QNX Software Systems
QNX Neutrino® Secure Kernel v6.4

Security Target

Evaluation Assurance Level: EAL4+
Document Version: 1.1

Prepared for:

QNX Software Systems
175 Terence Matthews Crescent
Ottawa, Ontario K2M 1W8
Canada
Phone: +1 (613) 591-0931
http://www.qnx.com

Prepared by:

Corsec Security, Inc.
10340 Democracy Lane, Suite 201
Fairfax, VA 22030
United States of America
Phone: +1 (703) 267-6050
http://www.corsec.com

Revision History

Version	Modification Date	Modified By	Description of Changes
0.1	2008-02-24	Nathan Lee	Initial draft.
0.2	2008-02-26	Nathan Lee	Updates based on QSS feedback.
0.3	2008-03-04	Nathan Lee	Updates based on Corsec QRP and QSS feedback.
0.4	2008-05-16	Nathan Lee	Updates based on lab ORs.
0.5	2008-06-06	Nathan Lee	Miscellaneous updates to harmonize ST with design documentation.
0.6	2008-07-31	Greg Milliken	Updates based on lab ORs.
0.7	2008-10-30	Nathan Lee	Updates based on lab ORs and ADV updates.
0.8	2008-11-25	Nathan Lee	Miscellaneous updates and re-incorporated previously removed SFRs based on new lab ORs.
0.9	2008-12-01	Nathan Lee	Updates based on lab ORs.
0.9.1	2008-12-02	Nathan Lee	Corrected typo in Application Note for FDP_ACF.1.
1.0	2008-12-08	Nathan Lee	Updates based on lab ORs.
1.1	2008-12-15	Nathan Lee	Added detail on shared memory resource manager.

Table of Contents

Table of Figures

Table of Tables

1 Security Target Introduction

This section identifies the Security Target (ST), Target of Evaluation (TOE), ST conventions, ST conformance claims, and the ST organization. The Target of Evaluation is the QNX Neutrino® Secure Kernel v6.4, and will hereafter be referred to as the TOE throughout this document. The TOE is a secure kernel and *C*-language library for the QNX Neutrino® Realtime Operating System (RTOS).

1.1 Purpose

This ST provides mapping of the Security Environment to the Security Requirements that the TOE meets in order to remove, diminish or mitigate the defined threats in the following sections:

- Security Target Introduction (Section 1) – Provides a brief summary of the ST contents and describes the organization of other sections within this document.
- TOE Description (Section 2) – Provides an overview of the TOE security functions and describes the physical and logical boundaries for the TOE.
- Security Environment (Section 3) – Describes the threats and assumptions that pertain to the TOE and its environment.
- Security Objectives (Section 4) – Identifies the security objectives that are satisfied by the TOE and its environment.
- Security Requirements (Section 5) – Presents the Security Functional Requirements (SFRs) and Security Assurance Requirements (SARs) met by the TOE and by the TOE's environment.
- TOE Summary Specification (Section 6) – Describes the security functions provided by the TOE that satisfy the security functional requirements and objectives.
- Protection Profile Claims (Section 7) – Provides the identification of any ST Protection Profile claims as well as a justification to support such claims.
- Rationale (Section 8) – Presents the rationale for the security objectives, requirements, and the TOE summary specifications that relate to their consistency, completeness, and suitability.
- Acronyms (Section 9) – Defines the acronyms used within this ST.

1.2 Security Target, TOE and CC Identification and Conformance

Table 1 - ST, TOE, and CC Identification and Conformance

ST Title	QNX Software Systems QNX Neutrino® Secure Kernel v6.4 Security Target
ST Version	Version 1.1
Author	Corsec Security, Inc. Nathan Lee
TOE Identification	QNX Neutrino® Secure Kernel v6.4
Common Criteria (CC) Identification and Conformance	Common Criteria for Information Technology Security Evaluation, Version 2.3, August 2005 (aligned with ISO/IEC 15408:2005); CC Part 2 conformant; CC Part 3 conformant; PP claim (none); Parts 2 and 3 Interpretations from the Interpreted CEM as of February 25, 2008 were reviewed, and no interpretations apply to the claims made in this ST.
Protection Profile (PP) Identification	None.
Evaluation Assurance Level	EAL4+ (augmented with ALC_FLR.1)
Keywords	Realtime operating system, RTOS, kernel

1.3 Conventions and Terminology

1.3.1 Conventions

There are several font variations used within this ST. Selected presentation choices are discussed here to aid the Security Target reader.

The CC allows for assignment, refinement, selection and iteration operations to be performed on security functional requirements. These operations are performed as described in Parts 2 and 3 of the CC, and are shown as follows:

- Completed assignment statements are identified using [*italicized text within brackets*].
- Completed selection statements are identified using [*underlined italicized text within brackets*].
- Refinements are identified using **bold text**. Any text removed is stricken (Example: ~~TSF Data~~) and should be considered as a refinement.
- Iterations are identified by appending a letter in parentheses following the component title. For example, FAU_GEN.1(a) Audit Data Generation would be the first iteration and FAU_GEN.1(b) Audit Data Generation would be the second iteration.

1.3.2 Terminology

This document assumes that the reader is already familiar with fundamental computer science and programming concepts and terminology, and so does not define them here. However, the following terminology clarifications are provided due to the specialized nature of the TOE and this Security Target document:

- **Process**: A non-schedulable entity, which defines the address space and a few data areas. A process must have at least one thread running in it.
- **Thread**: The schedulable entity under QNX Neutrino. A thread is a flow of execution. Each thread exists within the context of a single process.
- **User**: A thread with a priority from 1-63, which lacks root privileges.
- **Administrator**: A thread with a priority from 64-255, which possesses root privileges.

- **CPU Bandwidth**: CPU bandwidth refers to the available portion of the CPU available to the AP at a given time. CPU bandwidth does not reflect the total time or number of CPU cycles that a process or thread uses.

2 TOE Description

The TOE Description provides an overview of the TOE. This section describes the general capabilities and security functionality of the TOE. The TOE description provides a context for the TOE evaluation by identifying the product type, describing the product, and defining the specific evaluated configuration.

2.1 Product Type

The QNX Neutrino® Secure Kernel provides the microkernel for the QNX Neutrino Realtime Operating System. QNX Neutrino provides a memory-protected microkernel architecture for reliable, scalable, and realtime performance for embedded applications.

2.2 Product Description

The QNX Neutrino RTOS is designed for applications requiring nonstop, 24 hours a day, 365 days a year operation. It implements POSIX-compliant users, groups, permissions, usermasks, processes, threads, and priorities:

- Users: On QNX Neutrino and other Unix-like operating systems, users are identified within the kernel by an integer value called a "user identifier", often abbreviated as "UID" or "user ID". Permissions can be assigned to individual users. As implied in Section 1.3.2 above, users in the context of this TOE are threads executing within the kernel.

- Groups: On QNX Neutrino and other Unix-like systems, multiple users can be categorized into groups, which allows system permissions to be delegated to groups of users in an organized fashion. Groups are identified within the kernel by an integer valued called a "group identifier", often abbreviated as "GID" or "group ID".

- Permissions: Users and groups can be granted privileges on the system in the form of permission bits. Permissions indicate what actions that user or group is allowed to perform.

- Usermasks: Each user (and each process associated with that user) has a usermask (often abbreviated “umask”) which limits what permission bits a process may set when creating an object in the pathname space. If a bit is “on” in the user’s umask then Neutrino will not allow the corresponding permission to be set for objects created by that user.

- Processes: Processes, defined in Section 1.3.2 above, contain threads.

- Threads: Threads, also defined in Section 1.3.2 above, are the primary “agents” that act on users’ behalves. Threads are contained in processes.

- Priorities: A priority is an attribute of processes and threads, in the form of an integer in the range 0 – 255, that determines how “important” that particular process or thread is. Processes or threads with lower priorities are “less important” than processes or threads with higher priorities, and Neutrino’s scheduler function uses each process’ and thread’s priority as a component in determining how much CPU bandwidth each process and thread should receive. Each process and thread has two types of priority: an *actual* priority and an *effective* priority. The actual priority is the priority set by the process or thread itself, and indicates to the kernel how important the process or thread considers itself to be. The effective priority, however, is the priority that Neutrino’s scheduler function is currently using to schedule that process or thread for access to controlled resources – the effective priority is adjusted up and down by the scheduler on a regular basis, while the actual priority is only adjusted by the process or thread itself.

- Pathname space: The QNX pathname space is an abstract container that holds a logical grouping of unique identifiers (pathnames). All devices, filesystems, and other addressable system entities are contained by and addressed via the pathname space. Since QNX Neutrino is a distributed, microkernel OS with virtually all non-kernel functionality provided by user-installable programs, objects in the pathname space are

managed by user-level or kernel-level server programs called “resource managers”. A resource manager registers with the Process Manager to manage a specific portion of the pathname space, and then all accesses to that portion of the pathname space are mediated by that resource manager (that is, the resource manager determines what to do with the access request). The only resource manager contained within the TOE boundary of the QNX Neutrino Secure Kernel v6.4 is the shared memory resource manager, which is part of *procnto*.

QNX Neutrino® RTOS is built on a true microkernel architecture. The QNX Neutrino Secure Kernel operates as a self-contained, protected microkernel within the QNX Neutrino RTOS. With the QNX Neutrino RTOS, every driver, application, protocol stack, and file system runs outside the QNX Neutrino Secure Kernel, in the safety of memory-protected user space. This modularity has two primary benefits:

1. The QNX Neutrino Secure Kernel can be used as a core system building block that can be deployed in conjunction a variety of optional operating system technologies such as networking stacks or file systems. This gives the flexibility of tailoring the set of QNX Neutrino RTOS components, or customer developed components that can be used to implement a system.

2. Virtually any component can fail and be automatically restarted without affecting other components or the QNX Neutrino Secure Kernel. This inherently modular design allows administrators and developers to dynamically upgrade modules, introduce new features, or deploy bug fixes without costly downtime or system outages.

The adaptive partitioning feature of the QNX Neutrino Secure Kernel provides CPU[1] time guarantees to provide a level of protection (known as partitions) between groups of processes and threads. To achieve the highest level of performance, adaptive partitioning allows applications to use all available CPU cycles under normal operating conditions. During overload conditions, adaptive partitioning enforces hard resource guarantees, ensuring applications receive their budgeted share of CPU time.

The QNX Neutrino Secure Kernel symmetric multiprocessing (SMP) feature allows user processes to execute on any CPU core in a SMP complex.

2.3 TOE Boundary and Scope

This section will primarily address what physical and logical components of the TOE are included in evaluation. Figure 1 illustrates the TOE boundary:

[1] CPU: Central Processing Unit

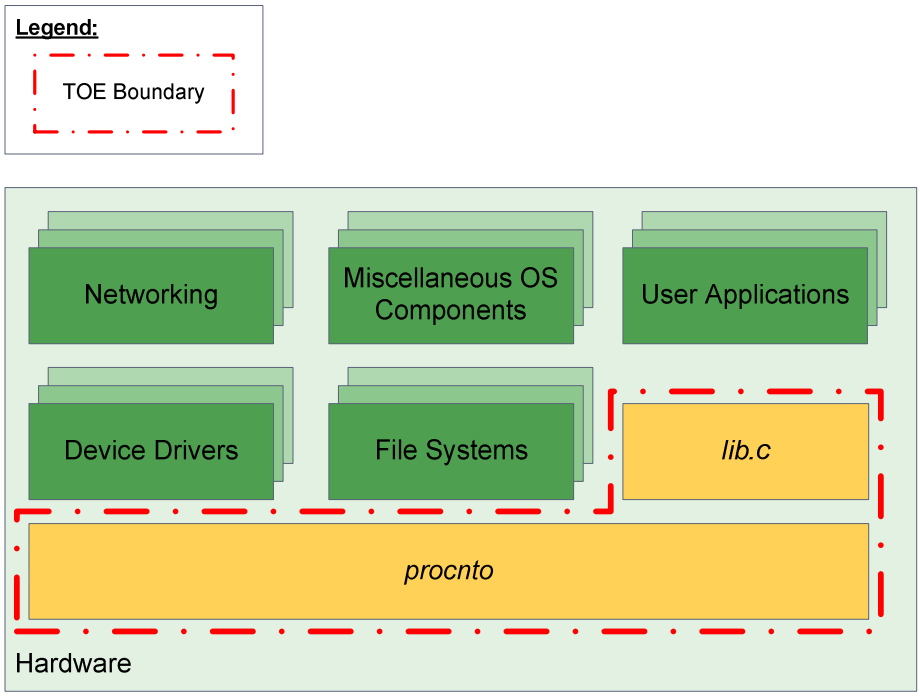

Figure 1 - TOE Boundary

2.3.1 Physical Boundary

The TOE is a software-only product comprising the *procnto* RTOS kernel and the *lib/c C*-language library. It is installed along with various other external support components as part of a full operating system deployment as depicted in Figure 1 above. The TOE runs on the following CPU architectures:

- ARM9
- ARM11
- x86 Multicore

2.3.1.1 TOE Environment

The TOE Environment consists of the following components:

- Other operating system components
- Applications
- Hardware

2.3.1.1.1 Security Considerations in the TOE Environment:

The TOE is a RTOS kernel and C language support library, and is intended to be embedded in an appliance with other utility software – it is not designed for stand-alone deployment. Its architecture focuses on providing reliable execution of realtime, mission-critical applications, and for this reason the TOE itself does not implement the traditional set of IT security checks-and-balances, instead leaving these up to the TOE Environment. When the TOE is deployed as part of a larger, properly configured system it will perform its functions as designed; care must be taken by the TOE administrators to ensure that the hardware on which the TOE is installed, and the other operating system components with which it interacts, are properly designed, configured, and deployed.

In order to build the TOE, the TOE administrator must use the appropriate build file and build options. These are detailed in the *QNX Neutrino® Secure Kernel v6.4 Common Criteria Guidance Supplement* document. The build file is a text file containing instructions that specify the contents and other details of an OS image that can be built with automated tools. The build file and build options ensure that the source code that constitutes the TOE is built in a manner that provides the security and configuration that was evaluated. The build file should be reviewed and controlled by the TOE administrator at all times to ensure that the TOE is properly built and configured.

In order to operate the TOE in the CC-compliant configuration, TOE administrators must ensure that all application interactions with *procnto* are mediated by and occur through the *lib/c* library – applications are not allowed to access or communicate with *procnto* directly.

2.3.2 Logical Boundary

The TOE comprises the functionality provided by the *procnto* microkernel and the *lib/c* library when they are compiled in the CC-compliant configuration as specified in the Common Criteria Guidance Supplement. Two notable features of the CC-compliant configuration of *procnto* are Adaptive Partitioning (AP) and zeroization of released memory.

The TOE's logical boundary includes all of the claimed TSFs. The SFRs implemented by the TOE are grouped under the following Security Function Classes:

- FDP User Data Protection
- FIA Identification and Authentication
- FMT Security Management
- FPT Protection of the TSF
- FRU Resource Utilization

These functions are discussed in greater detail in Section 6.1 below.

2.3.3 Physical and Logical Features and Functionality Not Included in the Evaluated Configuration of the TOE

Although the TOE can be acquired as a stand-alone product from QNX, it is more commonly distributed as one component of the QNX Neutrino RTOS development package. This package includes many optional software components which are not utilized by all developers. The TOE boundary excludes these optional components and includes only *procnto* and *lib/c* in order to provide developers with the broadest range of CC-compliant development options.

3 Security Environment

This section describes the security aspects of the environment in which the TOE 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:

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

3.1 Assumptions

This section describes the security aspects of the intended environment for the evaluated TOE. The operational environment must be managed in accordance with assurance requirement documentation for delivery, operation, and user guidance. The following specific conditions are required to ensure the security of the TOE and are assumed to exist in an environment where this TOE is employed.

Table 2 - Assumptions

Name	Description
A.MANAGE	There is one or more competent individuals (administrators) assigned to manage the TOE.
A.NOEVIL	The people administering the TOE and writing processes and threads for execution by the TOE are non-hostile, appropriately trained, and follow all guidance.
A.PHYSICAL	It is assumed that the non-IT environment provides the TOE with appropriate physical security commensurate with the value of the IT assets protected by the TOE.
A.TRUSTED_INDIVIDUAL	It is assumed that any individual allowed to perform procedures upon which the security of the TOE may depend is trusted with assurance commensurate with the value of the IT assets.

3.2 Threats to Security

This section identifies the threats to the IT assets against which protection is required by the TOE or by the security environment. The threat agents are divided into two categories:

- TOE administrators: They have extensive knowledge of how the TOE operates and are assumed to possess a high skill level, moderate resources to alter TOE configuration settings/parameters and physical access to the TOE. (TOE administrators are, however, assumed not to be willfully hostile to the TOE)
- Misbehaving processes or threads which the TOE executes: They are created by programmers who have extensive knowledge of how the TOE operates and are assumed to possess a high skill level, limited resources to alter TOE configuration settings/parameters and no physical access to the TOE.

TOE Administrators are assumed to have no motivation to attack the TOE, and misbehaving processes or threads are assumed to be created by programmers with low motivation to attack the TOE. The IT assets requiring protection are the user data transitioning through the TOE and the processes and threads being executed by the TOE. Removal, diminution and mitigation of the threats are through the objectives identified in Section 4 - Security Objectives.

The following threats are applicable:

Table 3 - Threats

Name	Description
T.DENIAL_OF_SERVICE	A misbehaving thread may block others from system resources (i.e., processing time) via a resource exhaustion attack.
T.INSTALL	An administrator may incorrectly install or configure the TOE, resulting in ineffective security mechanisms.
T.UNAUTHORIZED _ACCESS	A process or thread may gain access to resources or TOE security management functions for which it is not authorized according to the TOE security policy.

3.3 Organizational Security Policies

There are no Organizational Security Policies.

4 Security Objectives

This section identifies the security objectives for the TOE and its supporting environment. The security objectives identify the responsibilities of the TOE and its environment to meet the TOE's security needs.

4.1 Security Objectives for the TOE

The specific security objectives are as follows:

Table 4 - Security Objectives for the TOE

Name	Description
O.ACCESS	The TOE will ensure that processes and threads gain only authorized access to resources.
O.EXECUTION_PRIORITY	The TOE will provide mechanisms that ensure that processes and threads with higher priorities and higher Adaptive Partitioning budgets are given more access to CPU time than processes and threads of lower priorities or lower AP budgets.
O.FAILURE_ISOLATION	The TOE will prevent a failure of one process or thread from affecting other unrelated processes and threads.
O.RESIDUAL_INFORMATION	The TOE will ensure that any information contained in a resource is not released to processes or threads when the resource is reallocated.
O.RESOURCE_ALLOCATION	The TOE will provide mechanisms that enforce constraints on the allocation of resources.
O.SUBJECT_ISOLATION	The TOE will provide mechanisms to protect each process or thread from unauthorized interference by other processes or threads.

4.2 Security Objectives for the Environment

4.2.1 IT Security Objectives

The are no IT security objectives are to be satisfied by the environment.

4.2.2 Non-IT Security Objectives

The following non-IT environment security objectives are to be satisfied without imposing technical requirements on the TOE. That is, they will not require the implementation of functions in the TOE hardware and/or software. Thus, they will be satisfied largely through application of procedural or administrative measures.

Table 5 - Non-IT Security Objectives

Name	Description
OE.ADMIN_GUIDANCE	The TOE will provide administrators with the necessary information for secure management of the TOE.
OE.INSTALL_GUIDANCE	The TOE will be delivered with the appropriate installation guidance to establish and maintain TOE security.

Name	Description
OE.MANAGE	Sites deploying the TOE will provide competent, non-hostile TOE administrators who are appropriately trained and follow all administrator guidance. TOE administrators will ensure the system is used securely.
OE.PHYSICAL	Physical security will be provided for the TOE by the non- IT environment commensurate with the value of the IT assets protected by the TOE.
OE.TRUSTED_INDIVIDUAL	Any individual allowed to perform procedures upon which the security of the TOE may depend must be trusted with assurance commensurate with the value of the IT assets.
OE.INSTALL	Those responsible for the TOE must ensure that the TOE is delivered, installed, managed, and operated in a manner that prevents disclosure, modification, destruction, and other threats to the TOE that result from a deficiency in delivery or customer site security.

5 Security Requirements

This section defines the Security Functional Requirements (SFRs) and Security Assurance Requirements (SARs) met by the TOE as well as Security Functional Requirements met by the TOE IT environment. These requirements are presented following the conventions identified in Section 1.3.1.

5.1 TOE Security Functional Requirements

This section specifies the SFRs for the TOE. This section organizes the SFRs by CC class. Table 6 identifies all SFRs implemented by the TOE and indicates the ST operations performed on each requirement.

Table 6 - TOE Security Functional Requirements

Name	Description	S	A	R	I
FDP_ACC.1	Subset access control		✓		
FDP_ACF.1	Security attribute based access control		✓	✓	
FDP_IFC.1	Subset information flow control		✓		
FDP_IFF.1	Simple security attributes		✓	✓	
FDP_RIP.2	Full residual information protection	✓		✓	
FIA_ATD.1	User attribute definition		✓	✓	
FIA_UID.2	User identification before any action			✓	
FMT_MOF.1	Management of security functions behavior	✓	✓		
FMT_MSA.1(a)	Management of security attributes	✓	✓		✓
FMT_MSA.1(b)	Management of security attributes	✓	✓		✓
FMT_MSA.1(c)	Management of security attributes	✓	✓	✓	✓
FMT_MSA.3	Static attribute initialisation	✓	✓	✓	
FMT_SMF.1	Specification of management functions		✓		
FMT_SMR.1	Security roles		✓	✓	
FPT_FLS.1	Failure with preservation of secure state		✓		
FPT_RVM.1	Non-bypassability of the TSP				
FPT_SEP.1	TSF domain separation				
FRU_FLT.1	Degraded fault tolerance		✓		
FRU_PRS.1	Limited priority of service		✓	✓	
FRU_RSA.1	Maximum quota	✓	✓		

Note: S=Selection; A=Assignment; R=Refinement; I=Iteration

Section 5.1 contains the functional components from the Common Criteria (CC) Part 2 with the operations completed. For the conventions used in performing CC operations please refer to Section 1.3.1.

5.1.1 Class FDP: User Data Protection

FDP_ACC.1 Subset access control

Hierarchical to: No other components.

FDP_ACC.1.1

The TSF shall enforce the [*access control SFP*] on [

- *Subjects: listed in Table 7*
- *Objects: listed in Table 7*
- *Operations: listed in Table 7*].

Dependencies: FDP_ACF.1 Security attribute based access control

Table 7 – Access Control Matrix

Subject	Subject Attribute (Control Parameter)	Operation	Object (Resource)	Object Attribute	Protection Offered
Threads	*Effective priority*	*Use*	*CPU bandwidth*	*None*	*If running the Adaptive Partitioning (AP) Scheduler, groups of threads are executed based on their effective priorities and their AP assignments.*

Application Note: Table 7 provides a matrix detailing which operations threads ("subjects" in CC-terminology) may attempt to perform on resources (called "objects" in CC-terminology), and what attributes of each are used to determine whether or not to allow the operation. This matrix is referred to by FDP_ACC.1 and FDP_ACF.1. In short: a thread's use of CPU time is mediated by the thread's effective priority, which is calculated by the Scheduler using the thread's self-set actual priority as a parameter.

FDP_ACF.1 Security attribute based access control

Hierarchical to: No other components.

FDP_ACF.1.1

The TSF shall enforce the [*access control SFP*] to objects based on the following: [

- *Subjects: listed in Table 7*
- *Subject attributes: listed in Table 7*
- *Objects: listed in Table 7*
- *Object attributes: listed in Table 7*].

FDP_ACF.1.2

The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed: [*if the thread currently has the highest effective priority of all threads, as determined by the Scheduler, then grant the thread access to CPU time, otherwise deny the access*].

FDP_ACF.1.3

The TSF shall **not** explicitly authorise access of subjects to objects ~~based on the following additional rules: [assignment: rules, based on security attributes, that explicitly authorise access of subjects to objects]~~.

FDP_ACF.1.4

The TSF shall **not** explicitly deny access of subjects to objects ~~based on the [assignment: rules, based on security attributes, that explicitly deny access of subjects to objects]~~.

Dependencies: **FDP_ACC.1 Subset access control**
FMT_MSA.3 Static attribute initialization

Application Note: FDP_ACC.1 and FDP_ACF.1 describe the mechanism used to determine whether or not a thread is allowed to access a shared memory object. In short: if the thread's associated user ID has permission then allow the access, otherwise deny the access.

FDP_IFC.1 Subset information flow control

Hierarchical to: **No other components.**

FDP_IFC.1.1

The TSF shall enforce the [*information flow control SFP*] on [

- *Subjects: listed in Table 8*
- *Information: listed in Table 8*
- *Operations: listed in Table 8*].

Dependencies: **FDP_IFF.1 Simple security attributes**

Table 8 – Information Flow Control Matrix

Subject	Subject Attribute (Control Parameter)	Operation	Object (Resource)	Object Attribute	Protection Offered
Threads	*Usermask*	*Create*	*Symbol in the pathname space*[2]	*Permission bits*	*Used to determine the default permissions when an object is created in the pathname space. This operation can be attempted by any thread; however, its success or failure will depend upon the thread's permissions to access that part of the pathname space, as determined by the resource manager.*[2]
	User ID and Group ID[3]	*Access,* *Create*	*Symbol in the pathname space*[2]	*Permission bits*	*Threads must have appropriate permissions (either explicit permissions, or implicit permissions due to running as root) to access symbols in the pathname space, as determined by the applicable resource manager.*[2]

Application Note: Table 8 provides a matrix detailing which operations threads ("subjects" in CC-terminology) may attempt to perform on symbols in the pathname space (which, in the CC-evaluated configuration, are shared memory objects), and what attributes of each are used to determine whether or not to allow the operation. This matrix is referred to by FDP_IFC.1 and FDP_IFF.1. In short:

- *A thread's usermask is used to determine what permission bits are applied to an object created by that thread in the pathname space. Threads may create symbols in the pathname space hierarchy and apply access restrictions to them in the form of permission bits. Any thread may create a new symbol in the "root" of the pathname space hierarchy (as long as the root of the pathname space is not being managed by another thread or process). A thread may create a symbol under a pre-existing symbol (a "branch") of the pathname space if the branch is owned by that thread or process, or if the permission bits on that branch (including the pathname space root, if it is being managed by another thread or process) allow symbol creation to the thread.*

- *A thread's user ID and group ID is used to determine whether or not it has permission to access a resource in the pathname space, based on the object's permission bits. "Access" permission includes both permission to read or write to pre-existing symbols and permission to create symbols under a pre-existing symbol. The resource manager (in the case of the TOE, the shared memory resource manager in* procnto*)*

[2] In the CC-evaluated configuration, symbols in the pathname space are shared memory objects, and the resource manager for shared memory objects is contained within *procnto*.

[3] The Real User ID and Group ID are inherited from the calling primitive (`fork`, `vfork`, `exec`, or `spawn`). The effective User ID and Group ID are also determined this way for the `fork` primitive. For `vfork`, `exec` and `spawn`, effective User ID and Group ID can be determined by arguments passed by the calling primitive. See http://qnx.com/developers/docs/6.4.0/neutrino/sys_arch/proc.html for more information. Additional information on each primitive can be found at http://qnx.com/developers/docs/6.4.0/neutrino/lib_ref/s/spawn.html, http://qnx.com/developers/docs/6.4.0/neutrino/lib_ref/f/fork.html, http://qnx.com/developers/docs/6.4.0/neutrino/lib_ref/v/vfork.html, and http://qnx.com/developers/docs/6.4.0/neutrino/lib_ref/e/execl.html. A description of what each User ID and Group ID means can be found at http://qnx.com/developers/docs/6.4.0/neutrino/user_guide/accounts.html.

maintains a list of access permissions for the objects under its control and determines whether or not to allow an access attempt.

FDP_IFF.1 Simple security attributes

Hierarchical to: No other components.

FDP_IFF.1.1

The TSF shall enforce the [*information flow control SFP*] based on the following types of subject and information security attributes: [

- *Subjects: listed in Table 8*
- *Subject attributes: listed in Table 8*
- *Information: listed in Table 8*
- *Information attributes: listed in Table 8*].

FDP_IFF.1.2

The TSF shall permit an information flow between a controlled subject and controlled information via a controlled operation if the following rules hold: [

- *Pathname space symbol creation: use the usermask of the thread to determine what permission bits to set on the newly-created symbol*
- *Pathname space symbol access: if the permission bits of the shared memory allow access to the thread's user ID or group ID, or if the thread is root, then allow the access, otherwise deny the access*].

Application Note: The "pathname space symbol creation" operation can be attempted by any thread; however, its success or failure will depend upon the thread's permissions to access that part of the pathname space.

FDP_IFF.1.3

The TSF shall enforce **no additional rules** ~~the [assignment: additional information flow control SFP rules]~~.

FDP_IFF.1.4

The TSF shall provide **no additional SFP capabilities** ~~the following [assignment: list of additional SFP capabilities]~~.

FDP_IFF.1.5

The TSF shall **not** explicitly authorise an information flow ~~based on the following rules: [assignment: rules, based on security attributes, that explicitly authorise information flows]~~.

FDP_IFF.1.6

The TSF shall **not** explicitly deny an information flow ~~based on the following rules: [assignment: rules, based on security attributes, that explicitly deny information flows]~~.

Dependencies: **FDP_IFC.1 Subset information flow control**
FMT_MSA.3 Static attribute initialisation

FDP_RIP.2 Full residual information protection

Hierarchical to: **FDP_RIP.1**

FDP_RIP.2.1

The TSF shall ensure that any previous information content of a **volatile memory** resource is made unavailable upon the [*deallocation of the resource from*] all objects.

Dependencies: **No dependencies**

Application Note: *When compiled in the CC-evaluated configuration, memory objects are zeroized when they are freed.*

5.1.2 Class FIA: Identification and Authentication

FIA_ATD.1 User attribute definition

Hierarchical to: No other components.

FIA_ATD.1.1

The TSF shall maintain the following list of security attributes belonging to individual ~~user~~ **processes or threads**: [

- *user mask*
- *User ID*
- *Group ID*
- *actual priority*
- *effective priority*
- *list of associated Adaptive Partitioning assignments*].

Dependencies: No dependencies

Application Note: Process and threads have other attributes which are defined in struct `thread_entry` and struct `process_entry`. The attributes listed in FIA_ATD.1 are the attributes specifically used to implement the TOE functionality described in this Security Target.

Note that processes and threads have two types of "priorities": "actual" priority and "effective" priority. The actual priority is a value that is set by the process or thread itself and indicates to the scheduler "how important" the process or thread currently considers itself to be; the effective priority is a value calculated by the scheduler which indicates "how important" the scheduler currently considers that process or thread to be. The scheduler uses the actual priority as a parameter when calculating the value of the effective priority, and only the effective priority is used by the scheduler to make scheduling decisions.

Note also that processes and threads may be assigned to more than one Adaptive Partition (thus, the applicable attribute above is "list of associated Adaptive Partitioning assignments").

FIA_UID.2 User identification before any action

Hierarchical to: FIA_UID.1

FIA_UID.2.1

The TSF shall require each ~~user~~ **process or thread** to ~~identify itself~~ **be identified** before allowing any other TSF-mediated actions on behalf of that ~~user~~ **process or thread**.

Dependencies: No dependencies

5.1.3 Class FMT: Security Management

FMT_MOF.1 Management of security functions behaviour

Hierarchical to: No other components.

FMT_MOF.1.1

The TSF shall restrict the ability to [*modify the behaviour of*] the functions [*create a process or thread, kill a process or thread*] to [*the process or thread itself, other threads within the same process, or a process or thread with root privileges*].

Dependencies: FMT_SMF.1 Specification of management functions
FMT_SMR.1 Security roles

Application Note: For the purposes of FMT_MOF.1, "modify the behavior of" is intended to mean "perform".

FMT_MSA.1(a) Management of security attributes

Hierarchical to: No other components.

FMT_MSA.1.1(a)

The TSF shall enforce the [*Information Flow Control SFP*] to restrict the ability to [*modify*] the security attributes [*user mask, User ID, Group ID*] to [*a process or thread with root privileges (based on its priority as defined in FMT_SMR.1)*].

Dependencies: [FDP_ACC.1 Subset access control or
FDP_IFC.1 Subset information flow control]
FMT_SMF.1 Specification of management functions
FMT_SMR.1 Security roles

FMT_MSA.1(b) Management of security attributes

Hierarchical to: No other components.

FMT_MSA.1.1(b)

The TSF shall enforce the [*Access Control SFP*] to restrict the ability to [*modify*] the security attributes [*effective priority, list of associated Adaptive Partitioning assignments*] to [*no role*].

Dependencies: [FDP_ACC.1 Subset access control or
FDP_IFC.1 Subset information flow control]
FMT_SMF.1 Specification of management functions
FMT_SMR.1 Security roles

Application Note: The effective priority is not, in fact, modified by any role, but rather is adjusted (calculated) by the TOE itself as a way of performing scheduling "bookkeeping." For this reason, it is listed as being modified by "no role". The list of associated Adaptive Partitioning assignments is specified by the process' programmer as part of the process itself, and is not modified during runtime.

FMT_MSA.1(c) Management of security attributes

Hierarchical to: No other components.

FMT_MSA.1.1(c)

The TSF shall ~~enforce the [*Access Control SFP*] to~~ restrict the ability to [*modify*] the security attributes [*actual priority*] to [*the process or thread itself, or another process or thread with root privileges (based on its priority as defined in FMT_SMR.1)*].

Dependencies: [FDP_ACC.1 Subset access control or
FDP_IFC.1 Subset information flow control]
FMT_SMF.1 Specification of management functions
FMT_SMR.1 Security roles

Application Note: The actual priority can be modified by the process or thread itself, and any other process or thread running with root privileges. This restriction is due to the design of the TOE itself, rather than an arbitrary SFP, and so the SFR has been refined to reflect this.

FMT_MSA.3 Static attribute initialisation

Hierarchical to: No other components.

FMT_MSA.3.1

The TSF shall enforce the [*Access Control SFP and Information Flow Control SFP*] to provide [*permissive*] default values for security attributes that are used to enforce the SFP.

FMT_MSA.3.2

The TSF shall allow the [*processes and threads with root privileges, or the process or thread itself*] to specify alternative initial values to override the default values **(except for the default value of the object's effective priority, which may not be overridden by any subject)** when an object or information is created.

Dependencies: FMT_MSA.1 Management of security attributes
FMT_SMR.1 Security roles

Application Note: Each process and thread have attributes which are defined in struct `thread_entry` and struct `process_entry` when the process or thread is created. The attributes listed in FIA_ATD.1 are the attributes specifically used to implement the TOE functionality described in this Security Target.

Each process and thread has an "actual" and "effective" priority. At process or thread creation, and at all times afterward, the scheduler function of `procnto` is able to override the "actual" priority of a process or thread by changing its' "effective" priority to a value calculated by the scheduler, using the "actual" priority as a calculation parameter.

FMT_SMF.1 Specification of Management Functions

Hierarchical to: No other components.

FMT_SMF.1.1

The TSF shall be capable of performing the following security management functions: [*configure a subject's effective priority; configure a subject's Adaptive Partitioning assignments; create a process or thread; kill a process or thread*].

Dependencies: No Dependencies

FMT_SMR.1 Security roles

Hierarchical to: No other components.

FMT_SMR.1.1

The TSF shall maintain the roles**:** [

- *Priority 0: idle thread*
- *Priorities 1 – 63: subjects without root privileges, or subjects with root privileges choosing to run at a lower priority*
- *Priorities 64 – 255: subjects with root privileges*].

FMT_SMR.1.2

The TSF shall be able to associate **processes and threads** ~~users~~ with roles.

Dependencies: FIA_UID.1 Timing of identification

5.1.4 Class FPT: Protection of the TSF

FPT_FLS.1 Failure with preservation of secure state

Hierarchical to: No other components.

FPT_FLS.1.1

The TSF shall preserve a secure state when the following types of failures occur: [*a failure of a thread in one process will not cause threads in another unrelated process to fail*].

Dependencies: ADV_SPM.1 Informal TOE security policy model

Application Note: FPT_FLS.1 refers to the fact that the TOE maintains separation of memory address spaces between unrelated processes. Thus, threads in two unrelated processes – that is, two processes that do not share a shared memory object – cannot corrupt the resources belonging to the other thread or otherwise interfere with the operation of the other thread.

FPT_RVM.1 Non-bypassability of the TSP

Hierarchical to: No other components.

FPT_RVM.1.1

The TSF shall ensure that TSP enforcement functions are invoked and succeed before each function within the TSC is allowed to proceed.

Dependencies: No dependencies

FPT_SEP.1 TSF domain separation

Hierarchical to: No other components.

FPT_SEP.1.1

The TSF shall maintain a security domain for its own execution that protects it from interference and tampering by untrusted subjects.

FPT_SEP.1.2

The TSF shall enforce separation between the security domains of subjects in the TSC.

Dependencies: No dependencies

5.1.5 Class FRU: Resource Utilization

FRU_FLT.1 Degraded fault tolerance

Hierarchical to: No other components.

FRU_FLT.1.1

The TSF shall ensure the operation of [*other unrelated processes and threads*] when the following failures occur: [*failure or other execution errors by a process or thread*].

Dependencies: FPT_FLS.1 Failure with preservation of secure state

Application Note: FRU_FLT.1 is closely related to FPT_FLS.1 above.

FRU_PRS.1 Limited priority of service

Hierarchical to: No other components.

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 [*CPU bandwidth*] shall be mediated on the basis of the subject's assigned priority **and the CPU time budget of its assigned Adaptive Partition**.

Dependencies: No dependencies

FRU_RSA.1 Maximum quotas

Hierarchical to: No other components.

FRU_RSA.1.1

The TSF shall enforce maximum quotas of the following resources: [*CPU bandwidth*] that [*subjects*] can use [*over a specified period of time*].

Dependencies: No dependencies

5.2 Security Functional Requirements on the IT Environment

There are no security requirements for the TOE's IT environment.

5.3 Assurance Requirements

This section defines the assurance requirements for the TOE. Assurance requirements are taken from the CC Part 3 and are EAL4 augmented with ALC_FLR.1. Table 9 – Assurance Requirements summarizes the requirements.

Table 9 – Assurance Requirements

Assurance Requirements	
Class ACM: Configuration management	ACM_AUT.1 Partial CM automation
	ACM_CAP.4 General support and acceptance procedures
	ACM_SCP.2 Problem tracking CM coverage
Class ADO: Delivery and operation	ADO_DEL.2 Detection of modification
	ADO_IGS.1 Installation, generation, and start-up procedures
Class ADV: Development	ADV_FSP.2 Fully defined external interfaces
	ADV_HLD.2 Security-enforcing high-level design
	ADV_IMP.1 Subset of the implementation of the TSF
	ADV_LLD.1 Descriptive low-level design
	ADV_RCR.1 Informal correspondence demonstration
	ADV_SPM.1 Informal TOE security policy model
Class AGD: Guidance documents	AGD_ADM.1 Administrator guidance
	AGD_USR.1 User guidance
Class ALC: Life cycle support	ALC_DVS.1 Development security
	ALC_FLR.1 Basic flaw remediation
	ALC_LCD.1 Developer defined Life cycle model
	ALC_TAT.1 Well-defined development tools

Assurance Requirements	
Class ATE: Tests	ATE_COV.2 Analysis of coverage
	ATE_DPT.1 Testing: high-level design
	ATE_FUN.1 Functional testing
	ATE_IND.2 Independent testing – sample
Class AVA: Vulnerability assessment	AVA_MSU.2 Validation of analysis
	AVA_SOF.1 Strength of TOE security function evaluation
	AVA_VLA.2 Independent vulnerability analysis

6 TOE Summary Specification

This section presents information to detail how the TOE meets the functional and assurance requirements described in previous sections of this ST.

6.1 TOE Security Functions

Each of the Security Functional Requirements (SFRs) and the associated descriptions correspond to the security functions. Hence, each function is described by how it specifically satisfies each of its related requirements. This serves to both describe the security functions and rationalize that the security functions satisfy the necessary requirements.

Table 10 – Mapping of TOE Security Functions to Security Functional Requirements

TOE Security Function	SFR ID	Description
User Data Protection	FDP_ACC.1	Subset access control
	FDP_ACF.1	Security attribute based access control
	FDP_IFC.1	Subset information flow control
	FDP_IFF.1	Simple security attributes
	FDP_RIP.2	Full residual information protection
Identification	FIA_ATD.1	User attribute definition
	FIA_UID.2	User identification before any action
Security Management	FMT_MOF.1	Management of security functions behaviour
	FMT_MSA.1(a)	Management of security attributes
	FMT_MSA.1(b)	Management of security attributes
	FMT_MSA.1(c)	Management of security attributes
	FMT_MSA.3	Static attribute initialisation
	FMT_SMF.1	Specification of management functions
	FMT_SMR.1	Security roles
Protection of TSF	FPT_FLS.1	Failure with preservation of secure state
	FPT_RVM.1	Non-bypassability of the TSP
	FPT_SEP.1	TSF domain separation
Resource Utilization	FRU_FLT.1	Degraded fault tolerance
	FRU_PRS.1	Limited priority of service
	FRU_RSA.1	Maximum quotas

6.1.1 User Data Protection

The User Data Protection TSF is primarily concerned with ensuring that processes and threads can only access the resources for which they have the appropriate permissions or attributes.

When built with the CC-approved configuration[4], the TOE overwrites the contents of a resource stored in volatile memory (such as RAM[5]) before re-allocating that memory space to another resource. This ensures that the previous information content is not available to other processes or threads.

The TOE enforces the following Security Functional Policies (SFPs):

6.1.1.1 Access Control SFP

The Access Control SFP is concerned with mediating access to CPU time, and supports the FRU_RSA.1 "Maximum quotas" SFR. The Access Control SFP can be generally stated in the following manner: "*If a thread currently has the highest effective priority of all threads, as determined by the Scheduler, then grant the thread access to CPU time, otherwise deny the access.*" The Access Control SFP is applied to processes (and their associated threads) and CPU time based on the attributes listed in Table 7 above.

6.1.1.2 Information Flow Control SFP

The Information Flow Control SFP can be generally stated in the following manner: "*A thread may access a shared memory object if the thread's user ID or group ID has the appropriate permissions. When a thread creates a shared memory object, the thread's usermask is used to set the initial permission bits of the newly-created object.*" The Information Flow Control SFP is applied to processes (and their associated threads) based on the permission bits of the shared memory and the process or thread's user mask, user ID, and group ID.

A shared memory object is a range of memory which is allocated for simultaneous access by more than one process. Shared memory objects are accessed via a corresponding symbol in the pathname space hierarchy at `/shmem`, which is implemented by the shared memory resource manager contained within the TOE (other "filesystems" are implemented by resource managers which are outside of the TOE). Each shared memory object has permission bits which indicate what process user masks are allowed accesses. When a thread attempts to access a shared memory object, *procnto* compares the thread's user ID (and the group ID, if required) to the permission bits of the object and determines whether to allow or deny the access.

Threads may create symbols in the pathname space hierarchy (corresponding to shared memory objects) and apply access restrictions to them in the form of permission bits. Any thread may create a new symbol in the "root" of the pathname space at `/shmem`, so long as `/shmem` is not being managed by another thread or process. A thread may create a symbol under a pre-existing symbol (a "branch") of the pathname space if the branch is owned by that thread or process, or if the permission bits on that branch (including the pathname space root, if it is being managed by another thread or process) allow symbol creation to the thread. Attempts by threads to create new shared memory objects are governed by *procnto*, which determines whether or not the thread's user ID (and the group ID, if required) is authorized to create new shared memory objects in the requested branch of the pathname space. Threads running as "root" are always allowed to perform any requested action.[6]

TOE Security Functional Requirements Satisfied: FDP_ACC.1, FDP_ACF.1, FDP_IFC.1, FDP_IFF.1, FDP_RIP.2.

[4] For details, see the Common Criteria Supplement Guide.
[5] RAM: Random Access Memory
[6] A process or thread is considered to be running as root if it has an effective UID of "0" and an effective GID of "0".

6.1.2 Identification

The "users" of the TOE are processes and threads. Processes are "containers" for one or more threads. Each process has an associated user mask, User ID, Group ID, actual priority, effective priority, and list of associated Adaptive Partitioning assignments[7], and these attributes are inherited by the process' threads. The User ID indicates the user that "owns" (is responsible for) that process, and associates the permissions assigned to that user with that process. The Group ID similarly indicates the user group that the process owner belongs to, and associates the related group permissions with that process. The user mask determines the default permissions that will be set on any object that the process creates in the pathname space. The actual priority indicates the priority that the process or thread itself (or another thread running with root privileges) wishes itself to run with. The effective priority indicates the current calculated priority that is being used by the `procnto` scheduler to schedule the process for access to CPU bandwidth; it uses the actual priority as a parameter when calculating this value. The list of Adaptive Partitioning assignments determines indicates which AP partitions the process is assigned to, and is used by the scheduler when making scheduling decisions.

Each thread inherits all of these attributes from its parent process. Process and threads have other attributes which are defined in struct `thread_entry` and struct `process_entry`. The attributes listed in FIA_ATD.1 are the attributes specifically used to implement the TOE functionality described in this Security Target. The unique identifier for a process is its process ID; the unique identifier for a thread is the combination of its process ID and its thread ID. All attributes are assigned at the moment that the process or thread is created – it is impossible to have a process or thread without an ID. This ensures that no TSF-mediated actions are performed for a process or thread before it is identified.

TOE Security Functional Requirements Satisfied: FIA_ATD.1, FIA_UID.2.

6.1.3 Security Management

The TOE provides four security management functions:

- Create a process or thread
- Kill a process or thread
- Configure a process' or thread's effective priority
- Configure a process' Adaptive Partitioning assignments[8]

The TOE implements "roles" as process and thread priorities, where higher numbered priorities are guaranteed more access to resources than lower numbered priorities. Priority "0" is the idle thread – no other thread may have priority 0. Priorities 1 – 63 are threads without root privileges, or threads with root privileges that have currently chosen to run at a lower priority. Priorities 64 – 255 are threads with root privileges. Processes and threads actually have two priorities – an "actual" priority and an "effective" priority. The actual priority is a value that is set by the process or thread itself and indicates to the scheduler (an entity within `procnto`) "how important" the process or thread currently considers itself to be; the effective priority is a value calculated by the scheduler that indicates "how important" the scheduler currently considers that process or thread to be. The scheduler uses the actual priority as a parameter when calculating the value of the effective priority, and only the effective priority is used by the scheduler

[7] Note that processes and threads may be assigned to more than one Adaptive Partition.
[8] This includes management of CPU time budget. Assignment of an adaptive partition includes the assignment of an associated CPU bandwidth limit (see Terminology). The CPU bandwidth limit dictates the CPU time budget. In other words, CPU bandwidth limit is an attribute of Adaptive Partitioning assignments.

to make scheduling decisions. Threads may change their own actual priority at any time, and they may adjust the actual priority of another thread if they have root privileges. The scheduler function of `procnto` can "override" the actual priority of a thread by adjusting its effective priority for scheduling purposes.

Any process or thread may create a new process or thread, which will inherit the applicable security attributes of the process that created it. Permission to kill a process or thread is limited to the process or thread itself, other threads contained in the process that spawned the thread, and to threads with root privileges. The ability to modify the actual priority of a process or thread is limited to the process or thread itself and to other threads running with root privileges. The ability to modify the effective priority of a process or thread is limited to the scheduler.

The TOE manages the SFPs discussed in Section 6.1.1 above by restricting the ability to modify the subject security attributes that can be modified to no TOE role, the process or thread itself, or another process or thread with root privileges, as appropriate:

- The actual priority can be modified by the process or thread itself and by other threads running with root privileges.
- The list of Adaptive Partitioning assignments and the effective priority cannot be modified by any TOE role. Note that, although the effective priority can be modified by the scheduler function of `procnto`, such modification is performed as part of the normal functioning of the TOE and cannot be directly manipulated by any TOE role.
- The other attributes specified in FIA_ATD.1 (user mask, User ID, Group ID) are typically set and managed outside of the TOE, although they can also be modified by processes or threads running with root privileges.

The TOE also manages the SFPs by providing permissive default values for the security attributes that are used to enforce the SFPs – the security attributes defined in FIA_ATD.1 are specified by the process or thread itself when it is created. The scheduler can override the default effective priority value and specify an alternative initial value.

TOE Security Functional Requirements Satisfied: FMT_MOF.1, FMT_MSA.1(a), FMT_MSA.1 (b), FMT_MSA.1(c), FMT_MSA.3, FMT_SMF.1, FMT_SMR.1.

6.1.4 Protection of the TSF

The TOE maintains its own protected memory space (devoted to the kernel), which is kept separate from the memory spaces allocated to the threads it executes. This separation allows the TOE to protect itself from tampering and interference by the threads it executes, since the TOE does not allow the threads it executes to access or modify its protected memory unless they are running as root.[9] The TOE also maintains the separation of the memory spaces of each thread it executes.

The TOE cannot execute a thread unless all of `procnto`'s functions (such as the scheduler, memory manager, etc.) are functioning properly – this ensures that the TOE Security Policy (TSP) enforcement functions are invoked and succeed before threads are executed.

The TOE's scheduling and separation functions ensure that failures in executed threads will not cause a failure in the TOE itself – all TOE functions will continue normally.

TOE Security Functional Requirements Satisfied: FPT_FLS.1, FPT_RVM.1, FPT_SEP.1.

[9] Care must be taken to ensure that threads entrusted with root privileges do not maliciously overwrite kernel memory.

6.1.5 Resource Utilization

The TOE's scheduling and separation functions ensure that failure of a thread in one process – whether due to poor design or malicious behavior – will not directly cause threads in another process to fail.[10]

Each thread has a priority and an "effective" priority: the priority is the "actual" priority (that is, the priority that the thread "would like to have"), while the "effective" priority is changed dynamically by the scheduler in order to implement quotas as defined by FRU_RSA.1. CPU time is allocated to each thread based on their effective priorities – threads with higher priorities are given "more" access to CPU time then threads with lower priority; however, higher priority threads cannot "starve" lower priority threads within a different AP partition – CPU time allocation is determined by the scheduler, which uses a thread's priority as well as the thread's AP partition CPU time budget as variables when determining the allocation schedule. Threads are allocated CPU time by the scheduler. For a detailed explanation of the scheduler and Adaptive Partitioning, please refer to the official QNX manuals.

Quotas are enforced on the total amount of CPU bandwidth that any particular AP can use. CPU bandwidth refers to the available portion of the CPU available to the AP at a given time. Quotas do not reflect total time or number of CPU cycles that a process or thread uses.

TOE Security Functional Requirements Satisfied: FRU_FLT.1, FRU_PRS.1, FRU_RSA.1.

6.2 TOE Security Assurance Measures

EAL4+ was chosen to provide a basic level of independently assured security. This section of the Security Target maps the assurance requirements of the TOE for a CC EAL4+ level of assurance to the assurance measures used for the development and maintenance of the TOE. The following table provides a mapping of the appropriate documentation to the TOE assurance requirements.

Note to Evaluator: The final versions of these documents have not yet been produced. The version numbers will be completed when the evaluation is close to completion and the documents have been finalized.

Table 11 - Assurance Measures Mapping to TOE Security Assurance Requirements (SARs)

Assurance Component	Assurance Measure
ACM_AUT.1	QNX Software Systems QNX Neutrino® Secure Kernel v6.4 Configuration Management
ACM_CAP.4	
ACM_SCP.2	
ADO_DEL.2	QNX Software Systems QNX Neutrino® Secure Kernel v6.4 Secure Delivery

[10] It is possible that a thread in a separate process which relies on actions performed by the failing thread could fail due to poor coding practices. For example, the second thread might assume that the first thread will never fail, and could then itself fail due to lack of error checking. However, consideration of these types of issues is beyond the scope of this CC certification. Care should always be taken to use good coding practices.

Assurance Component	Assurance Measure
ADO_IGS.1	QNX Neutrino® Secure Kernel v6.4 Installation Guidance
ADV_FSP.2	QNX Software Systems QNX Neutrino® Secure Kernel v6.4 TOE Architecture: Functional Specification, High Level Design, Low Level Design, and Representation Correspondence
ADV_HLD.2	
ADV_LLD.1	
ADV_RCR.1	
ADV_IMP.1	QNX Software Systems QNX Neutrino® Secure Kernel v6.4 - Implementation Representation
ADV_SPM.1	QNX Software Systems QNX Neutrino® Secure Kernel v6.4 Informal Security Policy Model
AGD_ADM.1	QNX Software Systems QNX Neutrino® Secure Kernel v6.4 Common Criteria Supplement Guide
AGD_USR.1	
ALC_DVS.1	QNX Software Systems QNX Neutrino® Secure Kernel v6.4 Life Cycle Support
ALC_FLR.1[11]	
ALC_LCD.1	
ALC_TAT.1	
ATE_COV.2	QNX Software Systems QNX Neutrino® Secure Kernel v6.4 Functional Tests, Coverage, and Depth Analysis
ATE_DPT.1	
ATE_FUN.1	
ATE_IND.2	[Performed by testing laboratory]
AVA_MSU.2	QNX Software Systems QNX Neutrino® Secure Kernel v6.4 Vulnerability Analysis
AVA_SOF.1	

[11] Augmentation to EAL 4+ assurance level.

Assurance Component	Assurance Measure
AVA_VLA.2	

7 Protection Profile Claims

There are no protection profile claims for this security target.

8 Rationale

This section provides the rationale for the selection of the security requirements, objectives, assumptions, and threats. In particular, it shows that the security requirements are suitable to meet the security objectives, which in turn are shown to be suitable to cover all aspects of the TOE security environment.

Table 12 below provides a mapping of assumptions, threats, and security functional requirements to the objectives for the TOE and TOE Environment, showing that the mapping is complete.

Table 12 - Relationship of Security Threats to Objectives

		O.ACCESS	O.EXECUTION_PRIORITY	O. FAILURE_ISOLATION	O.RESIDUAL_INFORMATION	O.RESOURCE_ALLOCATION	O.SUBJECT_ISOLATION	OE.INSTALL	OE.ADMIN_GUIDANCE	OE.INSTALL_GUIDANCE	OE.MANAGE	OE.PHYSICAL	OE.TRUSTED_INDIVIDUAL
Threats	T.DENIAL_OF_SERVICE		✓	✓		✓							
	T.INSTALL							✓	✓	✓	✓		
	T.UNAUTHORIZED_ACCESS	✓			✓		✓					✓	
Assumptions	A.MANAGE							✓			✓		
	A.NOEVIL							✓			✓		
	A.PHYSICAL											✓	
	A.TRUSTED_INDIVIDUAL												✓
Security Functional Requirements	FDP_ACC.1	✓											
	FDP_ACF.1	✓											
	FDP_IFC.1						✓						
	FDP_IFF.1	✓					✓						
	FDP_RIP.2				✓								
	FIA_ATD.1	✓	✓										
	FIA_UID.2	✓											
	FMT_MOF.1	✓											
	FMT_MSA.1(a)	✓	✓										

	O.ACCESS	O.EXECUTION_PRIORITY	O. FAILURE_ISOLATION	O.RESIDUAL_INFORMATION	O.RESOURCE_ALLOCATION	O.SUBJECT_ISOLATION	OE.INSTALL	OE.ADMIN_GUIDANCE	OE.INSTALL_GUIDANCE	OE.MANAGE	OE.PHYSICAL	OE.TRUSTED_INDIVIDUAL
FMT_MSA.1(b)	✓	✓										
FMT_MSA.1(c)	✓	✓										
FMT_MSA.3	✓	✓										
FMT_SMF.1	✓	✓										
FMT_SMR.1		✓										
FPT_FLS.1			✓									
FPT_RVM.1	✓		✓									
FPT_SEP.1						✓						
FRU_FLT.1			✓			✓						
FRU_PRS.1		✓										
FRU_RSA.1					✓							

8.1 Security Objectives Rationale

This section provides a rationale for the existence of each assumption and threat that compose the Security Target.

8.1.1 Security Objectives Rationale Relating to Threats

Table 13 - Threats:Objectives Mapping

Threats	Objectives	Rationale
T.DENIAL_OF_SERVICE A misbehaving process or thread may block others from system resources (i.e., processing time) via a resource exhaustion attack.	O.EXECUTION_PRIORITY The TOE will provide mechanisms that ensure that processes and threads with higher priorities and higher Adaptive Partitioning budgets are given more access to CPU time than processes and threads of lower priorities or lower AP budgets.	O.EXECUTION_PRIORITY ensures that a process or thread with a higher priority or AP budget will be given more access to CPU time than a process or thread with a lower priority or AP budget.

Threats	Objectives	Rationale
	O.FAILURE_ISOLATION The TOE will prevent a failure of one process or thread from affecting other unrelated processes and threads.	O.FAILURE_ISOLATION contributes to mitigation of this threat by ensuring that a failure of one process or thread does not cause a denial of service for another unrelated process or thread.
	O.RESOURCE_ALLOCATION The TOE will provide mechanisms that enforce constraints on the allocation of resources.	O.RESOURCE_ALLOCATION contributes to mitigation of this threat by ensuring that the TOE allocates system resources to subjects according to the total amount of CPU time each subject is using simultaneously.
T.INSTALL An administrator may incorrectly install or configure the TOE, resulting in ineffective security mechanisms.	OE.ADMIN_GUIDANCE The TOE will provide administrators with the necessary information for secure management of the TOE.	OE.ADMIN_GUIDANCE ensures that the necessary information to securely manage the TOE be provided to administrators of the TOE.
	OE.INSTALL_GUIDANCE The TOE will be delivered with the appropriate installation guidance to establish and maintain TOE security.	OE.INSTALL_GUIDANCE ensures that the appropriate information to securely install and maintain TOE security be provided as part of the delivered TOE.
	OE.INSTALL Those responsible for the TOE must ensure that the TOE is delivered, installed, managed, and operated in a manner that prevents disclosure, modification, destruction, and other threats to the TOE that result from a deficiency in delivery or customer site security..	OE.INSTALL ensures that administrators will deliver, install, manage, and operate the TOE in a manner that prevents disclosure, modification, destruction, and other threats to the TOE that result from a deficiency in delivery or customer site security.
	OE.MANAGE Sites deploying the TOE will provide competent, non-hostile TOE administrators who are appropriately trained and follow all administrator guidance. TOE administrators will ensure the system is used securely.	OE.MANAGE ensures that administrators will follow all administrator guidance, are non-hostile, and will ensure that the system is used in a secure manner.
T.UNAUTHORIZED _ACCESS A process or thread may gain access to resources or TOE security management functions for which it is not authorized according to the TOE security policy.	O.ACCESS The TOE will ensure that processes and threads gain only authorized access to resources.	O.ACCESS ensures that processes and threads can gain access only to those resources for which they are authorized.
	O.RESIDUAL_INFORMATION The TOE will ensure that any information contained in a resource is not released to processes or threads when the resource is reallocated.	O.RESIDUAL_INFORMATION ensures that residual information contained in a resource, once disassociated from one process or thread, is not accessible when the resource is allocated to another process or thread.

Threats	Objectives	Rationale
	O.SUBJECT_ISOLATION The TOE will provide mechanisms to protect each process or thread from unauthorized interference by other processes or threads.	O.SUBJECT_ISOLATION ensures that a process or thread cannot interfere with another process or thread by accessing resources for which it is not authorized.
	OE.PHYSICAL Physical security will be provided for the TOE by the non- IT environment commensurate with the value of the IT assets protected by the TOE.	OE.PHYSICAL establishes physical controls that restrict physical access to the TOE to only authorized personnel.

8.1.2 Security Objectives Rationale Relating to Assumptions

Table 14 - Assumptions:Objectives Mapping

Assumptions	Objectives	Rationale
A.MANAGE There is one or more competent individuals (administrators) assigned to manage the TOE.	OE.INSTALL Those responsible for the TOE must ensure that the TOE is delivered, installed, managed, and operated in a manner that prevents disclosure, modification, destruction, and other threats to the TOE that result from a deficiency in delivery or customer site security.	OE.INSTALL ensures that the TOE will be installed correctly and configured securely.
	OE.MANAGE Sites deploying the TOE will provide competent, non-hostile TOE administrators who are appropriately trained and follow all administrator guidance. TOE administrators will ensure the system is used securely.	OE.MANAGE ensures that the TOE will be managed by competent, non-hostile administrators who will configure the system securely to limit access to the TOE's configuration data.
A.NOEVIL The people administering the TOE and writing processes and threads for execution by the TOE are non-hostile, appropriately trained, and follow all guidance.	OE.INSTALL Those responsible for the TOE must ensure that the TOE is delivered, installed, managed, and operated in a manner that prevents disclosure, modification, destruction, and other threats to the TOE that result from a deficiency in delivery or customer site security.	OE.INSTALL ensures that the TOE will be installed correctly and configured securely.
	OE.MANAGE Sites deploying the TOE will provide competent, non-hostile TOE administrators who are appropriately trained and follow all administrator guidance. TOE administrators will ensure the system is used securely.	OE.MANAGE ensures that the TOE will be managed by competent, non-hostile administrators who will configure the system securely to limit access to the TOE's configuration data.

Assumptions	Objectives	Rationale
A.PHYSICAL It is assumed that the non-IT environment provides the TOE with appropriate physical security commensurate with the value of the IT assets protected by the TOE.	OE.PHYSICAL Physical security will be provided for the TOE by the non- IT environment commensurate with the value of the IT assets protected by the TOE.	OE.PHYSICAL addresses this assumption by requiring the non-IT environment to provide physical security for the TOE that is commensurate with the value of the IT assets protected by the TOE.
A.TRUSTED_INDIVIDUAL It is assumed that any individual allowed to perform procedures upon which the security of the TOE may depend is trusted with assurance commensurate with the value of the IT assets.	OE.TRUSTED_INDIVIDUAL Any individual allowed to perform procedures upon which the security of the TOE may depend must be trusted with assurance commensurate with the value of the IT assets.	OE.TRUSTED_INDIVIDUAL addresses this assumption by requiring that any individual who is allowed to perform procedures that affect the security of the TOE be trusted with assurance commensurate with the value of the IT assets.

8.2 Security Functional Requirements Rationale

The following discussion provides detailed evidence of coverage for each security objective.

8.2.1 Rationale for Security Functional Requirements of the TOE Objectives

Table 15 - Objectives:SFRs Mapping

Objective	Requirements Addressing the Objective	Rationale
O.ACCESS The TOE will ensure that processes and threads gain only authorized access to resources.	FDP_ACF.1 Security attribute based access control	FDP_ACF.1 specifies the attributes used to enforce the access control SFP.
	FDP_IFF.1 Simple security attributes	FDP_IFF.1 specifies the information flow control SFP.
	FIA_ATD.1 User attribute definition	FIA_ATD.1 specifies the process and thread attributes which are used for enforcing the access control SFP.
	FIA_UID.2 User identification before any action	FIA_UID.2 specifies that the TOE may perform no actions for a process or thread before it is identified.
	FMT_MSA.3 Static attribute initialisation	FMT_MSA.3 specifies how the default security attributes for process and threads are determined and overridden.
	FDP_ACC.1 Subset access control	FDP_ACC.1 requires the TSF to enforce the access control SFP.

Objective	Requirements Addressing the Objective	Rationale
	FMT_SMF.1 Specification of management functions	FMT_SMF.1 requires the TSF to provide capability to configure a process' or thread's security attributes and to create or kill it.
	FMT_MOF.1 Management of security functions behavior	FMT_MOF.1 requires the TSF to restrict the capability to configure a process' or thread's security attributes and to create or kill it to other appropriate processes or threads.
	FPT_RVM.1 Non-bypassability of the TSP	FPT_RVM.1 requires that the security functions be non-bypassable.
	FMT_MSA.1(a), FMT_MSA.1(b), FMT_MSA.1(c) Management of security attributes	The iterations of FMT_MSA.1 require the TSF to restrict the modification of process or thread security attributes to authorized entities.
O.EXECUTION_PRIORITY The TOE will provide mechanisms that ensure that processes and threads with higher priorities and higher Adaptive Partitioning budgets are given more access to CPU time than processes and threads of lower priorities or lower AP budgets.	FMT_SMF.1 Specification of management functions	FMT_SMF.1 requires the TSF to provide capability to configure a process' or thread's priority.
	FIA_ATD.1 User attribute definition	FIA_ATD.1 specifies the process and thread attributes which are used to determine the priority of service that a process or thread should receive.
	FMT_MSA.1(a), FMT_MSA.1(b), FMT_MSA.1(c) Management of security attributes	The iterations of FMT_MSA.1 require the TSF to restrict the modification of process and thread security attributes to authorized entities.
	FRU_PRS.1 Limited priority of service	FRU_PRS.2 requires the TSF to use each thread's priority when determining how to utilize CPU time.
	FMT_MSA.3 Static attribute initialisation	FMT_MSA.3 specifies how the default security attributes for processes and threads are determined and overridden.
	FMT_SMR.1 Security roles	FMT_SMR.1 specifies the roles maintained by the TSF.
O.FAILURE_ISOLATION The TOE will prevent a failure of one process or thread from affecting other unrelated processes and threads.	FPT_FLS.1 Failure with preservation of secure state	FPT_FLS.1 requires the TSF to preserve a secure state when a process or thread encounters an error.
	FPT_RVM.1 Non-bypassability of the TSP	FPT_RVM.1 requires that the security functions be non-bypassable.

Objective	Requirements Addressing the Objective	Rationale
	FRU_FLT.1 Degraded fault tolerance	FRU_FLT.1 requires the TSF to prevent a failure of one process or thread from causing the TOE to stop rendering services to all other processes and threads.
O.RESIDUAL_INFORMATION The TOE will ensure that any information contained in a resource is not released to processes and threads when the resource is reallocated.	FDP_RIP.2 Full residual information protection	FDP_RIP.2 requires the TSF to destroy the contents of a volatile memory resource before reallocating the memory to another resource.
O.RESOURCE_ALLOCATION The TOE will provide mechanisms that enforce constraints on the allocation of resources.	FRU_RSA.1 Maximum quota	FRU_RSA.1 requires the TSF to enforce maximum quotas on resources.
O.SUBJECT_ISOLATION The TOE will provide mechanisms to protect each process or thread from unauthorized interference by other processes or threads.	FDP_IFC.1 Subset information flow control	FDP_IFC.1 requires the TSF to enforce the information flow control SFP.
	FDP_IFF.1 Simple security attributes	FDP_IFF.1 specifies the information flow control SFP.
	FPT_SEP.1 TSF domain separation	FPT_SEP.1 requires that the TOE protect its subjects (processes and threads) from interference and tampering by untrusted subjects.
	FRU_FLT.1 Degraded fault tolerance	FRU_FLT.1 requires the TSF to prevent a failure of one process or thread from causing the TOE to stop rendering services to all other processes and threads.

8.3 Security Assurance Requirements Rationale

EAL4+ was chosen to provide a basic level of independently assured security and thorough investigation of the TOE and its development. As such, minimal additional tasks are placed upon the vendor assuming the vendor follows reasonable software engineering practices and can provide support to the evaluation for design and testing efforts. The chosen assurance level is appropriate with the threats defined for the environment. The TOE is expected to be in a non-hostile position and embedded in or protected by other products designed to address threats that correspond with the intended environment. At EAL4+, the TOE will have incurred an independent vulnerability analysis to support its introduction into the non-hostile environment.

The augmentation of ALC_FLR.1 was chosen to give greater assurance of the developer's on-going flaw remediation processes.

8.4 Rationale for Refinements of Security Functional Requirements

The following refinements of Security Functional Requirements from CC version 2.3 have been made to clarify the content of the SFRs, and make them easier to read:

- FDP_ACF.1: Refined to clarify that there are no additional rules.
- FDP_IFF.1: Refined to clarify that there are no additional rules.
- FDP_RIP.2: Refined to say specify that only volatile memory is destroyed when de-allocated.
- FIA_ATD.1: Refined to replace "user" with "process or threads" since the TOE does not directly support human users.
- FIA_UID.2: Refined to replace "user" with "process or threads" since the TOE does not directly support human users. Replaced “identify itself” with “be identified” to indicate that the identification is passive rather than active.
- FMT_MSA.1(c): Refined to strike the mention of a specific SFP, since the restriction on modifying a thread’s actual priority is implemented by the design of the TOE rather than an arbitrary SFP.
- FMT_MSA.3: Refined to clarify that the effective priority of an object may not be overridden by any subject.
- FMT_SMR.1: Refined to clarify that priorities are the "roles"; also replaced "user" with "process or threads" since the TOE does not directly support human users.
- FRU_PRS.1: Refined to clarify that Adaptive Partition budgets are also used to mediate access.

8.5 Dependency Rationale

This ST does satisfy all the requirement dependencies of the Common Criteria. Table 16 lists each requirement to which the TOE claims conformance with a dependency and indicates whether the dependent requirement was included. As the table indicates, all dependencies have been met.

Table 16 - Functional Requirements Dependencies

SFR ID	Dependencies	Dependency Met?
FDP_ACC.1	FDP_ACF.1	✓
FDP_ACF.1	FDP_ACC.1	✓
	FMT_MSA.3	✓
FDP_IFC.1	FDP_IFF.1	✓
FDP_IFF.1	FDP_IFC.1	✓
	FMT_MSA.3	✓
FDP_RIP.2	none	✓
FIA_ATD.1	none	✓
FIA_UID.2	none	✓

SFR ID	Dependencies	Dependency Met?
FMT_MOF.1	FMT_SMF.1	✓
	FMT_SMR.1	✓
FMT_MSA.1(a) FMT_MSA.1(b) FMT_MSA.1(c)	FMT_SMR.1	✓
	FDP_ACC.1	✓
	FDP_IFC.1	✓
	FMT_SMF.1	✓
FMT_MSA.3	FMT_MSA.1	✓
	FMT_SMR.1	✓
FMT_SMF.1	none	✓
FMT_SMR.1	FIA_UID.1	✓ (by FIA_UID.2, which is hierarchical to FIA_UID.1)
FPT_FLS.1	ADV_SPM.1	✓
FPT_RVM.1	none	✓
FPT_SEP.1	none	✓
FRU_FLT.1	FPT_FLS.1	✓
FRU_PRS.1	none	✓
FRU_RSA.1	none	✓

8.6 TOE Summary Specification Rationale

8.6.1 TOE Summary Specification Rationale for the Security Functional Requirements

Each subsection in the TOE Summary Specification (Section 6) describes a security function of the TOE. Each description is organized by set of requirements with rationale that indicates how these requirements are satisfied by aspects of the corresponding security function. The set of security functions works to satisfy all of the security functions and assurance requirements. Furthermore, all of the security functions are necessary in order for the TSF to provide the required security functionality. This section, in conjunction with the TOE Summary Specification section, provides evidence that the security functions are suitable to fulfill the TOE security requirements.

Table 17 identifies the relationship between security requirements and security functions, showing that all security requirements are addressed and all security functions are necessary (i.e., they correspond to at least one security requirement).

Table 17 - Mapping of Security Functional Requirements to TOE Security Functions

TOE Security Function	SFR
User Data Protection	FDP_ACC.1
	FDP_ACF.1
	FDP_IFC.1
	FDP_IFF.1
	FDP_RIP.2
Identification	FIA_ATD.1
	FIA_UID.2
Security Management	FMT_MOF.1
	FMT_MSA.1(a), FMT_MSA.1(b), FMT_MSA.1(c)
	FMT_MSA.3
	FMT_SMF.1
	FMT_SMR.1
Protection of TOE Security Functions	FPT_FLS.1
	FPT_RVM.1
	FPT_SEP.1
Resource Utilization	FRU_FLT.1
	FRU_PRS.1
	FRU_RSA.1

8.6.2 TOE Summary Specification Rationale for the Security Assurance Requirements

EAL4+ was chosen to provide a basic level of independently assured security. The chosen assurance level is consistent with the postulated threat environment. The TOE is expected to operate in a non-hostile position and be embedded in or protected by other products designed to address threats that correspond with the intended environment.

8.6.2.1 Configuration Management

The Configuration Management documentation provides a description of tools used to control the configuration items and how they are used by QNX. The documentation provides a complete configuration item list and a unique reference for each item. Additionally, the configuration management system is described including procedures that are used by developers to control and track changes that are made to the TOE. The documentation further details the TOE configuration items that are controlled by the configuration management system.

Corresponding CC Assurance Components:

- Generation support and acceptance procedures
- Partial CM automation
- Problem tracking CM coverage

8.6.2.2 Secure Delivery and Operation

The Delivery and Operation documentation provides a description of the secure delivery procedures implemented by QNX to protect against TOE modification during product delivery. The Installation Documentation provided by QNX details the procedures for installing the TOE and placing the TOE in a secure state offering the same protection properties as the master copy of the TOE. The Installation Documentation provides guidance to the administrator on the TOE configuration parameters and how they affect the TSF.

Corresponding CC Assurance Components:

- Delivery Procedures
- Installation, Generation, and Start-Up Procedures

8.6.2.3 Development

The QNX design documentation consists of several related design documents that address the components of the TOE at different levels of abstraction. The following design documents address the Development Assurance Requirements:

- The Functional Specification provides a description of the security functions provided by the TOE and a description of the external interfaces to the TSF. The Functional Specification covers the purpose and method of use and a list of effects, exceptions, and errors message for each external TSF interface.
- The High-Level Design provides a top level design specification that refines the TSF functional specification into the major constituent parts (subsystems) of the TSF. The high-level design identifies the basic structure of the TSF, the major elements, a listing of all interfaces, and the purpose and method of use for each interface.
- The Low-Level Design describes each security supporting module in terms of its purpose and interaction with other modules. It describes the TSF in terms of modules, designating each module as either security-enforcing or security-supporting. It provides an algorithmic description for each security-enforcing module detailed enough to represent the TSF implementation.
- The Implementation Representation unambiguously defines the TSF to a level of detail such that the TSF can be generated without further design decisions. It also describes the relationships between all portions of the implementation.
- The Security Policy Model provides an informal TSP model and it demonstrates correspondence between the functional specification and the TSP model by showing that all of the security functions in the functional specification are consistent and complete with respect to the TSP model. The TSP model describes the rules and characteristics of all policies of the TSP that can be modeled. The model should include a rationale that demonstrates that it is consistent and complete with respect to all policies of the TSP that can be modeled.
- The Correspondence Analysis demonstrates the correspondence between each of the TSF representations provided. This mapping is performed to show the functions traced from the ST description to the High-Level Design.

Corresponding CC Assurance Components:

- Functional Specification with Complete Summary
- Security-Enforcing High-Level Design
- Descriptive Low-Level Design
- Implementation of the TSF
- Informal TOE Security Policy Model
- Informal Representation Correspondence

8.6.2.4 Guidance Documentation

The QNX Guidance documentation provides administrator and user guidance on how to securely operate the TOE. The Administrator Guidance provides descriptions of the security functions provided by the TOE. Additionally, it provides detailed accurate information on how to administer the TOE in a secure manner and how to effectively use the TSF privileges and protective functions. The User Guidance provided directs users on how to operate the TOE in a secure manner. Additionally, User Guidance explains the user-visible security functions and how they are to be used and explains the user's role in maintaining the TOE's Security. QNX provides single versions of documents which address the administrator Guidance and User Guidance; there are no separate guidance documents specifically for non-administrator users of the TOE.

Corresponding CC Assurance Components:

- Administrator Guidance
- User Guidance

8.6.2.5 Life Cycle Support Documents

The Life Cycle Support documentation describes all the physical, procedural, personnel, and other security measures that are necessary to protect the confidentiality and integrity of the TOE design and implementation in its development environment. It provides evidence that these security measures are followed during the development and maintenance of the TOE. It provides evidence that these security measures are followed during the development and maintenance of the TOE. The flaw remediation procedures addressed to the TOE developers are provided and so are the established procedures for accepting and acting upon all reports of security flaws and requests for corrections of those flaws. The flaw remediation guidance addressed to TOE users is provided. The description also contains the procedures used by QNX to track all reported security flaws in each release of the TOE. The established life-cycle model to be used in the development and maintenance of the TOE is documented and explanation on why the model is used is also documented. The selected implementation-dependent options of the development tools are described.

Corresponding CC Assurance Components:

- Identification of Development Security Measures
- Flaw Reporting Procedures
- Developer Defined Life Cycle Model
- Well-defined Development Tools

8.6.2.6 Tests

There are a number of components that make up the Test documentation. The Coverage Analysis demonstrates the testing performed against the functional specification. The Coverage Analysis demonstrates the correspondence between the tests identified in the test documentation and the TSF as described in the functional specification. The depth analysis demonstrates that the tests identified in the test documentation are sufficient to demonstrate that the TSF operates in accordance with its high-level design and low-level design. QNX Test Plans and Test Procedures, which detail the overall efforts of the testing effort and break down the specific steps taken by a tester, are also provided. The Independent Testing documentation provides an equivalent set of resources to those that were used in the developer's functional testing.

Corresponding CC Assurance Components:

- Analysis of Coverage
- High-Level Design
- Functional Testing
- Independent Testing

8.6.2.7 Vulnerability and TOE Strength of Function Analysis

The Strength of TOE Security Function Analysis demonstrates the strength of the probabilistic or permutational mechanisms employed to provide security functions within the TOE and how they exceed the minimum SOF requirements. The Independent Vulnerability Analysis documentation describes the analysis of the TOE deliverables performed to search for ways in which a user can violate the TSP, and the disposition of the identified vulnerabilities.

Corresponding CC Assurance Components:

- Strength of TOE Security Function Evaluation
- Independent Vulnerability Analysis

8.7 Strength of Function

No Strength of Function (SOF) claim is applicable for this TOE since there are no security functions or security functional requirements which have probabilistic or permutational functions.

9 Acronyms

Table 18 - Acronyms

Acronym	Definition
AP	Adaptive Partitioning
CC	The Common Criteria for Information Technology Security Evaluation
CPU	Central Processing Unit
ID	Identification; Identifier
IT	Information Technology
PP	Protection Profile
RAM	Random Access Memory
RTOS	Realtime Operating System
SFR	Security Functional Requirement
SMP	Symmetric Multiprocessing
SOF	Strength of Function
ST	Security Target
TOE	Target of Evaluation
TSF	TOE Security Function
TSP	TOE Security Policy
TSC	TOE Scope of Control