory protection hardware to enforce memory isolation between the kernel and user layers.

4.2 SECURITY OBJECTIVES FOR THE OPERATIONAL ENVIRONMENT

The security objectives for the Operational Environment determine the responsibility of the environment in countering the threats, enforcing the OSPs and upholding the assumptions. Each objective must be traced back to aspects of identified threats to be countered by the environment, to aspects of OSPs to be enforced by the environment and to assumptions to be uphold by the environment.

OE.TRUSTWORTHY_PERSONNEL: The System Integrator and the System Administrator shall both be trustworthy when performing any action during the normal course of their defined duties.

In particular, the System Integrator shall adhere to the following:

- To perform the installation process of the binary image onto the trusted board (SCMB90051A_V2.0) according to the provided TOE guidance (and, if necessary, according to the hardware manuals).
- To perform the installation process of any applications, created by Software Developers, onto the TOE before the release of the final device.
- To integrate the board, containing the TOE, into the final device to be purchased.

In addition, the System Administrator shall adhere to the following:

- To perform the installation process of applications, created by Software Developers, into the TOE after purchase of the final device.

OE.SECURE_INTEGRATION_AND_ADMINISTRATION: The System Integrator and the System Administrator entities shall conduct their activities in a controlled environment which provide commensurate security to the integrity and confidentiality of the TOE. Both entities shall ensure that the relevant staff is properly trained, can be trusted upon and are not wilfully negligent while carrying out their activities under the controlled environment.

OE.TRUSTED_PLATFORM: The TOE shall run on the trusted a SCMB90051A_V2.0 board. The device driver, root service and system service, shall be loaded as part of TOE installation.

OE.PLATFORM_PROTECTION: For the physical protection of the TOE, the roles of system integrator and system administrator must adhere to the following properties;

System Integrator:

- To ensure the physical protection of the board (and TOE) prior to release of the final device.

System Administrator:

- To ensure the physical protection of the final device in which the TOE is integrated.

4.3 SECURITY OBJECTIVES RATIONALE

The following table (*Table 2*) provides a mapping of security objectives tracing each security objective for the TOE back to threats countered by that security objective and OSPs enforced by that security objective, and each security objective for the operational environment back to threats countered by that security objective, OSPs enforced by that security objective, and assumptions upheld by that security objective.

This illustrates that the security objectives counter all threats, the security objectives enforce all OSPs and the security objectives for the operational environment uphold all assumptions.

	OT.PRIORITY	OT.SAFE_SECURE_STATE	OT.ACCESS_CONTROL	OE.TRUSTWORTHY_PERSONNEL	OE.TRUSTED_PLATFORM	OE.PLATFORM_PROTECTION	OE.SECURE_INTEGRATION_AND_ADMINISTRATION
T.UNAUTHORIZED_ACCESS			X				
T.QUEUE SKIPPING	X						
T.MALFUNCTION		X					
OSP.INTERRUPTS		X					
A.TRUSTWORTHY_PERSONNEL				X			
A.TRUSTED_PLATFORM					X		
A.PLATFORM_PROTECTION						X	
A.SECURE_INTEGRATION_AND_ADMINISTRATION							X

Table 2 Security Objectives vs Security Problem Definition

4.3.1 THREATS

T.UNAUTHORIZED_ACCESS: OT.ACCESS CONTROL prevents unauthorized access to kernel data and kernel code during normal operation of the TOE.

T.QUEUE SKIPPING: OT.PRIORITY ensures that threads using the TOE are executed on a priority-based queue.

T.MALFUNCTION: OT.SAFE SECURE STATE protects the TOE against the interruptions which come from unexpected thread behaviour because of defective programming, or incorrect use of the APIs or environment malfunction which could lead in an unsecure state of the TOE that could put assets in danger. All interrupts will result in the cancelation of the threads or the halting of the system to prevent malfunction of the TOE.

4.3.1.1 THREAT MAPPING TO SECURITY OBJECTIVES

The following table maps the threats of the security problem established to the security objectives of the TOE and the security objectives of the operational environment.

Threats	Security Objectives
T.UNAUTHORIZED_ACCESS	OT.ACCESS_CONTROL
T.QUEUE_SKIPPING	OT.PRIORITY
T.MALFUNCTION	OT.SAFE_SECURE_STATE

Table 3 Threats vs Security Objectives

4.3.2 ASSUMPTIONS

A.TRUSTWORTHY PERSONNEL: The assumption A.TRUSTWORTHY PERSONNEL is directly upheld by **OE.TRUSTWORTHY PERSONNEL**.

A.TRUSTED_PLATFORM: The assumption A.PLATFORM is directly upheld by **OE.TRUSTED_PLATFORM**.

A.PLATFORM_PROTECTION: The assumption A.TRUSTED_PLATFORM is directly upheld by **OE.PLATFORM_PROTECTION**.

A.SECURE_INTEGRATION_AND_ADMINISTRATION: The assumption A.SECURE_INTEGRATION_AND_ADMINISTRATION is directly upheld by **OE.SECURE_INTEGRATION_AND_ADMINISTRATION**.

4.3.2.1 ASSUMPTION MAPPING TO SECURITY OBJECTIVES

The following table maps the assumptions of the problem established to the security objectives of the TOE and the security objectives of the operational environment.

Assumptions	Security Objectives
A.TRUSTWORTHY_PERSONNEL	OE.TRUSTWORTHY_PERSONNEL
A.TRUSTED_PLATFORM	OE.TRUSTED_PLATFORM
A.PLATFORM_PROTECTION	OE.PLATFORM_PROTECTION

Table 4 Assumptions vs Security Objectives for the Operational Environment

4.3.3 ORGANISATIONAL SECURITY POLICY RATIONALE

OSP.INTERRUPTS: The objective **OT.SAFE_SECURE_STATE** directly enforces this OSP by ensuring the preservation of a secure state when a hardware failure occurs which could cause TOE malfunction.

The following table maps the organisational security policies of the problem established to the security objectives of the TOE and the security objectives of the operational environment.

OSPs	Security Objectives
OSP.INTERRUPTS	OT.SAFE_SECURE_STATE

Table 5 OSPs vs Security Objectives

5 EXTENDED COMPONENTS DEFINITION

5.1 CLASS FDP: USER DATA PROTECTION

This class contains families specifying requirements related to protecting user data. FDP is split into four groups of families (listed below) that address user data within a TOE, during import, export, and storage as well as security attributes directly related to user data.

The families in this class are organised into four groups:

- User data protection security function policies:
- Forms of user data protection:
- Off-line storage, import and export:
- Inter-TSF communication:

The following extended families use the definition and structure of the predefined class FDP.

5.1.1 DELEGATED MEMORY ISOLATION (FDP_DMI)

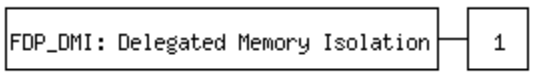
This SFR relies on the MPU, which is non-TOE hardware component.

Family behaviour

This family is used to describe isolation capabilities which the TOE offers that are based in the MPU.

The MPU establishes memory isolation between memory regions, based in an access control mechanism which allow or deny the access to between memory regions of the TOE.

Component levelling



FDP_DMI.1 Isolation of memory requires an external component to provide functionality which enforces isolation of memory when it is accessed by different entities.

Management: FDP_DMI.1

There are no management activities foreseen.

Audit: FDP_DMI.1

There are no auditable events foreseen.

FDP_DMI.1: Entity Controlled Data Isolation

Hierarchical to:

No other components.

Dependencies:

No dependencies.

FDP_DMI.1.1: The TSF shall configure and operate an *[selection: external, internal]* entity that enforces *[assignment: data structures]* between *[assignment: list of entities to be isolated]*, allowing *[assignment: allowed data flow]*, but restricting *[assignment: restricted data flow].*

6 SECURITY REQUIREMENTS

This section defines the Security functional requirements (SFRs) and the Security assurance requirements (SARs) that fulfill the TOE. Assignment, selection, iteration and refinement operations have been made, adhering to the following conventions:

- Assignments. They appear between square brackets. The word “assignment” is maintained and the resolution is presented in ***boldface, italic and blue color.***
- Selections. They appear between square brackets. The word “selection” is maintained and the resolution is presented in ***boldface, italic and blue color.***
- Iterations. It includes “/” and an “identifier” following requirement identifier that allows to distinguish the iterations of the requirement. Example: FCS_COP.1/XXX.
- Refinements: the text where the refinement has been done is shown ***bold, italic, and light red color.*** Where part of the content of a SFR component has been removed, the removed text is shown in ~~***bold, italic, light red color and crossed out.***~~

6.1 DEFINITIONS

The statement of the security functional requirements relies on the following characterization of the TOE in terms of subjects, objects, operations and their security attributes.

Users refer to entities outside the TOE and are defined as follows:

- **U.THREAD:** threads that are inside the user layer and invoke a system call to access resources or objects within the TOE.

Subjects refer to entities inside the TOE and are defined as follows:

- **S.KERNEL:** when threads reach the TOE (kernel layer) after invoking a system call, and are within the scheduler, this is when the kernel is involved as the subject.

Objects stand for kernel objects inside the TOE and are defined as follows:

- **OB.CNODE:** kernel object containing the permissions of what capabilities are allowed to be performed. This maps to the kernel object “*cnode*” in Table 6.
- **OB.NOTIFICATION:** kernel object that is used for event triggering between threads. This maps to the kernel object “*Notification”* in Table 6.
- **OB.ENDPOINT:** kernel object containing the information transmitted between two users, along with the state of the communication. This maps to the kernel object *“Endpoint”* in Table 6.

- **OB.IRQ_CONTROL:** kernel object to control interrupt request by associating or disassociating an event to an interrupt. This maps to the kernel object "*IRQ Control*" in Table 6.
- **OB.IRQ_HANDLER:** kernel object which will handle interrupts by forcing an event to wait, or respond to an interrupt. This maps to the kernel object *"IRQ Handler"* in Table 6.
- **OB.THREAD:** kernel object that performs operations over the thread itself, this includes creating a new thread, modifying a thread properties and changing the priority of an existing thread. This maps to the kernel object "*Thread*" in Table 6.
- **OB.RAM**: kernel object that is used to create and initialise memory space for a thread within the RAM. This maps to the kernel object "*RAM*" in Table 6.

Note: All of these objects have their corresponding operations defined as follows in Table 6:

Kernel Object	Operation	Description
cnode	1. CAP_THREAD_FLUSH_CNODE	1. Delete the thread capability space (cnode)
Notification	1. CAP_NTFN_OP_WAIT 2. CAP_NTFN_OP_SIGNAL 3. CAP_NTFN_OP_BDCAST 4. CAP_NTFN_OP_CANCEL	1. Wait for other thread to notify through the event 2. Notify a waiting thread through an event 3. Notify all waiting threads through the events 4. Invalidate event objects
Endpoint	1. CAP_EP_OP_SEND 2. CAP_EP_OP_RECV 3. CAP_EP_OP_CALL 4. CAP_EP_OP_REPLY 5. CAP_EP_OP_REPLY_WAIT 6. CAP_EP_CANCEL	1. Send data through endpoint objects 2. Receive data through endpoint objects 3. Initiate RPC calls through endpoint objects 4. Reply to RPC calls made through endpoint objects 5. Reply to the RPC call through the endpoint object and receive the response 6. Invalidate Endpoint objects
IRQ Control	1. CAP_IRQ_OP_GET 2. CAP_IRQ_OP_PUT	1. Associate the event object corresponding to the interrupt number with the IRQ HANDLER capability object 2. Disassociate the event object corresponding to the interrupt number from the IRQ HANDLER capability object
IRQ Handler	1. CAP_IRQ_OP_WAIT 2. CAP_IRQ_OP_ACK 3. CAP_IRQ_OP_ACK_WAIT	1. Wait for the interrupt associated with the IRQ HANDLER capability object (notified through the event object)

		2. Respond to interrupts associated with IRQ HANDLER capability objects 3. Wait for the interrupt associated with the IRQ HANDLER capability object and respond it
Thread	1. CAP_THREAD_CLONE 2. CAP_THREAD_CONFIG 3. CAP_THREAD_CHANGE_QUE 4. CAP_THREAD_FLUSH_CNODE 5. CAP_THREAD_DUMP 6. CAP_THREAD_SEM_V 7. CAP_THREAD_BLOCK 8. CAP_THREAD_SUSPEND 9. CAP_THREAD_CANCEL	1. Create a new thread 2. Configure the thread's properties 3. Modify the thread's priority (if it is ready, re-join the ready queue) 4. Delete the thread capability space (cnode) 5. Return information about the thread, such as stack address and ipc buffer address 6. Wake up a thread. 7. Block a specified thread. 8. Suspend or resume a specified thread. 9. Cancel a specified thread.
RAM	1. CAP_RAM_MKCAP 2. CAP_RAM_FREE	1. Creates a capability object in the thread's capability space and initialize it 2. Release the capability object in the thread's capability space

Table 6 Kernel Objects and Operations

The security attributes are defined as follows:

- **ATTR.PRIV_CSPACE.CNODE.CSLOT:** private permission of a thread to invoke an operation in a kernel object. This consists of a pair of values, the pointer to the kernel object and the type of the kernel object.
- **ATTR.INIT_CSPACE.CNODE.CSLOT:** public permission of a thread to invoke an operation in a kernel object. This consists of a pair of values, the pointer to the kernel object and the type of the kernel object. This permission is checked if the thread does not contain the required permission in the private_cspace.
- **ATTR.THREAD.ID:** ID of the thread.

The operations on kernel objects are defined as follows:

- **OP.INVOKE:** invoke system calls to operate with the kernel objects. All operations corresponding to each type of kernel objects are automatically authorized once permissions over the object are granted.

6.2 SECURITY FUNCTIONAL POLICIES

This Security Target defines the following Security Function Policies (SFPs):

6.2.1 CAPABILITY-BASED ACCESS CONTROL SFP (SFP.CAP):

1. **Purpose:** To control the access to kernel objects through invocation of capability-based system calls located within the system call module of the TOE.
2. **Users:** threads which are located within the user layer. Because of this, they are also referred as “user threads” (non-TOE).
3. **Subjects:** the kernel (TOE).
4. **Objects:** Kernel objects (the corresponding operations are defined in Table 6).
5. **Security attributes:** Permissions ascertained from private_cspace and initial_cspace.
6. **SFR instances:** FDP_ACC.1, FDP_ACF.1, FMT_MSA.3

6.3 SECURITY FUNCTIONAL REQUIREMENTS

This section defines the Security Functional Requirements (SFRs) the TOE has to enforce in order to fulfil the security objectives.

Application Note:

The security policy mentioned in the SFRs is the following:

SFP.CAP: Implementation of proper management of capability access.

6.3.1 FDP: USER DATA PROTECTION

6.3.1.1 FDP_ACC.1: SUBSET ACCESS CONTROL

FDP_ACC.1.1 The TSF shall enforce the ***[assignment: SFP.CAP]*** on ***[assignment:***

- ***Subjects: S.KERNEL***
- ***Objects: OB.NOTIFICATION, OB.ENDPOINT, OB.IRQ_CONTROL, OB.IRQ_HANDLER, OB.THREAD, OB.RAM, OB.CNODE***
- ***Operations: OP.INVOKE]***

6.3.1.2 FDP_ACF.1: SECURITY ATTRIBUTE BASED ACCESS CONTROL

FDP_ACF.1.1 The TSF shall enforce the ***[assignment: SFP.CAP]*** to objects based on the following: ***[assignment:***

- ***Subjects: S.KERNEL***
- ***Objects: OB.NOTIFICATION, OB.ENDPOINT, OB.IRQ_CONTROL, OB.IRQ_HANDLER, OB.THREAD, OB.RAM, OB.CNODE***
- ***Security attributes: ATTR.PRIV_CSPACE.CNODE.CSLOT, ATTR.INIT_CSPACE.CNODE.CSLOT]***.

FDP_ACF.1.2 The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed: ***[assignment:***

S.KERNEL will perform OP.INVOKE on OB.NOTIFICATION, OB.ENDPOINT, OB.IRQ_CONTROL, OB.IRQ_HANDLER, OB.THREAD, OB.RAM, OB.CNODE on behalf of U.THREAD if any of the below conditions stand:

a. ATTR.PRIV_CSPACE.CNODE.CSLOT matches the pointer and type of the kernel object, with respect to the kernel object that is target of the invocation,

b. ATTR.INIT_CSPACE.CNODE.CSLOT matches the pointer and type of the kernel object, with respect to the kernel object that is target of the invocation,

The corresponding operations on the object type are shown in Table 6 Kernel Objects and Operations.]

FDP_ACF.1.3 The TSF shall explicitly authorise access of subjects to objects based on the following additional rules: ***[assignment: None].***

FDP_ACF.1.4 The TSF shall explicitly deny access of subjects to objects based on the following additional rules: ***[assignment: None].***

Application note: Each thread retains its own private_cspace, the attribute THREAD.ID determines which private_cspace is checked in enforcement of the SFP.CAP.

6.3.1.3 FDP_DMI.1: DELEGATED MEMORY ISOLATION

FDP_DMI.1.1 The TSF shall configure and operate an ***[assignment: external]*** entity that enforces ***[assignment: memory isolation]*** between ***[assignment: memory in kernel space and in user space]***, allowing ***[assignment: threads in kernel space full access to kernel and user memory space]***, but restricting ***[assignment: threads in user space to access only memory in user space]***.

6.3.2 FIA: IDENTIFICATION AND AUTHENTICATION

6.3.2.1 FIA_ATD.1: USER ATTRIBUTE DEFINITION

FIA_ATD.1.1 The TSF shall maintain the following list of security attributes belonging to individual users: ***[assignment: ATTR.THREAD.ID, ATTR.PRIV_CSPACE.CNODE.CSLOT and ATTR.INIT_CSPACE.CNODE.CSLOT]***.

6.3.2.2 FIA_UID.2: USER IDENTIFICATION BEFORE ANY ACTION

FIA_UID.2.1 The TSF shall require each user to be successfully identified before allowing any other TSF-mediated actions on behalf of that user.

6.3.2.3 FIA_USB.1: USER-SUBJECT BINDING

FIA_USB.1.1 The TSF shall associate the following user security attributes with subjects acting on the behalf of that user: ***[assignment: ATTR.THREAD.ID, ATTR.PRIV_CSPACE.CNODE.CSLOT and ATTR.INIT_CSPACE.CNODE.CSLOT]***.

FIA_USB.1.2 The TSF shall enforce the following rules on the initial association of user security attributes with subjects acting on the behalf of users: ***[assignment: The ID assigned to each thread is unique and, when created, it gets assigned a private cnode to which other threads do not have any permission]***.

FIA_USB.1.3 The TSF shall enforce the following rules governing changes to the user security attributes associated with subjects acting on the behalf of users: ***[assignment:***

- ***The thread identifier ATTR.THREAD.ID cannot be modified.***
- ***The ATTR.PRIV_CSPACE.CNODE.CSLOT and ATTR.INIT_CSPACE.CNODE.CSLOT can only be modified by the thread-management and scheduling module of the kernel]***.

Application note: The ATTR.PRIV_CSPACE.CNODE.CSLOT and ATTR.INIT_CSPACE.CNODE.CSLOT can be modified by the kernel on behalf of U.THREAD, through either creation of an object or flushing of the cnode (see FDP_ACC.1 or FDP_ACF.1).

6.3.3 FMT: SECURITY MANAGEMENT

6.3.3.1 FMT_MSA.3: STATIC ATTRIBUTE INITALIZATION

FMT_MSA.3.1 The TSF shall enforce the ***[assignment: SFP.CAP]*** to provide ***[selection: restrictive]*** default values for security attributes that are used to enforce the SFP.

FMT_MSA.3.2 The TSF shall allow the ***[assignment: none]*** to specify alternative initial values to override the default values when an object or information is crated.

Application note: any operation that causes a modification of the OB.CNODE object is always performed through S.KERNEL, which is the only entity that has direct access to modify the object.

Application note: Restrictive default values refer to cnode, within a corresponding cspace (ATTR.PRIV_CSPACE.CNODE.CSLOT and ATTR.INIT_CSPACE.CNODE.CSLOT), that is empty of capabilities upon its initialization.

6.3.4 FPT: PROTECTION OF THE TSF

6.3.4.1 FPT_FLS.1: FAILURE WITH PRESERVATION OF SECURE STATE

FPT_FLS.1.1 The TSF shall preserve a secure state when the following types of failures occur: ***[assignment: HW interrupts caused by hardware failures, SW interrupts caused by software failures in user layer (OS and applications)]***.

6.3.5 FRU: RESOURCE UTILISATION

6.3.5.1 FRU_PRS.1: LIMITED PRIORITY OF SERVICE

FRU_PRS.1.1 The TSF shall assign a priority to each subject in the TSF.

FRU_PRS.1.2 The TSF shall ensure that each access to ***[assignment: CPU time]*** shall be mediated on the basis of the subject's assigned priority.

6.4 SECURITY ASSURANCE REQUIREMENTS

The development and the evaluation of the TOE shall be done in accordance to the following security assurance requirements: **EAL5 + ALC_FLR.1**.

The following table shows the assurance requirements by reference the individual components in [CC31R5P3]:

Assurance Class	Assurance Components
ASE: Security Target evaluation	ASE_CCL.1: Conformance claims ASE_ECD.1: Extended components definition ASE_INT.1: ST introduction ASE_TSS.1: TOE summary specification ASE_OBJ.2: Security objectives ASE_REQ.2: Derived security requirements ASE_SPD.1: Security problem definition
ALC: Life-cycle support	ALC_CMC.4: Production support, acceptance procedures and automation ALC_CMS.5: Development tools CM coverage ALC_DEL.1: Delivery procedures ALC_DVS.1: Identification of security measures ALC_LCD.1: Developer defined life-cycle model ALC_TAT.2: Compliance with implementation standards ALC_FLR.1: Basic flaw remediation
ADV: Development	ADV_ARC.1: Security architecture description ADV_IMP.1: Implementation representation of the TSF ADV_FSP.5: Complete semi-formal functional specification with additional error information ADV_INT.2: Well-structured internals ADV_TDS.4: Semiformal modular design

Assurance Class	Assurance Components
AGD: Guidance documents	AGD_OPE.1: Operational user guidance AGD_PRE.1: Preparative procedures
ATE: Tests	ATE_COV.2: Analysis of coverage ATE_DPT.3: Testing: modular design ATE_FUN.1: Functional testing ATE_IND.2: Independent testing - sample
AVA: Vulnerability assessment	AVA_VAN.4: Methodical vulnerability analysis

Table 7 Security Assurance Requirements

6.5 SECURITY REQUIREMENTS RATIONALE

6.5.1 NECESSITY AND SUFFICIENCY ANALYSIS

SFR / TOE Security Objective	OT.PRIORITY	OT.SAFE_SECURE_STATE	OT.ACCESS_CONTROL
FIA_ATD.1			X
FIA_USB.1			X
FPT_FLS.1		X	
FDP_ACC.1			X

SFR / TOE Security Objective	OT.PRIORITY	OT.SAFE_SECURE_STATE	OT.ACCESS_CONTROL
FDP_ACF.1			X
FRU_PRS.1	X		
FIA_UID.2			X
FMT_MSA.3			X
FDP_DMI.1			X

Table 8 SFRs / TOE Security Objectives coverage

6.5.2 SECURITY REQUIREMENT SUFFICIENCY

OT.PRIORITY: **FRU_PRS.1** requires that the TOE has to manage the usage of resources based on a priority-based queue.

OT.SAFE_SECURE_STATE: **FPT_FLS.1** requires that the TOE has to preserve a secure state after receiving an interruption caused by either hardware or software failure.

OT.ACCESS_CONTROL: The following requirements contribute to fulfil the objective:

- **FIA_ATD.1** enforces the management of the user identity and properties as security attributes which then become an input for access control functions.
- **FIA_UID.2** enforces the identification of the user before any action, thus allowing the access and services and data, and their processing, to authorized users only.
- **FIA_USB.1** enforces the association of the user identity to the active entity that acts on behalf of the user and to check that this is a valid identity. It is the starting point to ensure that only an authorized user accesses and processes services and data.

- **FDP_ACC.1**, **FDP_ACF.1** state the access control measures based on SFP.CAP, which establish capability-based access control policy to kernel objects. Thus, keeps kernel objects inaccessible to unauthorized users.
- **FDP_DMI.1** enforces memory isolation between memory in kernel space and in user space, hence preventing unauthorized access to confidential information in the kernel space.
- **FMT_MSA.3** requires that security attributes are static, this is enforced as the subjects that have been described have not been granted the ability to modify a cnode directly.

6.5.3 SFR DEPENDENCY RATIONALE

6.5.3.1 TABLE OF SFR DEPENDENCIES

The following table lists the dependencies for each requirement, indicating how they have been satisfied:

SFR	Required	Fulfilled	Missing
FIA_ATD.1	None	None	None
FIA_USB.1	FIA_ATD.1	FIA_ATD.1	None
FPT_FLS.1	None	None	None
FDP_ACC.1	FDP_ACF.1	FDP_ACF.1	None
FDP_ACF.1	FDP_ACC.1, FMT_MSA.3	FDP_ACC.1, FMT_MSA.3	None
FRU_PRS.1	None	None	None
FIA_UID.2	None	None	None
FMT_MSA.3	FMT_MSA.1, FMT_SMR.1	None	FMT_MSA.1, FMT_SMR.1
FDP_DMI.1	None	None	None

Table 9 SFR Dependencies

6.5.3.2 JUSTIFICATION FOR MISSING DEPENDENCIES

FMT_MSA.3 dependency on FMT_SMR.1

The security attributes can only be modified by the kernel, there are no roles that can make modifications to these attributes. Therefore, such roles are not necessary.

FMT_MSA.3 dependency on FMT_MSA.1

This requirement is for management of security attributes, as the TOE does not include management options for security attributes this security requirement has not been included.

The modification of the security attributes of a cnode is necessary within the lifecycle of a thread, modifications are performed through the use of operations that create or delete an object, or operations that flush the cspace of an object. These operations are invoked as an automatic response, therefore do not match the criteria required to be considered as management actions for FMT_MSA.1. The justification for non-inclusion is as follows;

- The creation of a kernel object (adding capabilities to the cnode) and the deletion of a kernel object (removing them from a threads cnode) are not considered as mechanisms that have the intention of modifying permissions. In both cases they are indirect mechanisms, by which the TSF automatically handles the cnode population (as performed by the kernel).
- Flushing is an operation that is able to purge the cspace of a thread, removing the capabilities within the private or initial cspace. The usage of this mechanism is restricted to the thread which invoked the operation, performing this operation to modify its own cnode. Therefore, as this function is exercised internally when the purging of the cspace of a thread is required, there is no intent of modifying the permissions of subjects on kernel objects.

6.5.4 SAR RATIONALE

The Evaluation Assurance Level 5 has been chosen to commensurate with the threat environment that is experienced by typical consumers of the TOE. The assurance level defined in this ST consists of the predefined assurance package EAL 5 with the augmentation ALC_FLR.1.

6.5.5 SAR DEPENDENCY RATIONALE

6.5.5.1 TABLE OF SAR DEPENDENCIES

SAR	Required	Fulfilled	Missing
ASE_CCL.1	ASE_INT.1, ASE_ECD.1, ASE_REQ.1	ASE_INT.1, ASE_ECD.1, ASE_REQ.2 (hierarchically above ASE_REQ.1)	None
ASE_ECD.1	None	None	None
ASE_INT.1	None	None	None

SAR	Required	Fulfilled	Missing
ASE_OBJ.2	ASE_SPD.1	ASE_SPD.1	None
ASE_REQ.2	ASE_OBJ.2, ASE_ECD.1	ASE_OBJ.2, ASE_ECD.1	None
ASE_TSS.1	ASE_INT.1, ASE_REQ.1, ADV_FSP.1	ASE_INT.1, ASE_REQ.2 (hierarchically above ASE_REQ.1), ADV_FSP.5 (hierarchically above ADV_FSP.1)	None
ALC_CMC.4	ALC_CMS.1, ALC_DVS.1, ALC_LCD.1	ALC_CMS.5 (hierarchically above ALC_CMS.1), ALC_DVS.1, ALC_LCD.1	None
ALC_CMS.5	None	None	None
ADV_FSP.5	ADV_TDS.1, ADV_IMP.1	ADV_TDS.4 (hierarchically above ADV_TDS.1), ADV_IMP.1	None
AGD_OPE.1	ADV_FSP.1	ADV_FSP.5 (hierarchically above ADV_FSP.1)	None
AGD_PRE.1	None	None	None
ATE_IND.2	ADV_FSP.2, AGD_OPE.1, AGD_PRE.1, ATE_COV.1, ATE_FUN.1	ADV_FSP.5 (hierarchically above ADV_FSP.2), AGD_OPE.1, AGD_PRE.1, ATE_COV.2 (hierarchically above ATE_COV.1), ATE_FUN.1	None
AVA_VAN.4	ADV_ARC.1, ADV_FSP.4, ADV_TDS.3, ADV_IMP.1, AGD_OPE.1, AGD_PRE.1, ATE_DPT.1	ADV_ARC.1, ADV_FSP.5 (hierarchically above ADV_FSP.4), ADV_TDS.4 (hierarchically above ADV_TDS.3), ADV_IMP.1, AGD_OPE.1, AGD_PRE.1,	None

SAR	Required	Fulfilled	Missing
		ATE_DPT.3 (hierarchically above ATE_DPT.1)	
ADV_ARC.1	ADV_FSP.1, ADV_TDS.1	ADV_FSP.5 (hierarchically above ADV_FSP.1), ADV_TDS.4 (hierarchically above ADV_TDS.1)	None
ADV_IMP.1	ADV_TDS.3, ALC_TAT.1	ADV_TDS.4 (hierarchically above ADV_TDS.3), ALC_TAT.2 (hierarchically above ALC_TAT.1)	None
ASE_SPD.1	None	None	None
ALC_DEL.1	None	None	None
ADV_INT.2	ADV_IMP.1, ADV_TDS.3, ALC_TAT.1	ADV_IMP.1, ADV_TDS.4 (hierarchically above ADV_TDS.3), ALC_TAT.2 (hierarchically above ALC_TAT.1)	None
ADV_TDS.4	ADV_FSP.5	ADV_FSP.5	None
ALC_DVS.1	None	None	None
ALC_LCD.1	None	None	None
ALC_TAT.2	ADV_IMP.1	ADV_IMP.1	None
ATE_COV.2	ADV_FSP.2, ATE_FUN.1	ADV_FSP.5 (hierarchically above ADV_FSP.2), ATE_FUN.1	None
ATE_DPT.3	ADV_ARC.1, ADV_TDS.4, ATE_FUN.1	ADV_ARC.1, ADV_TDS.4, ATE_FUN.1	None
ATE_FUN.1	ATE_COV.1	ATE_COV.2 (hierarchically above ATE_COV.1)	None

SAR	Required	Fulfilled	Missing
ALC_FLR.1	None	None	None

Table 10 SAR dependencies

7 TOE SUMMARY SPECIFICATION

The TOE performs the following security functions:

- User identification
- Capability-based Access Control
- Memory management
- Thread management
- Fault tolerance

7.1.1 USER IDENTIFICATION

When a thread attempts to invoke functionality within the TOE, it is passed to the scheduler to be handled by the kernel (S.KERNEL). Each thread (U.THREAD) is identified by its identifier (ATTR.THREAD.ID) and has two security attributes that specify its permissions (ATTR.PRIV_CSPACE.CNODE.CSLOT and ATTR.INIT_CSPACE.CNODE.CSLOT). The former specifies the capabilities owned by the thread, the latter specifies the capabilities of the threads created by the root service. The ATTR.INIT_CSPACE.CNODE.CSLOT can only be modified by the kernel through system calls invoked by the root service.

7.1.1.1 SFR SUMMARY

This security function covers the following: **FIA_ATD.1**, **FIA_UID.2** and **FIA_USB.1**.

7.1.2 CAPABILITY-BASED ACCESS CONTROL

The TOE enforces capability-based access control to protect the confidentiality and integrity of the objects. A *capability* is an unforgeable token of authority composed of the pointer to the kernel object and the object type. Therefore, a *capability* specifies the rights of a thread to perform an action in that specific kernel object. All threads can have multiple capabilities, which are stored in a cslot, which simply defined is a slot inside a structure called a cnode. When a thread makes a capability-based system call to a kernel object, the kernel first checks the calling thread's private_cspace to check if it has the corresponding capability to perform such action. If not, then the initial_cspace is evaluated in the same manner, to determine if the corresponding capability is therefore in the initial_cspace instead.

The requisites to be satisfied when a thread invokes a capability-based system call are the following:

1. Have the required capability in its cnode.
2. Pass as an argument the correct pointer to that kernel object.

As a result, not only does the thread have the required kernel object, but it also passes, as an argument, the corresponding capability to the kernel object it wants to operate. There are several types of capabilities, each for each type of kernel object. The type of capability englobes all the defined operations to perform in that type of object, hence when a thread has a capability of that type, it can perform all the available operations to that kernel object. For instance, if a thread has the capability type X, and X has 5 different available operations, the thread will be able to perform any of the 5 operations. For the types of capabilities, please refer to the application note of the requirement **FDP_ACC.1**.

The following attributes are initialized by the kernel upon creation of a thread; for the root service thread, the initial_cspace (ATTR.INIT_CSPACE.CNODE.CSLOT) is initialised by the kernel, after which the private capability space (ATTR.PRIV_CSPACE.CNODE.CSLOT) is initialised. During the lifetime of the cnode, the kernel will modify private_cspace or initial_cspace, this occurs when a thread invokes a system call to create a kernel object whereby the thread obtains the capability on that kernel object, so it is added to the respective cspace.

The cnode in either private_cspace or initial_cspace can be modified to remove capabilities on it, this action can be performed by either deleting the kernel object related to the capability, resulting in the capability being removed from the corresponding cspace, or the full cspace can be flushed through the invocation of a specific system call (CAP_THREAD_FLUSH).

Note: Notification and Endpoint objects are specifically included in initial_cspace when created, this is so different threads are able to communicate with each other. The rationale behind this is to avoid potential issues occurring; for example, when two threads attempt to access an object only in the cspace of one thread.

7.1.2.1 SFR SUMMARY

This security function covers the following SFRs: **FDP_ACC.1**, **FDP_ACF.1** and **FMT_MSA.3**.

7.1.3 MEMORY MANAGEMENT

The TOE configures and operates the MPU (non-TOE hardware) to enforce memory isolation between kernel space and user space. Only the threads operating in privileged thread mode have full access to both kernel and user layer, as the threads operating in unprivileged thread mode only have access to the memory in user space.

This security function covers the following SFR: **FDP_DMI.1**

7.1.4 THREAD MANAGEMENT

The TOE implements scheduling strategy for threads depending on if they have the same or different priority. For threads with different priorities *priority-based preemptible scheduling strategy* is adopted and round-robin round for threads with the same priority. This priority disseminates how much CPU time each subject has.

7.1.4.1 SFR SUMMARY

This security function covers the following SFR: **FRU_PRS.1**.

7.1.5 FAULT TOLERANCE

The TOE maintains a secure state when an error or failure occurs outside the TOE. These exceptions can be produced by the user layer, or by the hardware, either way the TOE is able to handle them. In case an unknown error occurs that generates interruptions to the TOE, which have not been contemplated, the TOE enters the secure state.

Moreover, during the initialization of the TOE, if an error occurs before initialization of the interrupt module, the TOE halts. However, if the TOE has already initialized the interrupt module, this error will be handled by such module.

In the case of either a software or hardware interrupt being received, the system call module is responsible for dealing with these issues, to do this it will invoke a system call which will cancel the thread that caused the interrupt.

7.1.5.1 SFR SUMMARY

This security function covers the following SFR: **FPT_FLS.1**.

8 ACRONYMS

The following table shows the acronyms used in this document.

Acronym	Meaning
PP	Protection Profile
CC	Common Criteria
TOE	Target of Evaluation
TSF	TOE Security Functionality
TSFi	TSF Interface
OSP	Organisational Security Policies
EAL	Evaluation Assurance Level
ST	Security Target
IT	Information Technology
IPC	Interprocess communication

Table 11 Abbreviations

9 GLOSSARY OF TERMS

The following table shows the glossary of terms used in this document.

Term	Meaning
Augmentation	Addition of one or more requirement(s) to a package
Evaluation Assurance Level	Set of assurance requirements drawn from CC Part 3, representing a point on the CC predefined assurance scale, that form an assurance package.
Operational Environment	Environment in which the TOE is operated.
Protection Profile	Implementation-independent statement of security needs for a TOE type.
Security Target	Implementation-dependent statement of security needs for a specific identified TOE.
Target Of Evaluation	Set of software, firmware and/or hardware possibly accompanied by guidance.
Memory Protection Unit (MPU)	A computer hardware unit that provides memory protection, usually implemented as part of the central processing unit (CPU).
Capability	Permission for a subject to perform an operation on a kernel object.
cnode	A list of permission a thread has, each element of a list is known as a cslot.
cslot	An entry of the cnode, consists of a pair of values; a pointer of the kernel object to which invoke the operation, and the operation.
TCB	The Thread Control Block, it is a library of information about the threads within the system.
Thread	The smallest sequence of programmed instructions that can be managed independently by a scheduler.
Kernel Object	A memory block allocated by the kernel and is accessible only by the kernel, a data structure whose members contain information about the object.
Private capability space	cnode associated to the threads, contains all the permissions a specific thread has.
Initial capability space	cnode of the root service thread, accessible by all the threads in user space.
root service	Primary thread of the user space, it creates the rest of the threads.
Kernel	The core component of an operating system, it serves as the interface between the physical hardware and the processes running on it.
Microkernel	The entirety of the operating system and the Kernel combined, designed to run with the minimal number of functions required to run.
SYSTICK	SYSTICK is a simple timer that is part of the NVIC controller in the Cortex-M microprocessor, intended to provide a periodic interrupt for an RTOS, but it can be used for other simple timing purposes in addition.

Term	Meaning
RPC	Remote Procedure Call, used when a program causes a procedure to execute in a different address space.

Table 12 Glossary of terms

10 DOCUMENT REFERENCES

The following table shows the references used in this document.

Reference	Document
[CC31R5P1]	Common Criteria for Information Technology Security Evaluation, Version 3.1, Revision 5, Part 1: Introduction and general model
[CC31R5P2]	Common Criteria for Information Technology Security Evaluation, Version 3.1, Revision 5, Part 2: Security functional components
[CC31R5P3]	Common Criteria for Information Technology Security Evaluation, Version 3.1, Revision 5, Part 3: Security assurance components
[CEM31R5P3]	Common Criteria Evaluation methodology, Version 3.1, Revision 5

Table 13 List of document references