LynxSecure 6.3.0 Architecture Guide

LynxSecure Release 6.3.0-rev16326

Contents

List of Figures

List of Tables

Preface

About this Guide

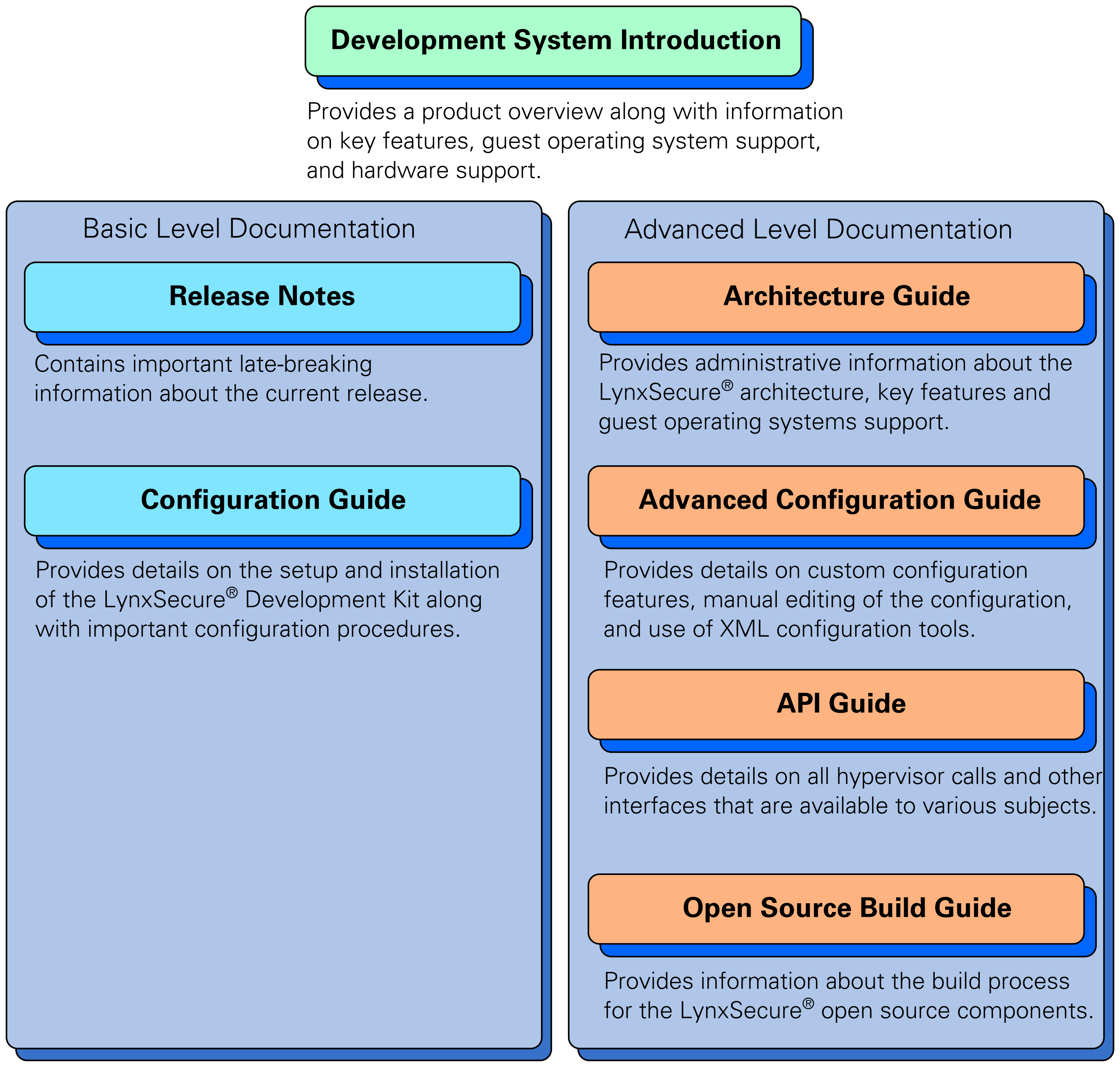
The *LynxSecure® Architecture Guide* describes the features of LynxSecure and how to use them. The types of the guest operating systems, referred to as subjects in LynxSecure, supported by the LynxSecure platform are described in detail.

Intended Audience

The information in this guide is designed and written for system integrators and software developers who will build applications on top of LynxSecure and the available subjects. A basic understanding of Linux®/Unix® and familiarity with fairly complex configuration procedures are recommended.

For More Information

For more information on the features of LynxSecure, refer to the following printed and online documentation.


Development System Introduction
Basic Level Documentation
Release Notes
Configuration Guide
Advanced Level Documentation
Architecture Guide
Advanced Configuration Guide
API Guide
Open Source Build Guide

Provides a product overview along with information on key features, guest operating system support, and hardware support.

Contains important late-breaking information about the current release.

Provides details on the setup and installation of the LynxSecure® Development Kit along with important configuration procedures.

Provides administrative information about the LynxSecure® architecture, key features and guest operating systems support.

Provides details on custom configuration features, manual editing of the configuration, and use of XML configuration tools.

Provides details on all hypervisor calls and other interfaces that are available to various subjects.

Provides information about the build process for the LynxSecure® open source components.

Typographical Conventions

The typefaces used in this manual, summarized below, emphasize important concepts. All references to file names and commands are case sensitive and should be typed accurately.

Font and Description	Examples
Times New Roman 10 pt. - Used for body text; *italicized* for emphasis, new terms, and book titles.	Refer to the *LynxSecure User's Guide*

Courier New 9 pt. - Used for environment variables, file names, functions, methods, options, parameter names, path names, commands, and computer data. Commands that need to be	`ls -l`
	`myprog.c`
	`/dev/null`
	`login:` **`myname`**

Font and Description	Examples
highlighted within body text, or commands that must be typed as-is by the user are **bolded**.	`# `**`cd /usr/home`**

Courier New Italic 9 pt. - Used for text that represents a variable, such as a file name or a value that must be entered by the user.	`cat` *`filename`*
	`mv` *`file1 file2`*

Courier New 7 pt. - Used for blocks of text that appear on the display screen after entering instructions or commands.	`Kernel: target1.srp`
	`> Loading: target1.srp`
	`> .......................`
	`> Booting`

Univers 45 Light Bold 8 pt. - keyboard options, button names, and menu sequences.	Enter, Ctrl-C

Special Notes

The following notations highlight any key points and cautionary notes that may appear in this manual.

NOTE: These callouts note important or useful points in the text.

CAUTION! Used for situations that present minor hazards that may interfere with or threaten equipment/performance.

Technical Support

Lynx Software Technologies, Inc. Technical Support is available Monday through Friday (excluding holidays) between 8:00 AM and 5:00 PM Pacific Time (U.S. Headquarters) or between 9:00 AM and 6:00 PM Central European Time (Europe).

The Lynx Software Technologies, Inc. World Wide Web home page [http://www.lynx.com] provides additional information about our products.

Lynx Software Technologies, Inc. U.S. Headquarters

Internet: `<support@lynx.com>`
Phone: (408) 979-3940
Fax: (408) 979-3920

Lynx Software Technologies, Inc. Europe

Internet: `<tech_europe@lynx.com>`
Phone: (+33) 1 30 85 06 00
Fax: (+33) 1 30 85 06 06

CHAPTER 1 *Overview*

LynxSecure® is a Separation Kernel Hypervisor (SKH), which allows multiple subjects to run concurrently on the same hardware, while strictly enforcing what resources each subject has access to, and when.

High-Level View of LynxSecure

This section provides a high level look at a LynxSecure system and some of the key components. Detailed explanation of each component will be given in their respective chapters.

LynxSecure addresses several key issues facing system integrators: Limited power, space, and weight budgets, System Security, Time and Space separation, Performance, Reliability and Certifiability.

In many modern environments, physical resources such as power, space, weight and heat are critically limited. The more functions a single piece of hardware can perform, the further that these limited resources can be stretched.

In security critical applications, it can be vital that one component be prevented from accessing the resources of another component. Separate computers can provide isolation, at the expense of space, weight, money, power, complexity and flexibility.

Many virtualization technologies operate at a significant cost to performance. LynxSecure provides mechanisms to minimize the performance impact. By reducing the number of hardware systems required, LynxSecure can reduce the number of potential points of failure. Some of the resources saved by consolidating hardware could be devoted to redundant systems thereby increasing reliability.

In LynxSecure, a virtualized Guest OS is known as a subject. LynxSecure exists as a layer between each subject and the underlying hardware, controlling subject access to system resources.

LynxSecure supports both paravirtualized subjects and fully virtualized subjects. Paravirtualization is the modification of a given subject to run on LynxSecure. Full virtualization makes it possible to run an unmodified operating system subject on LynxSecure.

Figure 1-1: LynxSecure High Level View

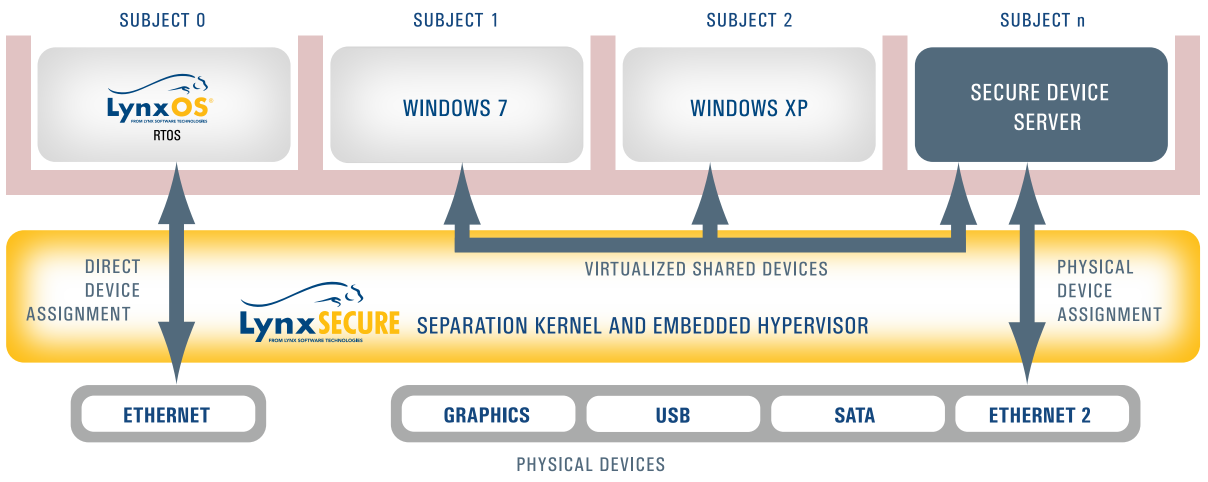


Figure 1-1 (page 2) presents one of the many deployment options LynxSecure and the subjects provide.

LynxSecure plays a critical role in controlling the information flow and access to resources.

LynxSecure supports Symmetric Multiprocessing (SMP), and runs as a multicore Separation Kernel Hypervisor (SKH). The cores utilized by LynxSecure are configurable via XML.

LynxSecure supports paravirtualized subjects. These subjects are partially modified to run on LynxSecure by conforming to the LynxSecure Application Programming Interface (API).

LynxSecure supports fully virtualized subjects. These subjects can run unmodified on LynxSecure, or with user space tuning. In general, these subjects include closed source OSes and kernels.

LynxSecure supports direct device assignment. Subjects can be configured using the XML file to have direct access to a given set of devices and host controllers.

LynxSecure supports virtual devices, such as virtual hard disk drive or virtual networking that fully virtualized subjects can use instead of directly assigned devices.

LynxSecure supports a lightweight subject called the LynxSecure Application (LSA). The LSA can be thought of as a minimal kernel context, providing a framework for many different kinds of kernel applications. The LSA is supported as either a 32-bit paravirtualized subject, 64-bit paravirtualized subject, or a 32-bit fully virtualized subject. LSA supports SMP.

LynxSecure provides enhanced ARINC-653 scheduling for deterministic time partitioning of resources. This is configured using the XML file.

Within the ARINC-653 scheduling framework, LynxSecure also supports an enhanced scheduling mode called Flexible Scheduling. This allows for a set of pre-defined subjects to share their given (static) portions of their execution time to each other, but not to other subjects; to allow for a more responsive client/server paradigm.

LynxSecure provides a framework for configurable security policy enforcement between subjects. The XML file specifies which subjects can communicate with other subjects, and systems events which can trigger and deliver audit messages to designated subjects.

LynxSecure supports Inter-VM Communication via message channels, synthetic interrupts, and shared memory.

Configuration of LynxSecure is performed using a configuration tool taking a human readable input (XML file format) to produce bootable LynxSecure images.

LynxSecure can utilize an IOMMU to provide additional protection.

CHAPTER 2 *LynxSecure Architecture*

A running LynxSecure® Separation Kernel (LSK) system consists of the booted Separation Kernel Runtime Package (SRP) and the target system.

The SRP consists of the following elements:

- The Runtime Initialization Function (RIF) binary
- The LynxSecure SKH binary
- The Binary Configuration Vector (BCV)
- The subject resources, including kernels and ramdisk images

Lynx Software Technologies, Inc. provides a development system consisting of these components. The System Integrator uses the provided tools and configures an HCV (Human Readable Configuration Vector), to produce the SRP.

For system integrators and applications developers, the primary items of interest are as follows:

- Separation Kernel Hypervisor (SKH)
- Subjects
- Separation Kernel Runtime Package (SRP)
- Boot loader

LynxSecure Configuration Tool

The configuration tool converts a human-readable configuration vector to a machine readable form. The configuration tool takes an XML configuration vector together with a set of object code modules and outputs the SKH runtime package, or SRP. For complete details on the configuration tool, refer to Chapter 4, "*Introduction to the Autoconfig Tool*" in *LynxSecure 6.3.0 Getting Started and Configuration Guide*.

Separation Kernel Hypervisor (SKH)

The Separation Kernel Hypervisor is a kernel that is both a hypervisor and a separation kernel.

The hypervisor functionality maintains an abstraction layer between the hardware resources of the target system and the software hosted by the SKH.

The separation kernel functionality partitions and isolates the software and hardware resources from each other.

At runtime, the SKH arbitrates the actions of subjects on the underlying system resources.

A Brief Description of a Subject

A subject is a collection of resources exported by the SKH, such as physical memory regions and CPU runtime, that executes its own code (for example, code of an accompanying OS) within a protective framework provided by the SKH.

In general, a subject may be thought of as the collection of resources accompanying a piece of software (like an OS) that allow it to be executed and monitored by the SKH. One of the resources of the subject is a container of that subject's software (for example, the code and data segment of the kernel of the associated OS).

The SKH arbitrates the actions of subjects based upon data supplied by the systems integrator during the configuration process. The configuration data is available to the SKH at boot time via the SRP. See the Chapter 2, "*Configuration Vector Organization*" in *LynxSecure 6.3.0 Advanced Configuration Guide*. The arbitration by the SKH of the actions of subjects on the underlying system resources includes the arbitration of CPU access time, CPU feature access, memory access, device I/O resource access, chipset feature access, firmware access, and access to the Application Binary Interface (ABI) into the SKH itself.

If a subject attempts to access a memory region for which it is not configured to have access, the SKH will block the access. The subject may perceive this as a fault (example, page fault or general protection fault) depending on the internal state of the subject's execution context.

NOTE: If a subject is configured with directly assigned devices, subject may access other memory regions using DMA capabilities of those devices. To prevent such unauthorized access, the target must have IOMMU hardware and the IOMMU option must be turned on in the configuration.

If a subject attempts to access a device (via either device memory, or an I/O port) for which it is not configured to have access, the SKH will likewise block the access. Again, the subject is responsible for handling any faults associated with the blocked access.

If a subject attempts to use a CPU feature for which it is not configured to have access, the SKH will block this access.

If a subject attempts to access a BIOS region for which it is not configured to have access, the SKH will block this access.

If a subject attempts to use the ABI into the SKH in a manner for which it is not configured, the SKH will block this access.

The ABI into the SKH is referred to as the hypercall interface. A brief description of the ABI provided by the SKH is as follows:

- A subject may make hypercalls to obtain or set the current time
- A subject may make hypercalls to send/receive messages with other subjects
- A subject may make hypercalls to perform administrative actions on other subjects

For a detailed description of available API and ABI interfaces from a subject to the hypervisor, please refer to *LynxSecure 6.3.0 API Guide*.

In general, when a subject is said to have access to a given exported resource, the subject is said to have a "flow" to the exported resource. These flows are established by the XML configuration system.

The SKH arbitrates several types of flows:

- Memory region flow - specifies physical memory region access for a subject
- Device flow - specifies device resource access for a subject
- Message flow - specifies message buffer access for a subject
- Subject flow - specifies actions a subject is allowed to perform on another subject

To see all the types of flows available to be configured for a subject, refer to the Chapter 4, "*XML Reference*" in *LynxSecure 6.3.0 Advanced Configuration Guide*.

The SKH is launched by the bootloader. Once the SKH has initialized itself, it will launch its subjects on the various CPUs as per the configuration specified in the XML file.

Subjects

A subject is a collection of resources exported by the LynxSecure Separation Kernel Hypervisor, and it is an entity with its own code executing within the protective framework of the LynxSecure Separation Kernel Hypervisor.

Subjects are launched and scheduled by the LynxSecure Separation Kernel Hypervisor, the LynxSecure Separation Kernel Hypervisor provides a platform emulation interface (i.e. virtual motherboard) to FV subjects or a Hypercall Interface to PV subjects.

Strictly speaking, a subject is not a virtual machine. However, portions of a subject may be thought of as being scheduled onto a virtual machine (or set of virtual machines) associated with the subject. However, the notion of a virtual machine is transparent to the system configurator or developer.

Subjects are usually a collection of the following resources:

- Physical memory regions;
- Virtual memory descriptors (associations of virtual addresses with physical memory regions);
- I/O device memory regions, I/O port space;
- CPU Core access and execution time.

Given that a subject is not only a collection of exported resources, but one that also executes (by definition), each subject must have at least one exported resource which is a physical memory region. This physical memory region may be used by the subject to hold code and data.

Given that a subject will execute its code, it must have scheduling time on a given set of CPUs. The LynxSecure Separation Kernel Hypervisor arbitrates the scheduling of subjects on the active CPUs of the target system. This arbitration is based upon scheduling data provided in the XML configuration file.

LynxSecure supports several types of subjects.

Fully Virtualized (FV) Subjects

With full virtualization, an operating system that is designed to run on industry standard hardware, can run in a virtual machine under LynxSecure without any modification of either the operating system or the applications running on it.

Fully virtualized subjects simulate the bare hardware platform behavior. As it happens on bare hardware, when a FV subject is started, control is passed to the virtual firmware running within the subject. The virtual firmware is responsible for booting the GuestOS from available media (virtual or physical).

Para-Virtualized (PV) Subjects

Para-virtualized subjects run operating systems and other software that has been created or modified specifically to run under LynxSecure. LynxSecure provides a PV Linux® Kernel and tools to create a PV Linux subject as part of the product.

Para-virtualized subjects' start-up sequence differs from that of FV subjects. Typically, a LynxSecure SRP includes the PV GuestOS kernel or other PV software image; it is loaded into subject memory automatically when the subject is started without any intermediate bootloader. Control is then passed to that image and the rest of the start-up process is determined by the software contained in it.

If the target system has no I/O MMU, there are limitations on assigning DMA-capable physical devices to subjects. These limitations are less severe for PV subjects than FV subjects, and therefore PV could be the subject type of choice for such systems.

LynxSecure Application (LSA)

LynxSecure separation kernel provides an environment to run applications that do not use any operating system. A LynxSecure Application (LSA) is a subject which is completely created by a developer. It can be considered as a program that runs directly on the system hardware interface provided by LynxSecure. In this sense, the LSA also works as a monitor or supervisor and shall handle such events as interrupts and exceptions.

There are many reasons to develop a LynxSecure Application. The most common is the need to develop an extremely lightweight subject, that needs and uses minimal resources. Another potential use of an LSA is porting a monitor or debugger developed for another platform to LynxSecure.

The LynxSecure distribution includes an LSA sample which can be used as a starting point for developing a real LSA subject. The LSA subject may either be run as a paravirtualized subject or as a fully virtualized subject. When the LSA runs as a paravirtualized subject, it uses the LynxSecure Hypercall interface to communicate with the LynxSecure SKH. The sample LSA demonstrates some of the Hypercall API and LynxSecure features.

Please refer to the *LynxSecure 6.3.0 API Guide* for full descriptions of the LynxSecure API calls, features and build procedures.

Built-in Test (BIT) Subjects

In order to make sure that a LynxSecure system is running in a secure state LynxSecure has a suite of Built-in Tests. These tests fall into two categories, the Initiated Built-In Test (IBIT) and the Continuous Built-In Test (CBIT). The IBIT tests are run at startup, the optional CBIT tests are continuously cycled through in the background on each processor once the system starts normal operation. If one of those tests detects an error, the system will switch to maintenance mode and a more in depth set of tests may be performed. Certain critical subjects can be configured to continue running in maintenance mode, but since a failed test can indicate a compromised system, all non-essential functionality should be halted until the errors can be corrected either automatically or through manual intervention.

IBIT

The Initiated Built-In Test is run at startup. IBIT subjects must be run from start to finish on all cores simultaneously and without pre-emption.

IBIT presence is mandatory.

The main responsibility of an IBIT subject is to run the tests located in the BIT Suite module - the Abstract Machine Test, a library of tests that are accessible from either CBIT, IBIT or SBIT (Startup Initiated Built-In Test). After each test, the result will be written to the BIT Results memory region. The BIT Results memory region is a region in memory that BIT subjects can write to and an authorized subject can read from. A test failure will result in an audit record being generated and stored via the Audit module's hypercall interface. After all IBIT subjects have finished, the master IBIT subject will signal the SSM to transition to the appropriate state.

CBIT

The Continuous Built-In Test is a subject that is scheduled in the normal operational scheduling policy. As it is given CPU time, it cycles through a set group of tests. If a test fails, the system will immediately transition to Maintenance Insecure Mode, where it will perform an Initiated Built-In Test.

Unlike IBIT, running CBIT tests is optional. In case it is not required, CBIT subjects shall be omitted in the configuration.

The Separation Kernel Runtime Package (SRP)

The SRP is a binary image, containing the SKH and the subjects and accompanying resources.

The SRP is loaded into a target system's main memory by the bootloader.

At system boot time, the bootloader will load the SRP into the target system’s main memory (DRAM) from storage. The storage may be local to the target system (example: hard disk, flash memory), or located across a network.

The SRP is built with the XML based configuration tool. The specification of which subjects and accompanying resources to include within a given SRP is made within a given XML configuration file. The XML configuration file is provided as input to the XML configuration tool. The primary output of the XML configuration tool is an SRP.

Subject's memory region resources may be initialized with binary images. An example of such images could be the kernel images of subjects, the compressed root file systems of subjects.

It is the responsibility of the system integrator to determine which accompanying resources are to be included in a given SRP. Refer to the XML configuration tool documentation (*LynxSecure 6.3.0 Getting Started and Configuration Guide*, *LynxSecure 6.3.0 Advanced Configuration Guide*) for how to specify the resources for a given SRP.

In general, the lifecycle of an SRP would be as follows:

- Decide which subjects and accompanying resources to include in an SRP
- Decide what external resources comprise the subjects, including the scheduling resources and the CPUs on which the subject will execute
- Create a new XML configuration file, or modify an existing XML configuration file, for the SRP.
- Build/locate the subjects and accompanying resources that are to be included in the SRP
- Use the XML configuration tool, with the given XML configuration file as input, producing the SRP.
- Place the SRP in a location accessible by the bootloader

Once constructed by the XML configuration tool, an SRP is not directly editable for the purposes of modification of that SRP. For example, to modify an SRP, one does not directly edit the SRP, one modifies the XML configuration file then runs the XML configuration tool with that XML file as input to produce a fresh SRP.

The Boot Loader

The boot loader is used to locate the SRP and load it into the memory. The bootloader supports booting from a HDD or USB local storage or the TFTP boot. PXE boot may also be used. Please refer to Chapter 5, "*Booting SRPs*" in *LynxSecure 6.3.0 Getting Started and Configuration Guide* for details.

CHAPTER 3

LynxSecure Features

This chapter provides a detailed description of the features in LynxSecure®.

Scheduling

The scheduled entities are subjects' virtual processors (VCPUs). VCPUs are allocated time slices, also called *minor frames* (in multiples of the System Clock Tick length).

Several minor frames in succession comprise a *major frame*, which is cyclically repeated by the scheduler.

A set of major frames, one per each physical CPU, makes a *scheduling policy*.

The configuration may describe multiple scheduling policies, and authorized subjects may switch between them.

LynxSecure Scheduler is driven by the periodic System Clock Tick (SCT) interrupt from a hardware timer. This interrupt is delivered to every physical CPU in the system at the rate specified in the configuration vector. The Scheduler is designed to offer deterministic behavior. For that reason, LynxSecure uses fixed cyclic scheduling.

Each configuration vector must contain at least one normal scheduling policy (active in the normal operation mode) and one Built-In Test (BIT) scheduling policy (active when LynxSecure performs the built-in tests). Optionally, the configuration vector may specify a separate scheduling policy for the Maintenance Mode and additional scheduling policies that an authorized subject may select when running in the normal operation mode.

It is up to the system integrator to provide predefined scheduling policies, one of which will be used following system initialization. These scheduling policies are created and configured in an offline fashion using the Human Readable Configuration Vector (HCV). Please refer to the section "Scheduling Policies" in *LynxSecure 6.3.0 Advanced Configuration Guide* for details.

Symmetric Multi-Processor Support

LynxSecure can utilize multiple CPUs in an Symmetric Multi-Processor (SMP) system to run subjects. The number of physical CPUs in the system that LynxSecure is expected to utilize is specified in the configuration vector. LynxSecure does not automatically use all available CPUs.

LynxSecure can host multiple subjects on each CPU and each subject may span multiple physical CPUs. This is configured by allocating physical CPU time to subjects' virtual processors (VCPUs). Each subject contains one or more VCPUs. Any physical CPU can only run one subject VCPU at a time. However, a physical CPU may run multiple virtual CPUs of different subjects in succession.

The configuration vector includes a set of schedules describing how each physical CPU time is divided between virtual CPUs of the subjects. LynxSecure requires that all processors have major scheduling frames of the same length; this keeps major frames synchronized across all processors.

If a subject contains multiple virtual CPUs, those VCPUs may be run simultaneously or non-simultaneously on several different physical CPUs. LynxSecure imposes the following limitations:

- Every VCPU is tied to one physical CPU; VCPUs may not migrate between physical CPUs.
- A subject is not allowed to have more than one VCPU scheduled to run on the same physical CPU.

Note that the code running in the subject may not be designed to run in an environment where the processors don't run simultaneously. For example, spin locks, which are widely used in most operating system kernels, would have extremely sub-optimal performance in such a case. Therefore, it's a good idea to run VCPUs of SMP subjects simultaneously on different physical processors unless the code running in this subject is specifically designed to handle the opposite case.

Hyper-Threading Support

LynxSecure supports Hyper-Threading. Some CPU/BIOS vendors may refer to Hyper-Threading as Symmetric Multi-Threading (SMT). By default, LynxSecure does not use more than one hardware thread in each CPU core. However, this can be overridden in the configuration vector.

In general, Hyper-Threading allows each physical CPU core in the system to execute two individual hardware "threads," whereas without Hyper-Threading each CPU core could be thought of as executing only one hardware thread per core.

NOTE: Unlike software threads, hardware threads do not migrate between CPU cores, they are fixed to a CPU core.

To software, such as operating systems, one effect of enabling Hyper-Threading is that two CPU cores appear available in place of each physical CPU core (of the disabled case). While this may be considered to double the total number of CPU cores available on the target board, the two hardware threads on each CPU core share that CPU core in an interleaved (concurrent) fashion, not a simultaneous fashion.

There are some caveats to using Hyper-Threading in LynxSecure. LynxSecure reports each hardware thread as a separate processor core with 1 hardware thread to the subject. Therefore, the subject's guest operating system will not be able to distinguish virtual CPUs running on different hardware threads of the same physical core from virtual CPUs running on different physical cores. This may cause suboptimal process scheduling in an SMT-aware guest operating system.

Because all hardware threads of the same processor core compete for the same set of processing resources, there are some separation considerations. It is recommended to assign all hardware threads of the same processor core to the same subject. In case the major scheduling frame of hardware threads sharing the same processor core is split between different subjects, the schedule should run the same subject on all of those hardware threads during the same time. Otherwise, subjects may affect each other's processing performance in ways non-deterministic and undefined by the processor manufacturer.

Flexible Scheduling

Flexible scheduling is an enhancement to the scheduling module that allows a subject to donate CPU time to another subject.

A group of subjects which can run in the same set of minor frames, or timeslice, is known as a scheduling family. The subject that is scheduled to run during those frames in the scheduling policy is called the timeslice owner. The owner can donate its CPU time to the other subjects, either directly, or by donating time to one subject, which may donate time to another and so forth. The owner may donate its entire timeslice, or only the time until the next SCT, or until an asynchronous event (such as an interrupt) occurs in the owner. The time recipient subject may return the timeslice to the original donor at any time. Subjects donate their time using hypervisor calls and must be authorized to do so in the configuration vector.

This feature allows a system designer to implement scheduling families where a group of subjects can be scheduled to run in a given timeslice according to a scheduling algorithm determined by the designer. For example, a group of subjects operating in a producer-consumer relationship could be configured so that each producer was authorized to schedule its consumers. This would allow for rapid, 'sequential' processing of packetized messages. The flexible scheduling feature is controlled by four hypercalls:

- Give the timeslice to the specified subject until the next SCT.
- Give the timeslice to the specified subject until the next asynchronous event in the timeslice owner subject.

- Give the timeslice to the specified subject indefinitely.
- Return the timeslice to the owning subject.

The first two interfaces give away the subject's timeslice temporarily. Donation until an asynchronous event lets the owner subject donate its idle time to other subjects. Temporary donation allows designers to implement 'interesting' scheduling algorithms like priority-preemptive scheduling.

The indefinite donation gives away the timeslice 'forever', or until the recipient subject returns it. This can be used for implementation of cooperative multitasking within a group of subjects.

Preemptive Scheduling

As shown in Figure 3-1 (page 13), a subject will be scheduled for the timeslice. Based on its own algorithms, it will schedule other subjects to run until the next SCT. When the next SCT interrupt occurs, or the other subjects yield to the owner, the owner will then schedule the next subject to run until the following clock tick.

One use is that the parent subject is effectively a traditional preemptive scheduler, scheduling the subjects as needed. The parent subject, and any other at that level get deterministic scheduling, and the child subject gets priority based scheduling out of the parent's subset of the whole schedule.

Subject 2 acts as a pre-emptive scheduler, scheduling subjects a, b, and c according to its own priority scheme.

The top line represents SCT interrupts. The major frame is 8 SCT intervals long.

The middle line is the subject in the static schedule.

The bottom line represents the subjects scheduled in subject 2's timeslice.

The expanded view of minor frames 1-3 shows how subject 2 owns the timeslice, and how after the SCT interrupt, subject 2 will run briefly, determining which subject to schedule, and passing control to that subject for the remainder of the SCT interval.

Figure 3-1: View of the Minor Frames

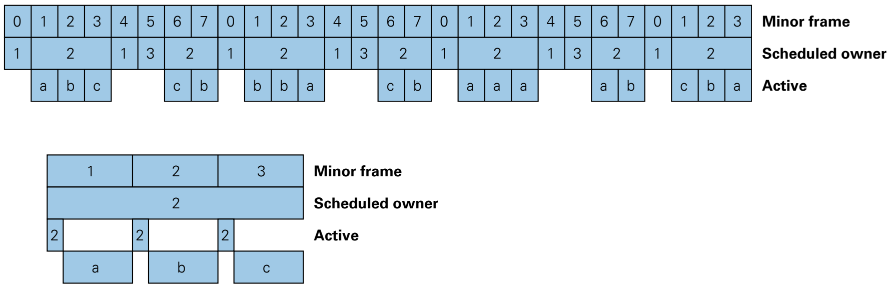


Table 3-1: Preemptive Scheduling

Scheduled Family	Owner	Other Subjects	Minor Frames
1	1	-	0, 4
2	2	a, b, c	1, 2, 3, 6, 7
3	3	-	5

A scheduling family is the set of subjects which run in a timeslice. Each of them could represent a different scheduling policy on the same system.

Sequential Processing

Another use of flexible scheduling is sequential processing of data. Having each subject responsible for a different phase of the sequential helps maintain isolation between subjects. One subject does one task, then passes the processed data directly to another subject, without having to wait for the subject to come up in the scheduling queue. Each subject can then process the data using resources available only to it. One subject receives a message from one source, another encrypts it using a key that only it has access to, the third retransmits the encrypted message on another network.

When a message comes in on subject `a`, subject 2 will pass control to subject `a` to read the message. Once the message is complete, subject `a` passes control to subject `b` to encrypt the message. When the message has been encrypted, subject `b` passes control to subject `c` to transmit it on another network.

The major frame is five SCT intervals long.

Figure 3-2: Sequential Processing

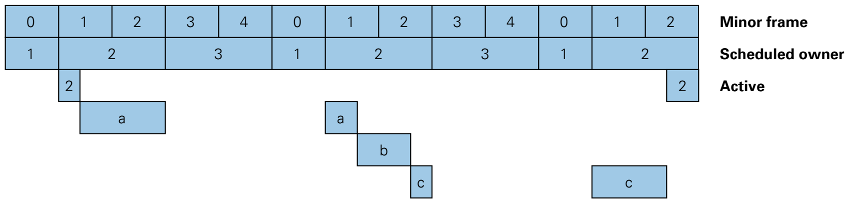


Table 3-2: Sequential Processing

Scheduled Family	Owner	Other Subjects	Minor Frames
1	1	-	0
2	2	a, b, c	1, 2
3	3	-	3, 4

Limitations

- Within a scheduling policy, all physical processors must have major frames that are the same number of SCTs long.
- The minimum length of a minor frame is 1 clock tick. The minimum length of a major frame is 4 clock ticks.
- Only one scheduling policy may be in effect at any time.
- Only one subject is allowed permission to change the scheduling policy.
- A scheduling policy change cannot be invoked until the previous scheduling policy change has completed.
- Scheduling policy changes do not occur immediately. A scheduling policy change can take anywhere between 1 SCT interval to a major frame length plus 1 SCT interval to complete. LynxSecure returns a status whether a request to change the scheduling policy was accepted or denied.

  If the scheduling policy change request is made at any time other than the last SCT interval of a major frame then the scheduling policy will change at the end of the major frame. If the request is made during the final SCT interval of a major frame then the scheduling policy will change at the end of the next major frame.
- Flexible Scheduling actions must be between subjects scheduled on the same physical CPU. If the recipient subject is an SMP subject (contains more than one VCPU), the VCPU running on the same physical processor is the recipient of the donated time. If the donor subject is an SMP subject, only the time slice of the requesting VCPU is donated.
- A given subject may only belong to one scheduling family.
- Care should be exercised when using subject watchdogs with Flexible Scheduling: the timeslice recipient subject can't feed the donor subject's watchdog.

Advanced Hardware Emulation

The virtual devices described in this section are supported with the help of a separate dedicated Virtual Device Server (VDS) subject. Most of them are only available in FV subjects; however, a few are also available in PV subjects.

NOTE: The information in this section pertains to the x86 platform only. LynxSecure for ARM currently doesn't provide either these virtual devices, or the VDS subject.

Virtual ACPI

LynxSecure can emulate ACPI resources and pass through host ACPI events to FV subjects.

Please refer to Chapter 11, "*Virtual ACPI (only x86 platform)*" in *LynxSecure 6.3.0 Getting Started and Configuration Guide* for more details.

Virtual Audio

Emulation of Intel® 82801 AC'97 and Intel 82801JD/DO HDA controllers is supported for FV subjects, providing the capability to play back and record sounds from and to the in-focus subject.

Please refer to Chapter 12, "*Virtual Audio (only x86 platform)*" in *LynxSecure 6.3.0 Getting Started and Configuration Guide* for more details.

Block Device Emulation

LynxSecure provides disk services through block device emulation.

Please refer to Chapter 13, "*Virtual Block Device (only x86 platform)*" in *LynxSecure 6.3.0 Getting Started and Configuration Guide* for more details.

Virtual Keyboard/Video/Mouse

The Virtual Keyboard/Video/Mouse feature provides access to the emulated PS/2 keyboard and mouse devices and a video device for FV subjects.

Please refer to Chapter 14, "*Virtual Keyboard Video Mouse (only x86 platform)*" in *LynxSecure 6.3.0 Getting Started and Configuration Guide* for more details.

Virtual Network

Emulation of Realtek™ 8029 PCI NE2000 and Intel 82540EM E1000 adapters is supported for FV subjects.

Please refer to Chapter 15, "*Virtual Network (only x86 platform)*" in *LynxSecure 6.3.0 Getting Started and Configuration Guide* for more details.

Virtual UART

Please refer to Chapter 16, "*Virtual UART (only x86 platform)*" in *LynxSecure 6.3.0 Getting Started and Configuration Guide* for more details.

Virtual USB

LynxSecure emulates USB 1.1 and USB 2.0 controllers for FV subjects.

Please refer to Chapter 18, "*Virtual USB (only x86 platform)*" in *LynxSecure 6.3.0 Getting Started and Configuration Guide* for more details.

Hypercall API/ABI Subject Authorizations/Permission Checking

Each hypercall provides a specific functionality. A subject's permission, or authorization, to access this functionality is configured in the HCV. Rather than specifying it by hypercall, it is specified by the functionality itself. For example, rather than being configured based on the timekeeping hypercalls, the subject is configured to allow or deny the ability to set or read the time, or the time base calibration value.

Fully virtualized subjects, however, are prohibited from making any hypercalls by default. The `guesthypercallperm="ALLOW"` subject attribute may be used to allow fully virtualized subjects to use hypercalls.

Message Passing API

LynxSecure provides a Message Passing API to allow for short asynchronous messages to be passed between two subjects.

The LynxSecure Message Passing API allows a given subject to send small fixed-size messages to another subject. The LynxSecure Message Passing API also allows for a given subject to receive a small fixed-size message from another subject.

Sending a message is a uni-directional flow. Example, when subject A sends a message to subject B, a message is sent from subject A to subject B; where subject B can then retrieve the message. However, for subject B to respond to subject A with a message of its own, subject B will need to be able to use another, separate, uni-directional flow.

At the high level, messages are sent from one subject to another. However, when a subject makes a hypercall to send a message, it specifies a "message channel" as the target of the message; that message channel contains an association with the receiving subject.

A message channel is a component of a unique uni-directional flow. A uni-directional flow is comprised of a message channel (control data), and a ring-buffer (data region).

Uni-directional flows between two given subjects are established at configuration time. For example, at configuration time, subject A may be set to send messages to subject B (via message channel M), while subject B may or may not also be set to send messages to subject A (via some other message channel).

To use the Message Passing API, subjects make a hypercall to LynxSecure. For example, when subject A creates a message, it then makes a hypercall to LynxSecure with the address of that message as an argument. Then, when LynxSecure delivers the message to subject B, subject B may be notified (message channel may be configured without an IRQ) that it has a message to pick up. Then, subject B can make a hypercall to retrieve the message.

Shared Memory

LynxSecure supports communication between subjects using shared memory. It is achieved by simply assigning the same memory region to two or more subjects in the configuration vector. The communication protocol is up to subject code; subjects may use synthetic interrupts for notifications (see section “PCI Device Interrupt Routing” (page 22)).

Physical Device Assignment to Subjects

LynxSecure supports direct assignment of physical devices to subjects. Normally, any physical device that has hardware resources such as I/O memory, ports and/or interrupts may be assigned to a subject. There are exceptions, however. For example, PCI-to-PCI bridges cannot be assigned to subjects. Devices can only be assigned to subjects with both read and write permissions. In other words, there is no read-only device assignment.

NOTE: Any physical device can only be assigned to at most one subject.

Devices that are not directly on the system bus but behind a controller device, such as i2c endpoints, AHCI mass storage devices, etc. usually cannot be assigned to subjects individually, but only along with their controller. Assigning a controller device to a subject automatically assigns all devices behind it to the same subject. Exceptions include:

- LynxSecure supports individual PCI endpoint device assignment to subjects on the x86 platform.
- LynxSecure supports individual USB endpoint device assignment to subjects when using the Virtual USB feature.

On the x86 platform, LynxSecure obtains information about the devices present in the system from the ACPI tables, except that PCI devices are discovered automatically.

On the ARM platform, LynxSecure obtains information about the devices present in the system from the Devicetree configuration information structure. A Devicetree blob is included in the LynxSecure software package for each supported target.

When a device is assigned to a subject, all of its resources are assigned to the same subject as well. These include:

- I/O memory
- Interrupts
- PCI configuration space registers
- I/O ports (x86)
- RMRRs (x86)
- Clocks (ARM)
- Other resources

The owner subject normally receives read/write information flow permissions for device resources; however, there may be exceptions where the permissions are restricted to read-only. This usually happens when the resource is shared with other devices assigned to other subjects.

LynxSecure picks the new address or ID whereby these resources can be addressed in the subject context. These are referred to as "guest" addresses/IDs. For some resources, LynxSecure may choose to place them at the same address or use the same ID as the corresponding host value.

Pseudo-devices

On the ARM platform, the Devicetree often contains nodes that do not describe any physical device, but rather contain configuration information that physical device nodes refer to. In order to make this information available in the subject's virtual Devicetree, LynxSecure allows the assignment of such device nodes (referred to as "pseudo-devices") to subjects as if they were physical devices. Moreover, LynxSecure allows the assignment of a single pseudo-device to multiple subjects. By definition, LynxSecure considers a device node to be a pseudo-device if the node doesn't have any physical resources such as I/O memory or interrupts.

Interrupt Routing

The system integrator can use the configuration vector to assign one or more hardware devices to a subject. Most devices are capable of generating interrupts, events that interrupt normal code execution and direct it to the interrupt service routine. Based on the configuration, LynxSecure will receive interrupts from those devices and route them to their owner subject.

There are two kinds of interrupts in LynxSecure subjects: external interrupts (interrupts from external physical devices), and synthetic interrupts, such as those generated by a virtual device, a message channel, or by software using hypercalls.

LynxSecure passes interrupt information necessary for subject software to identify interrupts to subjects using a special memory region called the Read-Only Page. In FV subjects, this information is also available to subject software via the platform-specific hardware configuration information structures, such as the ACPI tables on the x86 platform and the Devicetree on the ARM platform.

Each subject in LynxSecure has a virtual interrupt controller. When an interrupt is delivered to a subject, it appears as if coming from the virtual interrupt controller. For most subjects, the virtual interrupt controller resembles a physical interrupt controller that is typical for the target platform. However, paravirtualized subjects on the x86 platform are special. Their virtual interrupt controller is hypercall-based and doesn't follow the specifications for any physical interrupt controller.

LynxSecure also provides a hypercall for sending inter-subject interrupts, which takes the guest IRQ in the destination subject IRQ space as the argument. This hypercall is available in all subject types, but the permission to send each particular IRQ to each particular subject must be explicitly configured in the HCV.

Interrupt Identification on the x86 Platform

To the Operating System or other software, interrupts are distinguished by their associated IRQ number. LynxSecure finds out the IRQ of external PCI devices automatically. For platform legacy devices, however, the IRQ must be specified explicitly in the configuration.

For paravirtualized subjects, the interrupt vector number injected into the device owner subject is obtained by adding the subject's IRQ base value to the interrupt's IRQ number. For example, if the subject's IRQ base is 32 and the interrupt's IRQ is 7, interrupt vector 39 is injected into the subject. The subject's IRQ base can be explicitly specified in the configuration. If not specified, it defaults to 32.

The translation of the host IRQ to the injected interrupt vector is much more complicated for fully virtualized subjects. FV subjects feature a virtual I/O APIC interrupt controller. First, LynxSecure translates the host IRQ to guest IRQ and then the guest IRQ to the virtual I/O APIC input number using unspecified algorithms. Subject software also programs custom interrupt vectors for different virtual I/O APIC inputs; how these vectors are allocated depends entirely on subject software.

NOTE: If specifying the guest IRQ number explicitly in the HCV, one should avoid conflicts with other interrupts and also with any reserved values of interrupts that may be used by the specific subject. For example, for paravirtualized Linux® on the x86 platform, avoid the IRQ number that added to the subject's IRQ base would result in vector 128 (the system call vector).

For details on PCI interrupt routing, refer to section “PCI Device Interrupt Routing” (page 22).

Interrupt Identification on the ARM Platform

On the ARM platform, each subject has a single virtual Generic Interrupt Controller (GIC). The virtual GIC conforms to the ARM GIC specification version 2. This lets LynxSecure distinguish interrupts in a subject by a single IRQ number, similarly to the x86 platform. The IRQ number is the number of the input on the virtual GIC that the interrupt is connected to. These numbers must start at 32 due to the GIC design.

There are two ways to allocate the external interrupt guest IRQs, selected by a configuration option in the HCV and the Autoconfig Tool: "the host values" and "from an IRQ pool". The host values allocation, which is the default method, makes the guest IRQs of external interrupts match the corresponding host IRQs. This allocation type helps running unmodified software in a LynxSecure subject that has interrupt and other resource information hardcoded for bare hardware. Allocation from a pool takes less time during LynxSecure startup, and may be used if guest software uses the Devicetree to determine device resources rather than hardcoding them. Note that the "host values" allocation is only possible if the external interrupts may be identified by a single IRQ number in the host environment; for example, if the target platform has a single physical interrupt controller. If that is not the case, allocation from a pool is used. Synthetic and virtual device IRQs are always allocated from a pool.

The Virtual Interrupt Subsystem in x86 PV subjects

As mentioned above, paravirtualized subjects on the x86 platform use a hypercall-based virtual interrupt controller. It provides most of the features a hardware interrupt controller normally provides: setting the destination VCPU set of an interrupt, inter-processor interrupts, interrupt priorities and End-of-Interrupt (EOI) signaling. The hypercall-based virtual interrupt controller doesn't provide interrupt masking.

By default, all interrupts, both external and synthetic, are routed to VCPU 0 (the first virtual processor defined in the configuration vector) only. It is possible to route an interrupt to multiple VCPUs within the subject; when the corresponding event occurs, each of them will receive the interrupt.

For synthetic interrupts, the subject may choose to completely ignore an interrupt by routing its vector to an empty set of VCPUs. For external interrupts, however, the destination set of VCPUs must contain at least one VCPU. This is a hardware limitation.

The System Clock Tick interrupt, in addition to driving the Scheduler, is passed on to every paravirtualized subject. It is treated as a synthetic interrupt in this respect; in other words, a PV subject may block the delivery of the SCT interrupt to itself by setting its destination VCPU set to an empty set.

IOMMU Support in LynxSecure

IOMMU stands for I/O Memory Management Unit which works very similar to a processor's Memory Management Unit. The main purpose of having IOMMU hardware, such as Intel VT-d, is to provide isolation of device DMA access to main memory.

Provided that the hardware IOMMU is enabled, LynxSecure's IOMMU support is active by default (for a given configuration), with the option to explicitly de-activate it.

The actions of the IOMMU are transparent to subjects, and there are no subject-visible interfaces to the IOMMU.

When physical devices generate DMA reads and writes, the IOMMU ensures that the access is only allowed if it's allowed for the device owner subject. The IOMMU also remaps the access from the subject's guest physical memory address space to the host physical memory address space.

Physical Device Assignment to Subjects with an IOMMU

Normally, the IOMMU should be enabled, as long as it is supported by the hardware and the configuration assigns physical devices capable of DMA (such as PCI devices) to any subject. LynxSecure uses the IOMMU to make sure physical devices cannot violate the information flow policy specified in the HCV; also, to remap device memory accesses from the subject's guest physical memory address space to the host physical address space.

Physical Device Assignment to Subjects without an IOMMU

On a target with no IOMMU capability, physical device assignment to FV subjects is limited: only one FV subject can have assigned DMA-capable devices. The FV subject with assigned DMA-capable devices must be configured with the *identity memory map* using the Autoconfig Tool (`identitymem` subject option, see section “Autoconfig Syntax” in *LynxSecure 6.3.0 Getting Started and Configuration Guide*). The other FV subjects on that target can only have virtual devices assigned to them.

PV subjects don't have this limitation: any number of PV subjects may have assigned physical devices. This is achieved by LynxSecure automatically configuring PV subjects that have assigned physical devices as identity-mapped, if the target system has no IOMMU. This may result in an unusual RAM layout in the physical address space of the subject: the scattered RAM ranges assigned to the subject will appear at their scattered host physical locations in the guest physical address space as well, rather than being grouped together at the start of the guest physical address space. The software running in these PV subjects must be prepared to handle such unusual RAM layouts.

The identity memory mapping in a subject allows physical devices assigned to that subject to perform DMA accesses without the need of remapping. However, the lack of the IOMMU also means that devices may access memory that doesn't belong to their owner subject.

Interrupt Remapping (x86)

Interrupt Remapping is a security feature of the x86 IOMMU (Intel VT-d) that prevents guest software from interfering with the hypervisor by programming the MSI controls of PCI devices in malicious ways. Interrupt Remapping is an optional feature and not all VT-d hardware supports it. LynxSecure supports Interrupt Remapping if it is available. This feature is transparent to subjects; there are no Interrupt Remapping controls visible to them. Note that Interrupt Remapping is not the only layer of protection against MSI-related attacks, so systems without it are still protected to some degree.

Assigned Physical Devices and Subject Restart

Physical devices assigned to subjects may misbehave after their subject is restarted. This subsection explains the architectural reasons behind it and things LynxSecure does to help mitigate this issue.

When an operating system running on bare hardware is restarted, the entire hardware system is reset. This includes all the devices. When the operating system resumes control after a hardware reset, its device drivers may expect to find the devices in their initial state. Those drivers would fail to work properly if they don't find the device in the initial state.

Similarly, when a guest operating system running in a LynxSecure subject is restarted, all devices visible in the subject need to be reset. This is easy to do with emulated devices, but directly assigned physical devices require hardware support.

Resetting all the hardware devices in the system when one subject restarts is a bad idea, because some of those devices could be used by other subjects or the LynxSecure hypervisor itself. An individual device reset is required. Because LynxSecure doesn't normally contain device drivers for arbitrary devices, it needs a generic mechanism for device reset. Unfortunately, in many cases such a mechanism is not available.

The PCIe specification has defined a new optional capability for PCI devices called the Function Level Reset (FLR). With FLR, devices can be reset individually and in a generic manner, without knowing any specifics of the device. LynxSecure (more specifically, the virtual firmware) uses FLR to reset the physical PCI devices at every subject restart if this capability is supported by the device.

However, a lot of devices currently on the market do not support a generic reset mechanism. Such devices are not reset when their owner subject restarts and may malfunction after that, because their Guest OS driver expects to find the device in some particular initial state and fails otherwise. Another problem is that without reset, DMA-capable devices may continue to perform DMA to random areas of RAM and corrupt that memory, causing a GuestOS boot failure. This is a hardware limitation beyond LynxSecure control. LynxSecure warns the user about such devices with messages on the diagnostic console during startup.

LynxSecure virtual firmware only resets the assigned physical devices on subject restart in fully virtualized subjects. PV subjects don't have the virtual firmware and therefore must perform the device reset themselves, if necessary.

PCI and PCI Express Passthrough

On the x86 platform, LynxSecure supports *PCI Device Passthrough*, a mechanism of assigning arbitrary physical PCI and PCI Express (PCIe) end-point devices to subjects.

Most of the information in this section doesn't apply to the ARM platform, where LynxSecure doesn't currently support PCI passthrough for individual PCI devices. On some ARM targets, LynxSecure supports the assignment of a host-to-PCI bridge (and, consequently, all PCI devices behind it) to a subject.

IOMMU and PCI Interaction

PCI devices that share a parent PCI bridge that is not transparent to the IOMMU must be assigned to the same subject. Typically, legacy-PCI-to-legacy-PCI bridges and PCI-Express-to-legacy-PCI bridges are not transparent to the IOMMU. This is a hardware limitation of VT-d.

Some PCI devices, such as Intel on-board graphics and most USB controllers, have associated RAM regions in system memory that must be mapped into the device owner subject to ensure that the device is functional. Using VT-d terminology, these regions are called RMRRs (Reserved Memory Range Regions).

RMRRs are reported to LynxSecure by the BIOS using the ACPI tables. LynxSecure can automatically determine RMRR locations and create the necessary memory region flows when it starts up. It is also possible to explicitly declare RMRR regions and memory flows in the configuration vector.

Autoconfig Tool supports generating both explicit and implicit RMRR configurations depending on its options.

PCI Configuration Space

Each PCI and PCIe device has a set of registers called the PCI configuration space. Some registers in the PCI configuration space are safe to be given direct access to subjects and some are not. For example, Base Address Registers contain host physical addresses of device I/O and memory regions; if subjects were given direct access to those registers, they would be able to interfere with devices owned by other subjects by relocating the device I/O and memory regions in the host address space such that they overlap with those of other devices.

For that reason, LynxSecure partially virtualizes the PCI configuration space of assigned physical PCI devices. Some registers are directly available to guest software running in the subject, some are emulated and some are hidden. LynxSecure hides registers that it can't make directly available to the subject and doesn't emulate.

Notable physical PCI device capabilities that are available to subjects either directly or by emulation include Power Management, Function Level Reset (see above) and Message Signaled Interrupts (only to FV subjects and the Virtual Device Server subject).

Virtual PCI devices appear in the PCI device hierarchy, but their entire PCI configuration space as well as other registers and functions are emulated.

Subjects may reprogram PCI device Base Address Registers (BARs) for I/O and memory regions, with one exception: PV subjects are not allowed to reprogram memory BARs, because their guest physical address space is static. FV subjects don't have this limitation.

LynxSecure supports the PCIe extended configuration space. PCIe extended configuration space makes the advanced features of PCIe devices available to the guest operating system. It is normally accessed through a dedicated region in physical memory, but PV subjects must use hypercalls to access it instead.

The Autoconfig Tool enables PCIe extended configuration space access for FV subjects by default. It will reserve the same guest physical memory address range for PCIe access as the range reserved by the host BIOS on the target system for the same purpose. The address and the size of this memory range are not configurable and are not explicitly specified in the XML configuration file. Note that the PCIe memory range overrides any memory regions overlapping it in the subject.

When PCIe configuration space is turned off, the Legacy PCI configuration space is still available to the subject using the legacy configuration mechanism; so, the subject will see its assigned PCIe devices, but will not be able to use their advanced PCIe features.

PCI Configuration Mechanisms

PCI configuration mechanisms are ways for software to access the configuration space of PCI devices. The original PCI Specification described two configuration mechanisms (so called mechanism #1 and mechanism #2), and the later PCIe Specification added one more mechanism.

The PCI configuration space of a legacy PCI device is 256 bytes. PCIe adds to that the extended configuration space so that the total is 4096 bytes per device. The legacy part of the PCI configuration space of virtual and physical PCI devices is visible to all subjects via the PCI configuration mechanism #1. The way to access to the extended part depends on the subject type.

In FV subjects, the extended PCI configuration space is available via the standard PCIe configuration mechanism. This mechanism requires a memory region in the physical address space. Since the required memory region is relatively large and the access to the extended PCIe configuration space is not always required, this configuration mechanism is optional for FV subjects.

The memory-mapped PCIe configuration mechanism is not available to PV subjects. Instead, those subjects can use the hypercall PCI configuration interface to access the extended PCIe configuration space. No platform feature flow is needed in the configuration vector in that case.

PCI Device Firmware

Some PCI devices come with associated firmware that is activated during the system boot time. For example, graphics cards typically come with an associated Video BIOS, which provides video mode switching during the BIOS initialization and OS boot. Network cards often have associated firmware that enables booting over the network (PXE). This firmware may or may not be required for the proper functioning of the device in a virtual environment.

LynxSecure doesn't support PCI device firmware in PV subjects, because the subject boots directly into the GuestOS without involving the BIOS. Support in FV subjects depends on multiple factors and is discussed below.

In the PC architecture, PCI device firmware may be located in one of the two possible places: either on the card itself in its Expansion ROM (also called the Option ROM), or incorporated into the host BIOS. If the device is an add-on card, it's normally the former case. If the device is an on-board integrated device, it's normally the latter case.

LynxSecure support for Expansion ROMs depends on the hardware capabilities of the CPU. With the required capabilities present, Expansion ROMs are fully supported and are automatically utilized by the virtual BIOS. More specifically, for Intel CPUs the required capabilities are: the *Extended Page Table (EPT)* and *Unrestricted Guest*. Support for these features varies between CPU models, although fairly modern CPU models tend to support them; please contact the CPU manufacturer for information on a particular CPU model. If either or both of those two capabilities are missing, LynxSecure doesn't support Expansion ROMs.

Another factor that may prevent an Expansion ROM from functioning under LynxSecure is the host BIOS. Some host BIOS'es allocate PCI resources in a way that makes the Expansion ROM of some devices inaccessible to software without a resource re-allocation. LynxSecure makes an attempt to re-allocate PCI resources and repair such configurations; in some rare cases that may fail and the Expansion ROM remains inaccessible.

In case the firmware is incorporated into the host BIOS, in general, LynxSecure doesn't support it; in other words, such firmware will not be visible to the FV subject. However, LynxSecure attempts a work-around for graphics cards. This work-around is not guaranteed to work perfectly; in fact, most often it will only work for the initial boot of the FV subject and will stop working if the subject is restarted. The user may see such symptoms of this as the screen remaining dark or displaying garbage during the GuestOS boot time, and sometimes permanently until the next reboot of the target system. Sometimes, disabling device reset on subject restart may help with this problem. In general, the chances of getting PCI graphics cards working with the least number of issues are higher for add-on graphics cards than integrated graphics cards.

PCI Device Interrupt Routing

Based on the configuration, LynxSecure will receive interrupts (*external interrupts* in LynxSecure terms) from PCI devices and route them to their owner subject.

To the software running in a subject (e.g. an operating system), interrupts are distinguished by their associated IRQ number. In general, the guest IRQ number depends on the subject's IRQ base and the host IRQ of the external PCI device.

LynxSecure finds out the host IRQ of external PCI devices automatically. The hardware only supports a limited number of distinct host IRQs. LynxSecure allocates the host IRQs as follows: first, the IRQs for legacy (non-MSI) external interrupts are reserved. Then, any synthetic interrupt IRQs specified in the configuration vector are reserved. The remaining IRQs are used for Message Signaled Interrupts of MSI-capable devices. If it is impossible to allocate all MSI IRQs requested by assigned devices in the system, LynxSecure can emulate some of them by polling the corresponding device. For such emulated IRQs, LynxSecure prints a warning to the diagnostic console during initialization. If there are still some MSI IRQs that couldn't be allocated or emulated, LynxSecure stops and requests the user to change the configuration so that fewer MSI-capable devices are assigned to subjects.

For PV subjects, the subject's IRQ base can be explicitly specified in the configuration; if not specified, it defaults to 32.

All FV subjects require an IRQ base of 32.

The FV subject has its own virtual IRQ space, which is independent from the host IRQ space. The guest software running in a FV subject can program the virtual interrupt controller and assign the IRQ numbers for various interrupt sources (PCI physical device, PCI virtual device, etc.)

Message Signaled Interrupts

PCI devices can use one of the two interrupt signaling mechanisms:

- Legacy PCI interrupt pins;
- Message Signaled Interrupts (MSI).

Interrupt pins can be thought of as physical wires going from the device to the interrupt controller. This is the default mechanism for all PCI devices. There is a maximum of 4 pins per PCI bus, and the pins are typically shared by different PCI buses. This forces PCI devices to share the few available interrupt lines and limits the number of distinct interrupt vectors PCI devices in the system can generate.

The purpose of MSI is to remove the limitations of the legacy PCI interrupt pins. When MSI is enabled for an MSI-capable device, the device can be programmed to generate any interrupt vector. The total number of distinct interrupt vectors is limited by the platform, but it is still much greater than the number of distinct interrupt pin lines. For example, for a subject with an IRQ base of 32, the usable interrupt vector range is from 32 to 255.

LynxSecure supports both MSI and MSI-X (an extended version of MSI) PCI device capabilities. Guest software running in subjects can see the MSI capabilities of the assigned devices and can switch the devices between the legacy and the MSI interrupt signaling mechanisms.

Message Signaled Interrupts in x86 PV Subjects

On the x86 platform, PV subjects have some restrictions on using PCI device MSI interrupts:

- MSI interrupt vectors are pre-allocated and cannot be changed by guest software running in the PV subject.
- For the MSI capability, the Message Address and Message Data registers are read-only.
- For the MSI-X capability, the entire MSI-X table is read-only.
- If the physical device has both MSI and MSI-X capabilities, only the MSI-X capability is visible to guest software.

These subjects can use the Set Interrupt Routing hypercall to set the destination VCPU for MSI interrupts rather than setting it in the Message Address register.

Because the MSI-X Table is read-only, LynxSecure provides an extension for masking and unmasking MSI-X vectors. In order to mask an MSI-X vector, write the (normally read-only) MSI-X Table Offset register in the MSI-X capability with a 32-bit value where the high 16 bits are the MSI-X Extension Magic and the low 16 bits are the vector index to mask.

Similarly, in order to unmask an MSI-X vector, write the (normally read-only) MSI-X PBA Offset register in the MSI-X capability with a 32-bit value where the high 16 bits are the MSI-X Extension Magic and the low 16 bits are the vector index to unmask.

The MSI-X Extension Magic value is 534C (hexadecimal).

If it helps to resolve an IRQ sharing conflict between subjects, LynxSecure may use the MSI mechanism when guest software thinks the device is in the legacy interrupt signaling mode. To guest software, this looks as if the device generates the same interrupt vector in both legacy and MSI modes. This behavior doesn't apply to FV subjects.

PCI Optional Features

LynxSecure supports the following optional features of direct-assigned physical PCI devices:

- Message Signaled Interrupts (MSI and MSI-X)
- Power Management
- Function-Level Reset
- Single-Root I/O Virtualization (SR-IOV)

SR-IOV Support

SR-IOV is a feature of some PCIe devices allowing for multiple Virtual Machines to share one physical device. It is achieved by the primary physical SR-IOV capable device (called the *Physical Function*, or PF) presenting a certain number of additional physical devices on the PCI bus called the *Virtual Function* devices, or VFs. These VF devices can be assigned independently to different subjects in LynxSecure.

Care should be taken when using SR-IOV devices in an environment with strict information flow control. Note that the SR-IOV specification doesn't make any requirement for the (lack of) dependencies and interference between the PF and the VFs. As a result, the VFs typically cannot be considered completely independent devices; assigning them to different subjects may create information flow between the PF owner subject and the VF owner subjects, and thus between the VF owner subjects as well.

The VFs usually require that the owner subject of the PF device has an SR-IOV aware device driver loaded and configured to enable the VFs.

Subject Execution State Manager (SESM)

The Subject Execution State Manager enables one subject to alter the state of another subject or its own state: the target subject can be started, stopped, suspended, resumed or restarted.

SESM is available to subjects via the hypercall interface. In order to use SESM hypercalls on another subject, a subject needs appropriate permissions given to it in the configuration vector. Each of the five possible actions has a separate associated permission. Each initiator subject's SESM permissions are also distinct for distinct target subjects. However, subjects can always use SESM hypercalls on themselves, no explicit permissions are required in that case.

The SESM actions are a disruptive tool, in that if they are used to force a subject to restart, they restart the subject without the subject's knowledge. If a subject is to perform clean shut-down actions prior to being restarted by SESM actions, notification to that subject should be made through other mechanisms. For example, one can use message channels, shared memory communication, virtual networking, or other methods for one subject to inform another of the need to prepare for the impending shutdown; then the other subject can use a SESM hypercall on itself to shut itself down or restart.

Internal Timekeeping

LynxSecure hypervisor maintains two central times called the Wall time and the Monotonic time. The Wall time is given in seconds and nanoseconds since the epoch, or midnight on January 1st, 1970 Co-ordinated Universal Time. The Monotonic time is measured in seconds and nanoseconds since LynxSecure boot time.

The Wall time is used internally by LynxSecure for things like audit record timestamps, and can also be used by subjects. There are hypercalls that let a subject request or set the current Wall time and adjust the time flow rate, provided it has the proper permissions. It is therefore possible for a subject to synchronize the Wall time with a time server using the NTP protocol.

The Monotonic time can only be read by subjects, but never set. Just like the Wall time, requesting the current Monotonic time is done using a hypercall. This time is ideal for applications where a subject doesn't want the time to shift suddenly because another subject with the permission to set the time did so.

Subjects may maintain their own version of time using platform features like the Time Stamp Counter register and timer interrupts. However, running under the LynxSecure scheduler may impact the ability of a subject to do that accurately. For example, in case the subject's VCPUs don't occupy 100% of the physical CPU time in the schedule, multiple timer interrupts may arrive during the period when the subject is not scheduled to execute on a physical CPU. Due to hardware limitations, when the subject finally gets to run it will normally see just one interrupt. LynxSecure mitigates this for the System Clock Tick interrupt in paravirtualized subjects on the x86 platform (only those subjects have the SCT interrupt). LynxSecure queues the SCT interrupts if it isn't possible to deliver them right away; all of the missed SCT interrupts are delivered sequentially when the subject eventually gets to run. If the subject is using something other than the SCT interrupts to maintain its time, it may synchronize with LynxSecure Wall time to improve the accuracy.

An Explanation of Timebase Calibration

Initially, LynxSecure calibrates its Wall time rate using platform timers with a known frequency. For configurations where that is not enough, LynxSecure provides a mechanism for a subject to change the Wall time calibration parameters. Note that these parameters do not affect the rate of the Monotonic time.

A source of an accurate value of the real world time is dependent upon the implementation of each system, and therefore cannot be part of LynxSecure. Providing and reading that time source is the responsibility of the system integrator. Calibration of the internal timekeeping system assumes that there will be a subject with access to reading an accurate real time clock, and access to read and set the LynxSecure Wall time and calibration values.

The time calibration value determining the Wall time rate in LynxSecure is the number of nanoseconds per one increment of the timebase register. On the x86 platform, LynxSecure uses the Time Stamp Counter, which increments at the CPU frequency rate, as the timebase register. For example, LynxSecure running on a 1GHz CPU would normally have the corresponding timebase calibration value of 1 nanosecond.

On the ARM platform, LynxSecure uses the Generic timer as the timebase register.

The simplest way for a subject to set and calibrate the internal timekeeping is to read the external reference, and set the system time to that. After some period, it can simultaneously read the external reference and the internal time. Knowing, for example, that in 100 real world seconds the internal time advanced 99 seconds and the current calibration value is 1 nanosecond, the clock could be set one second ahead and the calibration value set to $1\left(\frac{100}{99}\right) \approx 1.01$ nanoseconds.

This approach, while simple, has various disadvantages, not the least of which is the discontinuous progression of apparent time. A much better approach, as long as the internal clock is not too far off of the reference clock is the approach used by NTP, which mimics the behavior of a phase locked loop. If the internal time is behind real world time, its progress is sped up, faster than real world time, until the delta approaches zero, at which point it is throttled back so that the internal rate of time passage matches that of the real world.

It is important to note that this algorithm depends upon (nearly) simultaneously reading the internal time and the reference time, but since the calibration value is simply the apparent rate of time passage, it does not depend upon how the timestamp counter is converted to real time.

Also, (assuming a steady clock) since time is always read in integer timebase ticks, the resolution of a time delta is always ½ of a timebase tick, whether the delta time is 3 ticks or 3,000,000.

This also means that no matter how many times the time stamp counter is read, the delta time will be an integer number of timebase ticks, so it doesn't matter whether the time is only read twice (once at the beginning, and once at the end) or 47 times.

Let's take a system with a 1GHz timebase clock, where the calibration value would be 1 nanosecond per tick. Let's first have the calibration routine read the time at 10 seconds, then at 10.001 seconds.

Time ref	Timebase	Delta	Calc time	
10.000000000	10000000000	.000000	10.000000000	Cal 1st read
10.001000000	10001000000	.001000	10.001000000	Cal 2nd read

Now let's look at what happens if several events happen in between:

Time ref	Timebase	Delta	Calc time	
10.000000000	10000000000	.000000	10.000000000	Cal 1st read
	10000100000	.000100	10.000100000	event 1
	10000500000	.000400	10.000500000	event 2
	10000750000	.000250	10.000750000	event 3
10.001000000	10001000000	.000250	10.001000000	Cal 2nd read

That is a system that is running at nominal, now let’s assume that it is running 10% slow. If we only read twice, the values we would get would be:

Time ref	Timebase	Delta	Calc time	
10.000000000	10000000000	.000000	10.000000000	Cal 1st read
10.001000000	10001000000	.000900	10.000900000	Cal 2nd read

And if we read and update the time three times between, we get:

Time ref	Timebase	Delta	Calc time	
10.000000000	10000000000	.000000	10.000000000	Cal 1st read
	10000090000	.000090	10.000090000	event 1
	10000450000	.000360	10.000450000	event 2
	10000675000	.000225	10.000675000	event 3
10.001000000	10000900000	.000225	10.000900000	Cal 2nd read

Audit Functionality

LynxSecure provides auditing capabilities to record events that occur within the SKH and subject software. The Hypercall interface provides a mechanism by which subjects can record an audit event to the SKH for the fusion of audit events across all authorized sources. Events that occur within the subject software can optionally be recorded using the audit subsystem, if the subject software has the necessary permissions and is configured to do so.

The audit functionality is configurable in a variety of ways by the system integrator, in an offline fashion using the Human Readable Configuration Vector (HCV), and runtime via the Hypercall interface.

The Audit feature has these basic auditing capabilities for LSK and trusted subjects:

- LynxSecure runtime generation of audit records caused during the course of subject execution
- Temporary storage of audit records generated by the LynxSecure runtime and trusted subjects, in an internal buffer in volatile memory
- Export of audit records to an authorized subject for permanent storage
- Configurable audit record filter

Audit Buffer Overflow

When the Audit Module has stored the maximum number of audit records and a new audit record is generated, the overflow action will occur. The overflow action is specified in the HCV and has the following options:

- `DROP` - discards the audit record but a count of the dropped event for each audit type is held in an overflow buffer.
- `RESCHED` - switches to a different scheduling policy. The name of the scheduling policy is specified in the parameter attribute (If this option is used a scheduling policy must be specified).
- `MAINTMODE` - invokes maintenance mode.
- `SHUTDOWN` - halt the SKH, including all subjects. This is an immediate halt, not a clean shutdown, so any information cached by subjects may be lost.

The Overflow audit record is an array of counters for various event types and is incremented by one whenever an overflow occurs. The overflow record can be retrieved by subjects by using a hypercall. After the records are retrieved, the counters are reset.

Description of Audit Rule Match Actions

- `DISCARD` - The audit record will be discarded

- `STOREONLY` - LynxSecure will attempt to store the Audit record
- `MAINTMODE` - LynxSecure will invoke Maintenance mode after storing the audit record
- `SHUTDOWN` - LynxSecure will attempt to store the Audit Record then invoke a system shutdown.

Priority of Audit Rules Versus Audit Overflow Action

In cases where an Audit Record causes an overflow of the Audit buffer, the rule covering the initial event and the rule covering the overflow event may specify different actions to be taken. The action taken will be the higher priority of the two specified actions. In decreasing order, the priority of the actions that these events may specify are:

- Shutdown
- Maintenance mode
- Change scheduling policy
- Store

To summarize the actions:

Shut Down the System

If one of these events could indicate that the system has been compromised, the most secure solution may be to shut down the system.

Enter Maintenance Mode

By entering Maintenance Mode, the system can perform an Initiated Built-In Test, and determine what, if any actions are needed to preserve LynxSecure integrity.

Change the Scheduling policy

It may be advisable to only run specific subjects in such an event. For example, the subjects configured may perform diagnostics, or all non-critical subjects may be disabled. Alternatively, a subject may be scheduled to read all of the events from the Audit buffer and write them out to secondary storage, such as the hard drive.

Store

Store the event in the Overflow Audit Record, a special location set aside to record an audit buffer overflow.

System State Manager (SSM)

SSM is responsible for determining what hypercalls and transitions are permitted from a given state. While SSM's primary function is system control through state management, it is an agent and not the decision maker. Determining what states to enter is left to LynxSecure components and the system integrator. State transitions bring major changes in LynxSecure's functionality, which may include changes to scheduling policy or a system wide shutdown or restart. SSM is available to subjects via the hypercall interface, subject to permissions.

System States

Startup

The state of LynxSecure when modules and data structures are being initialized on boot up.

Validation

A part of the boot sequence that guarantees security for the initial transition to the Operation state by running tests on each subject in the Startup Built-in Test (SBIT) schedule.

If a fault is detected, LynxSecure is transitioned to the Maintenance Insecure state.

If no faults are detected, LynxSecure is transitioned to the Operation state.

Operation

Normal operational state of LynxSecure.

Initiated Built-in Test (IBIT)

If a fault is detected, LynxSecure is transitioned to the Maintenance Insecure state.

If no faults are detected, LynxSecure is transitioned back to the state from which it was invoked.

Maintenance Secure

The state of LynxSecure when “maintenance” has been requested by the operator from the Operation state IBIT or other procedures can be carried out.

A return to the Operation state is permitted.

Maintenance Insecure

The state of LynxSecure when a fault has been detected. Critical subjects (defined in the HCV) continue to run, but other functionality is limited. An authorized subject can access audit and BIT information to determine the fault. An authorized subject can read the `BITRESULTS` memory region.

Restart

The state of LynxSecure when zeroing out memory prior to a BIOS triggered platform restart.

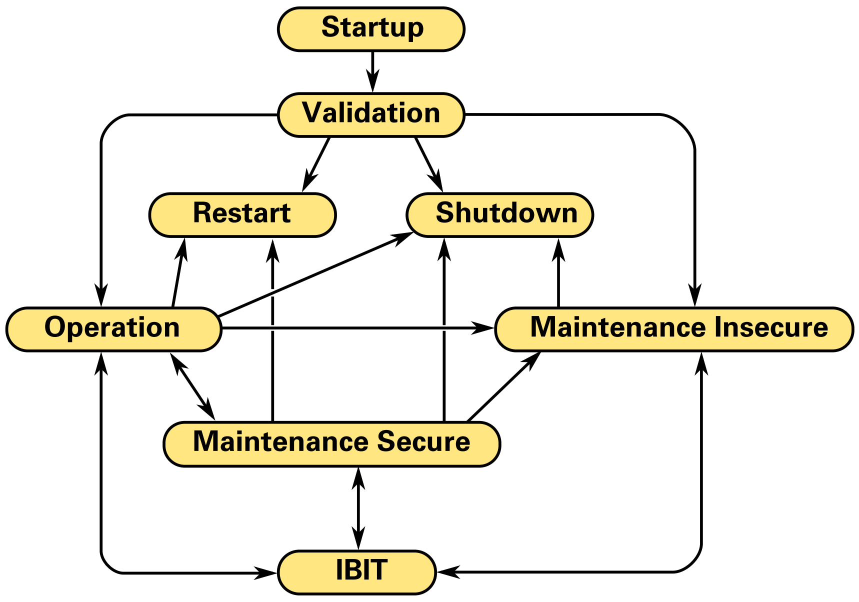
Shutdown

The state of LynxSecure when zeroing out memory prior to a BIOS triggered platform shutdown.

NOTE: The behavior described for the IBIT state is the one implemented in the standard IBIT implementation provided with LynxSecure, `ibit.bin`. Custom BIT implementation may validate system integrity and transition the system back to the Operation state.

Valid SSM transitions are described by Figure 3-3 (page 28).

Figure 3-3: SSM Transitions



Hypercalls are only allowed in specific system states. Please refer to the Appendix A, "*Hypercall Permission by System State*" in *LynxSecure 6.3.0 API Guide* for details on the hypercall permissions associated with the system states.

In general, hypercalls are denied for Startup, Shutdown, and Restart and allowed for Operation, Validation and Initiated BIT. Maintenance Insecure is only slightly more restrictive than Operation, denying hypercalls for a return to Operation or a transition to Restart.

Functionality of the Built-in Test (BIT) Module

In order to make sure that a LynxSecure system is running in a secure state LynxSecure has a suite of Built-in Tests. These tests fall into two categories, the Continuous Built-In The IBIT tests are run at startup. After the system starts normal operation, the optional CBIT tests are continuously cycled through in the background on each processor. If one of those tests detects an error, the system will switch to maintenance mode and a more in depth set of tests may be performed. Certain critical subjects can be configured to continue running in maintenance mode, but since a failed test can indicate a compromised system, all non-essential functionality should be halted until the errors can be corrected either automatically or through manual intervention.

TEST Result Buffers

IBIT and CBIT write their results to special result buffers. Those buffers are available to a special subject via the HCV. The subject will have READ permission to the BIT results memory region.

Configuring Built-in Test in the HCV

There will be one IBIT and up to one CBIT subject for each CPU.

Configuring BIT

In the following sample file, the BIT subjects are `ibit0`, `ibit1`, `cbit0` and `cbit1`. The snippet of an HCV is provided as an example. For details and explanation refer to the section "Built-In Test Subject Specifics" in *LynxSecure 6.3.0 Advanced Configuration Guide*.

```
<subject sname="ibit0" startaddr="00000000c0000000"
    initparams="verbosity=false"
    isa="32BIT" maintperm="ALLOW" on_fatalfault="maintenance"
    on_watchdog="maintenance" ramsize="0000000000100000" role="IBIT"
    type="PARAVIRT" watchdog_timeout="1000">
  <messagebufferflow msgbufname="ibit0_to_ibit1" writeperm="ALLOW"/>
  <messagebufferflow msgbufname="ibit1_to_ibit0" irq="auto" readperm="ALLOW"/>
  <auditflow auditname="audit" readperm="DENY" writeperm="ALLOW"/>
  <virtualprocessor vpname="ibit0.VCPU0"/>
</subject>
<subject sname="ibit1" startaddr="00000000c0000000"
    initparams="verbosity=false"
    isa="32BIT" maintperm="ALLOW" on_fatalfault="maintenance"
    on_watchdog="maintenance" ramsize="0000000000100000" role="IBIT"
    type="PARAVIRT" watchdog_timeout="1000">
  <messagebufferflow msgbufname="ibit0_to_ibit1" irq="auto" readperm="ALLOW"/>
  <messagebufferflow msgbufname="ibit1_to_ibit0" writeperm="ALLOW"/>
  <auditflow auditname="audit" readperm="DENY" writeperm="ALLOW"/>
  <virtualprocessor vpname="ibit1.VCPU0"/>
</subject>
<subject sname="cbit0" startaddr="00000000c0000000"
    initparams="verbosity=false"
    isa="32BIT" maintperm="ALLOW" on_fatalfault="maintenance"
    on_watchdog="maintenance" ramsize="0000000000100000" role="CBIT"
    type="PARAVIRT" watchdog_timeout="1000">
  <memoryflow memregname="cbit.bin" gva="auto" readperm="ALLOW"/>
  <auditflow auditname="audit" readperm="DENY" writeperm="ALLOW"/>
  <virtualprocessor vpname="cbit0.VCPU0"/>
</subject>
<subject sname="cbit1" startaddr="00000000c0000000"
    initparams="verbosity=false"
    isa="32BIT" maintperm="ALLOW" on_fatalfault="maintenance"
    on_watchdog="maintenance" ramsize="0000000000100000" role="CBIT"
    type="PARAVIRT" watchdog_timeout="1000">
  <memoryflow memregname="cbit.bin" gva="auto" readperm="ALLOW"/>
  <auditflow auditname="audit" readperm="DENY" writeperm="ALLOW"/>
  <virtualprocessor vpname="cbit1.VCPU0"/>
</subject>
```

Once launched, CBIT continuously invokes the BIT Suite tests and stores the results in dual buffers. With the completion of each individual test, the results are stored in one side of the dual buffer. This is the *current* side of the buffer. When the complete suite of tests is finished, the *current* side is marked as *previous*, and the other side is marked as *current*. The type of data stored for each individual test will contain the following:

- Time stamp: the current system time
- Test: the name of the individual test
- Results: a pass/fail indication

The last item of data in each side of the dual buffers will be an indication of *current* vs *previous*.

Each time the complete suite of tests is finished, and after the results are stored and the *previous* and *current* status fields are updated. The authorized subject can then read the contents of the dual buffers to determine the results of the tests.

While running the test suite, if a test fails, CBIT will first save the individual result in the *current* buffer as usual, and then generate an SKH Maintenance hypercall. The result of the Maintenance hypercall will cause the SKH to enter Maintenance mode. Therefore, CBIT will be interrupted and the *current* buffer will not contain a complete set of test results (unless the failure occurs on the last test). Because all test results are time-stamped, an authorized subject reading the dual buffers can determine the *current* buffer and then find the test that led to Maintenance mode.

Test Sequence Implemented in CBIT/BIT modules for the x86 platform:

- Verify that a PTE within one of the CBIT modules controlled page tables can be modified successfully (no errors returned).
- Verify an error is returned when attempting an invalid PTE modification in one of the CBIT/IBIT modules controlled page tables.
- Verify if LynxSecure can successfully modify one of its PDTs (no errors returned).
- Verify an error is returned when attempting an invalid PDT modification.
- Verify a page fault is generated when accessing an unmapped memory location.
- Verify a page fault is generated when writing to a read only mapped memory region.
- Verify an invalid-opcode exception is generated when attempting to execute a `GETSEC` instruction.
- Verify an invalid-opcode exception is generated when attempting to execute a `XSETBV` instruction.
- Verify that a `LYNXSK_PDE_PERMISSION_DENIED` error code is returned when attempting to invoke a hypercall method for which the module is not authorized.
- Verify an invalid-opcode exception is generated when attempting to execute a VMX control instruction.
- Perform write followed by read operations on an I/O device for which CBIT is not authorized, and verify that the return value is all 0xFF.
- Verify that no general protection fault occurs when performing a read/write operation of bits that the CBIT/IBIT subject is allowed to modify, in `CR3` register.
- Verify a general protection exception is generated when attempting a `MOV` to `CR3` instruction with an invalid (Page Directory) pointer argument.
- Verify an invalid-opcode exception is generated when attempting to execute a `MOV` to `CR8` instruction.
- Verify an invalid-opcode exception is generated when attempting to execute a `MOV` from `CR8` instruction.
- Verify a general protection exception is generated when attempting a `RDMSR` instruction with an unsupported MSR.

- Verify a general protection exception is generated when attempting a `WRMSR` instruction on a read-only MSR.
- Verify a general protection exception is generated when attempting a `RDPMC` instruction.
- Verify various functions of the high resolution timer module.

Maintenance Mode

Maintenance Mode (MTM) is a state of the system represented by a specific maintenance Mode Scheduling Policy. It is generally run while the system is in an insecure state or when an authorized subject has required to run it. Subjects that run in maintenance mode scheduling policy should ensure the following are implemented at a minimum:

- Verify audit buffer is not full
- Check the latest BIT results.(Implies that MTM subjects should be configured to have read access to the BIT Results memory page)
- Run BIT Suite tests.

Separation Kernel Debugger

LynxSecure provides an optional debugger module (SKDB) for the hypervisor. In the current release, the debugger module is not complete and should be considered experimental. When the debugger module is enabled, the hypervisor will enter the debugger in case of a panic - allowing the user to examine the hypervisor's structures, memory and registers. At this time, only the serial console is supported for SKDB operation. Also, currently there is no way to request an entry into the debugger from the console; the only way to enter SKDB during normal operation is using the Enter SKDB hypercall described in section “SKDB_ENTER - Enter SKDB” in *LynxSecure 6.3.0 API Guide*.

Table 3-3 (page 31) summarizes the commands available in SKDB:

Table 3-3: SKDB Commands

Command	Alias	Description
`help`	`h`	Without an argument, display the list of supported SKDB commands. With an argument, display the help message for the specified command.
`show`	`s`	Interpret the specified structure. See section “Command: show” (page 32) for details.
`mem`		Read from or write to the memory.
`io`		Read from or white to the I/O port.
`pci`		Read from or white to the PCI configuration space.
`regs`		Display CPU's general-purpose registers at the time SKDB was entered.
`vmcs`		Display VT-x registers of a given CPU.
`backtrace`	`t`	Display the stack trace of a given CPU.
`stopcpu`		SKDB may not stop some CPUs if they haven't finished the start-up sequence. In that case, the `stopcpu` command can be used to attempt to bring those CPUs into SKDB. Note that if at least one of the target CPUs hasn't initialized the exception handling yet, an attempt to stop it will reset the system.
`reboot`	`R`	Restart the system.
	`m`	Dump memory. This is an alias to the `mem read` command, provided for LynxOS® compatibility.
	`c`	Change memory or register value. This is an alias to either the `mem write` or `regs` command, provided for LynxOS compatibility.
`address`		Display the contents of the memory at *`address`*. Provided for LynxOS compatibility.

Command	Alias	Description
`%reg`		Display the value of a register, or the contents of the memory addressed by a register. Provided for LynxOS compatibility.
	`+`	Display the contents of the memory following the previous display. Provided for LynxOS compatibility.
	`-`	Display the contents of the memory preceding the previous display. Provided for LynxOS compatibility.

Command: show

The `show` command allows one to examine different structures used by the hypervisor. There are two variants of the `show` command:

- First, it is possible to invoke it as `show` *`type address`*, where *`type`* is a supported structure type and *`address`* is the HPA where the structure is located. Without arguments, the `show` command displays the list of supported types.
- Second, it is possible to use the `show` command to navigate the structures. Without arguments, the show command displays the list of navigation links from the current (last displayed) structure. If the current structure is an element of an array, `show +` and `show -` commands display the next and previous elements, respectively.

For example:

```
skdb> s subject 0           # displays the subject #0
skdb> s +                   # displays the subject #1
skdb> s vcpu 1              # displays the VCPU#1 of the subject #1
skdb> s -                   # displays the VCPU#0 of the subject #1
skdb> s ir                  # displays interrupt routing for VCPU#0
skdb> s subject 1 vcpu 0 ir # the same, as a single command
```

Memory Sanitization

The Memory Sanitization feature allows systems configurators to specify an additional GuestOS to run within a subject. This GuestOS (sanitization context) is run before the primary GuestOS of the subject. The sanitization subject executes until the next restart of that subject. For example, after the sanitization context executes, and clears subject memory (and optionally performs other user-defined actions), it can make a hypercall requesting subject restart. LynxSecure then launches the main GuestOS of the subject, such as Linux.

Each restart of the subject results in LynxSecure toggling to or from the sanitization context and the main GuestOS, depending on which was currently running. If the main GuestOS was running, a switch is made to the sanitization context, and if the sanitization context was running, a switch is made to the main GuestOS. LSA subject is recommended as the sanitization context. Nonetheless, system integrators can create their own sanitization context image, if necessary.

Memory Sanitization provides a broad and extendible mechanism. Systems integrators and developers can extend the scope of what a sanitization GuestOS does. For example, they can add code to a sanitization context, that performs additional sanitization of resources to which the given subject has a flow.

Memory Sanitization is an optional feature. It can be enabled on a per-subject basis.

Please refer to the section “Memory Sanitization” in *LynxSecure 6.3.0 Advanced Configuration Guide* for specific implementation details.

Platform- and Target-Specific Features

LynxSecure provides unique features for some platforms and targets.

x86 Features

The following features are shared by all targets on the x86 platform.

Intel Resource Director Technology

LynxSecure supports certain Intel CPU features from Resource Director Technology domain, such as Cache Allocation Technology.

NOTE: Not all processors support Intel Resource Director Technology.

The Cache Allocation Technology enables the last level cache distribution among the tasks running in the system improving the deterministic capabilities of the mission critical tasks.

The Basic Virtual Platform

The virtual platform provides at least the following features in each LynxSecure subject:

- Up to 32 virtual CPUs supporting 64-bit, 32-bit and 16-bit instruction sets.
- CPU code execution at the user and supervisor privilege levels. Nested virtualization is not supported.

Additionally, FV subjects provide the following features:

- A standard x86 Local Advanced Programmable Interrupt Controller (APIC) for each virtual CPU.
- A standard x86 I/O APIC interrupt controller.
- A standard x86 Programmable Interrupt Controller (PIC) device.
- A standard x86 Programmable Interval Timer (PIT) device.
- A standard x86 Real Time Clock (RTC) device.
- ACPI tables describing the subject's configuration.

x86 Limitations

The following features are notably not supported on the x86 platform, as compared to other platforms:

- The assignment of a host-to-PCI bridge to a subject.

ARM Features

The following features are shared by all targets on the ARM platform.

The Basic Virtual Platform

The virtual platform provides at least the following features in each LynxSecure subject:

- Up to 8 virtual CPUs conforming to the ARMv8-A specification and supporting the 64-bit instruction set, the 32-bit ARM and the THUMB instruction sets. Note that the ARM and THUMB instruction sets are only supported as long as the physical CPU supports them.
- CPU code execution at EL0 and EL1 privilege levels. Nested virtualization is not supported.

- A virtual interrupt controller conforming to the Generic Interrupt Controller version 2 specification.
- Generic Timers conforming to the ARMv8-A specification.
- Virtual PSCI (see below).

Additionally, FV subjects provide the following features:

- A Devicetree structure describing the subject's configuration.

Virtual PSCI

LynxSecure supports the Power State Coordination Interface (PSCI) in all types of subjects. The Virtual PSCI provides the following functions:

- PSCI_VERSION
- CPU_OFF
- CPU_ON
- AFFINITY_INFO
- SYSTEM_OFF
- SYSTEM_RESET

The CPU management functions such as CPU_ON and CPU_OFF functions affect subject VCPUs rather than the physical CPUs. The SYSTEM_OFF and SYSTEM_RESET functions affect the calling subject rather than the entire system.

The Virtual PSCI uses the SMC instruction firmware call conduit.

NOTE: The Virtual PSCI is the only way of managing the VCPU state in an SMP subject that is common across all the supported ARM targets.

System Clock Management

On many ARM targets, physical devices have associated clocks providing a clock input signal necessary for the device to operate. A clock is usually a set of registers in a platform clock controller and may be thought of as a special kind of a physical device. The clocks form a hierarchy, where each child clock takes the clock output of its parent as its clock input and modifies it in some way to produce its clock output. For example, a clock may divide the input frequency by a factor programmed in the clock's registers.

The root of the clock tree usually is a fixed frequency source such as an oscillator. If there are multiple fixed frequency sources in the system, the clock hierarchy may have multiple roots.

The hierarchical structure of system clocks results in a situation where clocks are shared between multiple physical devices. If not the device clock itself, then some of its parent clocks are typically shared. This means that LynxSecure must restrict subject access to clocks shared by devices that are assigned to different subjects. Otherwise, those subjects would have an unauthorized information flow channel between them through the clock's registers. Also, unauthorized interference with the other subjects' devices is possible via changing the shared clock frequency.

If a physical device is assigned to a subject, the subject receives the permission to read the corresponding clock and its parent clocks all the way to the clock tree root. However, it only receives the permission to write those clocks in that set that are not shared with any other subject. Attempts to write to shared clocks are denied and an Audit message reporting an information flow policy violation is logged.

NOTE: LynxSecure deviates from its regular "at most one owner subject for any physical device" and "no read-only devices" policies with respect to system clocks.

Additionally, LynxSecure prevents changes to the system clock hierarchy. Some targets allow dynamically switching a clock between a set of possible parent clocks. As such operations would affect the subject clock ownership during the subject runtime, LynxSecure always denies them. System integrators must ensure that the clock hierarchy is set to its final static state and any shared clocks are programmed with their final frequency before LynxSecure is started on that system.

Host-to-PCI Bridge Assignment to Subjects

On the ARM platform, LynxSecure supports the assignment of the target's host-to-PCI bridge to a subject. This effectively assigns all PCI devices behind that bridge to the same subject.

ARM Limitations

The following features are notably not supported on the ARM platform, as compared to other platforms:

- Assignment of individual PCI devices to subjects.
- The Virtual Device Server (VDS) subject.
- Any virtual devices requiring the presence of a VDS subject.

Speculative Execution Vulnerability Mitigations

Recently, a number of CPU security vulnerabilities has been discovered that involve side channels created by speculative code execution. These are known by the names of Meltdown, Spectre and L1TF (Foreshadow). See the CPU manufacturer information pages for explanations of these vulnerabilities and lists of the vulnerable CPU models:

- Intel: https://software.intel.com/security-software-guidance
- Arm®: https://developer.arm.com/support/arm-security-updates/speculative-processor-vulnerability

In this LynxSecure release, neither the hypervisor nor the subjects are vulnerable on the ARM platform, because the supported CPU model (Cortex-A53) is not vulnerable. The rest of this subsection applies to the x86 platform only.

If the physical CPU running LynxSecure is vulnerable to a speculative execution vulnerability, the subjects are also vulnerable, as far as information protection between the guest user space and the guest kernel and between different guest processes running in the same subject is concerned. We call these vectors of attacks *subject-to-self*. Due to the nature of speculative execution vulnerabilities, LynxSecure hypervisor has no means of mitigating subject-to-self attacks. Therefore, it's the guest system software running in the subject that is responsible for the appropriate mitigation measures. To assist in that, LynxSecure makes manufacturer-provided hardware mitigation techniques such as Indirect Branch Restricted Speculation (IBRS) available to subjects.

The rest of this subsection discusses the speculative execution vulnerabilities with respect to guest code attacking hypervisor code (*subject-to-hypervisor*) and guest code in one subject attacking guest code running in a different subject on the same physical CPU core (*subject-to-subject*). Subjects running on different physical CPU cores cannot attack each other using the speculative execution vulnerabilities.

Meltdown and Spectre

LynxSecure hypervisor is not vulnerable to subject-to-hypervisor Meltdown attacks, because it doesn't share any page tables with the subject. Meltdown also cannot be used for subject-to-subject attacks. Thus, Meltdown attacks can only be subject-to-self.

LynxSecure hypervisor includes some subject-to-hypervisor and subject-to-subject Spectre attack mitigations. Whether the mitigation is enabled depends on the vulnerability variant and some other factors, which will be described below.

Table 3-4: Spectre Mitigation by Variant (x86)

Spectre Variant	Brief Name	Mitigated?
1	Bounds Check Bypass	Yes
1.1	Bounds Check Bypass for Stores	Yes
2	Branch Target Injection	See Table 3-5 (page 36)
3	Rogue Data Cache Load, a.k.a. Meltdown	Not vulnerable (subject-to-self only)
3a	Rogue System Register Read	No; may be fixed by a microcode update
4	Speculative Store Bypass	Not vulnerable (subject-to-self only)
SpectreRSB	Attack using the Return Stack Buffer	Yes

Spectre variant 2 mitigations may require hardware mitigation techniques provided by microcode updates from the CPU manufacturer; CPU models released in year 2018 and later may already include the necessary support without the need for an update. For brevity, we refer to these techniques as "IBRS", which stands for Indirect Branch Restricted Speculation, although IBRS is just one of the new mitigation techniques provided by the microcode update. IBRS is also the name of the LynxSecure configuration option which controls the level to which these techniques are used by LynxSecure. The table below describes the Spectre v2 mitigation level depending on LynxSecure configuration and IBRS presence.

Table 3-5: Spectre v2 Mitigation (x86)

Configuration	Subject-to-Hypervisor	Subject-to-Subject	IBRS available to subjects?
No IBRS or IBRS disabled	Limited	No	No
IBRS present and in "subject" mode (default)	Limited	Yes	Yes
IBRS present and in "full" mode	Full	Yes	Yes

The "Limited" mitigation uses generic programming techniques such as retpolines, which come at a moderate performance cost. It may still be possible to compromise this mitigation, although no known exploits exist at the time of this writing. The "Full" mitigation uses IBRS in hypervisor mode, which provides a greater level of protection. Note that IBRS and other manufacturer-provided hardware mitigation techniques come with significant performance costs. The configurability lets the system integrator choose the balance between performance and security according to the use case.

L1TF

L1TF (Level 1 cache Terminal Fault) is a CPU security vulnerability found in x86-compatible CPUs manufactured by Intel only. See the manufacturer's page for the explanation of this vulnerability and the list of affected CPU models: https://www.intel.com/content/www/us/en/architecture-and-technology/l1tf.html

The L1TF vulnerability lets guest code running in subjects read host system memory contents that they have not been authorized to read. The data needs to be in the CPU's L1 data cache in order to be accessible with the L1TF attack.

LynxSecure provides mitigations for subject-to-hypervisor and subject-to-subject L1TF attacks occurring on the same CPU hardware thread. LynxSecure doesn't provide mitigations for L1TF attacks on sibling (belonging to the same CPU core) Hyper-Threading threads, because no acceptable mitigation exists. Users should consider not using Hyper-Threading on systems with CPUs vulnerable to the L1TF attack. However, there are considerations that may make the threat level from Hyper-Threading acceptable in some scenarios. They are discussed below.

Like in the Spectre case, L1TF mitigation has a significant performance cost, and LynxSecure provides several levels of mitigation so that the system integrator can choose the right balance between performance and security according to the use case:

- "none": no L1TF mitigation, no performance cost
- "conditional": limited L1TF mitigation [1] (conditional L1 flushing when switching from hypervisor to guest mode), moderate performance cost
- "always": full L1TF mitigation (always flush L1 cache when switching from hypervisor to guest mode), high performance cost

CPU microcode updates may improve the performance of the L1 cache flushes used in the mitigation, so we recommend installing those updates if L1TF mitigation is used.

Because the data needs to be in the L1 cache and sibling Hyper-Threading threads share the L1 cache, it's possible for sibling threads to attack each other. Therefore, a configuration where two sibling threads simultaneously run different subjects is susceptible to two kinds of L1TF attack: subject-to-subject (if performed when the other thread is executing subject code) and subject-to-hypervisor (if performed when the other thread is executing hypervisor code). LynxSecure has no means to mitigate subject-to-subject L1TF attacks in this case, so such a configuration should be avoided unless both subjects are trusted.

If the sibling threads run different VCPUs of the same subject, however, only subject-to-hypervisor attacks are possible between those threads. The subject-to-hypervisor L1TF attacks have a low chance of hitting secret data, because it is highly unlikely that the hypervisor brings secret data into the L1 data cache. The following types of data that may contain secrets needs to be considered in this case:

- Hypervisor secrets not associated with a particular subject[2]
- Any data associated with another subject that the attacking subject is not authorized to read
- Data associated with the attacking subject that the attacking subject is not authorized to read

The first two of these data types are normally never accessed by the hypervisor during the runtime when the attacking subject has the scheduling time slice. Note that the hypervisor doesn't have the subject RAM of any subject mapped into its address space. Certain actions, such as authorized hypercalls targeting another subject, may bring the other subject's data into the L1 data cache, although it is unlikely that the data is secret. Note that regular L1TF mitigations flush that data from the L1 cache before returning from hypervisor to subject code execution, but it may still be briefly available to the concurrently executing sibling thread, which is why Hyper-Threading presents a high security risk.

The last type of sensitive data, data associated with the current (attacking) subject that the attacking subject is not authorized to read, is likely to be brought into the L1 cache while executing in hypervisor mode. However, we believe that the security risk presented by that data is low.

Therefore, a configuration where sibling hardware threads of a CPU vulnerable to L1TF run the same subject may be acceptable from the security standpoint in use cases where security requirements are low. In order to reduce the risks, consider the following measures:

- Avoid including secret data in the LynxSecure configuration vector (HCV), as it will be present in the hypervisor address space during runtime.
- Avoid giving an untrusted subject occupying multiple hardware threads permission for actions on other subjects. Such actions include: sending synthetic interrupts, managing subject state, etc. Also avoid giving the subject virtual devices managed by VDS, if those devices are interrupt-driven.

[1]We believe that the mitigation is sufficient to prevent subject-to-subject and subject-to-hypervisor L1TF attacks within the same hardware thread, but it hasn't been formally proved.

[2]The hypervisor normally doesn't have any secret data. However, if the user includes secret data such as cryptographic keys in the LynxSecure configuration vector, it will be present in the hypervisor address space at runtime.

ZynqMP Features (ARM)

Virtual PMU Firmware

On the Xilinx® Zynq® Ultrascale+™ MPSoC family boards, LynxSecure supports the Virtual PMU firmware (PMUFW) in all types of subjects. The Virtual PMUFW conforms to the API of the Xilinx PMU firmware found on these boards. It provides the following functions:

- PM_GET_API_VERSION
- PM_INIT_FINALIZE
- PM_SYSTEM_SHUTDOWN
- PM_GET_NODE_STATUS
- PM_REQUEST_NODE
- PM_RELEASE_NODE
- PM_SET_REQUIREMENT
- PM_SET_MAX_LATENCY
- PM_RESET_ASSERT
- PM_RESET_GET_STATUS
- PM_MMIO_WRITE
- PM_MMIO_READ
- PM_GET_CHIPID

All other PMUFW functions are unsupported.

LynxSecure performs information flow control on the Virtual PMUFW calls. For example, a subject is only allowed to manage the power state of or reset physical devices that are assigned to it.

Subjects may use the PM_MMIO_READ and PM_MMIO_WRITE functions to access the system clocks. Information flow control is performed on these calls. See section “System Clock Management” (page 34) for more details.

APPENDIX A Glossary

Absolute Time
: The number of seconds since the epoch (Midnight, 1 January 1970 Universal Coordinated Time). Also referred to as Wall Time.

Authorized Subject
: has the ability to invoke privileged functions in the SKH Runtime Software. For example, an authorized subject can be given permission to request the SKH to startup or shutdown another subject. An authorized subject should be trusted by the system integrator to make proper use of its privileges, but is not trusted as part of SKH itself.

Autoconfig Tool
: A LynxSecure® command line utility to obtain target system hardware data and generate an HCV based on that data and the desired configuration passed as arguments.

Binary Configuration Vector
: A binary image of the unique configuration for a given SRP. Whenever the *Configuration Tool* is run, it produces an SRP which contains a Binary Configuration Vector.

Block Device Emulation (BDE)
: Provides virtual IDE/AHCI controller interfaces to a fully virtualized subject.

Bootloader
: When used in this generic sense, refers to the software that bootstraps the LynxSecure runtime software on a target platform.

Cache Allocation Technology (CAT)
: A feature available on Intel® processors. Allows the user to fine-tune the system performance by assigning portions of the last level cache (shared by all processor cores in a CPU package) to a particular task.

Capacity Bitmask (CBM)
: An integer number defining the amount of a system resource assigned to a particular task. The term is used in CPU resource partitioning technologies such as Intel RDT and CAT.

Certificate Authority (Certification Authority, CA)
: An entity in charge of issuing certificates to other entities. A root CA is the one implicitly trusted by the end-user software; the certificate issued by the root CA (root certificate) is self-signed. Certificates for intermediate CAs are signed by either the root CA or another intermediate CA.

Class of service (COS)
: An integer number identifying a CBM. The term is used in Intel CPU resource partitioning technologies such as RDT and CAT.

Configuration Tool
: A command line utility that converts a *Human Readable Configuration Vector* into the *Binary Configuration Vector*. It also enforces correct syntax and semantics in the data formats. Not to be confused with the *Autoconfig Tool*.

Configuration Vector
LynxSecure is configured using an offline configuration tool. The configuration vector is instantiated in multiple formats when in various stages of the system lifecycle: Human-Readable Configuration Vector (HCV), Binary Configuration Vector (BCV), and Runtime Configuration Vector (RCV).

Exported Resource
An exported resource is a resource made available by the Separation Kernel-Hypervisor (SKH) to one or more subjects.

External Interrupt
An interrupt generated by a real (non-virtual) device.

External Network
A network for which the LynxSecure platform is a node. LynxSecure interfaces with this network through a physical resource called a network interface card (NIC).

Full Virtualization
A mechanism by which an operating system, designed to run directly on a computing platform, can execute as a subject guest operating system without modification (i.e., without knowledge that it has been virtualized). Some implementations of full virtualization still use paravirtualized device drivers; but the operating system kernel itself remains unmodified.

Fully Virtualized Subject (FVS)
A component in LynxSecure responsible for platform initialization performed before executing the BIOS.

Guest Operating System (GuestOS)
An operating system executing in a virtualized context, rather than directly on the hardware with direct access to the hardware and privileged execution operations. When a GuestOS runs in LynxSecure it is also referred to as a Subject.
See Also Subject.

Human Readable Configuration Vector
An Extensible Markup Language (XML) file conformant to the XML Schema which defines LynxSecure configuration data.

Hypercall
An explicit software trap from subject code to the Hypervisor is a hypercall. A hypercall is similar to a system call on a non-virtual operating system environment. A hypercall allows indirect access to shared resources provided by the hypervisor.

Hypervisor or Virtual Machine Monitor (VMM)
For LynxSecure, software that runs directly on the hardware to provide virtualization for execution of multiple operating systems. In LynxSecure separation kernel and hypervisor are used interchangeably since both are realized at the same architectural layer.

Image
A contiguous collection of data and executable bits which makes up the LynxSecure on an executable hardware platform. The image is first deployed or downloaded as a file or a collection of files. When the hardware is powered on, the image is loaded from a storage medium, verified, and executed.

Installation and Packaging Toolset
The packaging toolset creates the runtime used to install LynxSecure onto the target platform. The installation tool is a runtime tool which installs LynxSecure onto a target system.

Installation Toolset
See Installation and Packaging Toolset.

Inter-Subject Communication Mechanisms

provided by the SKH allowing for subjects to communicate through a variety of well-defined interfaces. Each interface is provides different capabilities appropriate for different communication factors.

Interrupt

A system event, typically asynchronous, generated by a system device. An interrupt is generally used to indicate readiness of resources in an asynchronous manner. Interrupts usually have a well-defined source, path, and destination. Interrupts are usually "vectored," meaning that the control of some components of the source, path, and destination of a given interrupt are parameterized elements of an interrupt driven system.

Major Frame

A major frame consists of one or more minor frames. A major frame consists of a fixed amount of System Clock Tick (SCT) intervals. The sum of all SCT intervals assigned to each minor frame within a major frame equals the fixed amount of SCT intervals assigned to the major frame. A major frame is processor specific. Though two major frames may have identical minor frames, each major frame is distinct in that it is assigned to a unique processor and scheduling policy. In other words, a major frame is uniquely specified by its encompassing scheduling policy and assigned processor.

Memory region

A Memory region allows LynxSecure Security Kernel (LSK) to describe special handling required during the system boot up and during the working of the system.

Minor Frame

A minor frame consists of a subject id and a fixed number of SCT intervals. A minor frame binds a subject to a processor for a fixed amount of SCT intervals specified by the minor frame. Each minor frame belongs to a major frame.

Native

This is an operating system execution mode where the operating system has direct access to the hardware and privileged instructions without intermediary layers (such as a hypervisor or separation kernel). This is the typical way operating systems are deployed (no virtualization).

Paravirtualization

An approach of virtualization technology in which a guest OS is modified and recompiled prior to installation inside a virtual machine. A paravirtualized guest OS may run near native mode speed on the target hardware. This is also known as co-operative virtualization.

Platform Configuration Registers

A set of hardware registers provided by the Trusted Platform Module (TPM). These registers cannot be directly written to by software; instead, software *extends* a value into a register. The new value of the register is then a hash function of the old value and the extended value.

Processor Core

Modern processors may contain multiple processor cores. This is the smallest hardware processor resource that can be scheduled by the LynxSecure scheduling policy.

Read-Only Page (RO Page)

A per-subject memory structure describing the subject's configuration. Each subject can only access its own RO page, and the access is read-only. Despite the name, the structure is typically longer than one memory page. The RO page structure is defined in the `api.h` header file.

Reboot/Restart

Restarting a subject or the entire LynxSecure system under software control, without removing the power or (directly) triggering a reset line. It usually, though not always, refers to an orderly shutdown and restarting of the machine.

Resource

Resources are the totality of all hardware, firmware and software, and data that are executed, utilized, created, or protected by LynxSecure runtime software.

Resource Director Technology (RDT)

A feature available on Intel processors. Allows the user to fine-tune the system performance by monitoring and assigning a particular system resource (portion of the CPU cache, set the memory bandwidth, etc.) to a particular task.

RMRR

An RMRR memory region corresponds to a Reserved Memory Region lying in host physical memory. RMRRs are associated with devices that need a range of host physical memory to be identity-mapped in the DMA redirection tables for the device. These regions are typically used by USB and on-board graphics devices. RMRRs are not visible to the CPU; they are only accessible by device DMA. RMRRs may overlap with other memory region types, and RMRR mappings (memory flows) may overlap with mappings of other memory region types; in this case, the RMRR takes precedence over the other regions for DMA transfers to the overlapping address range. RMRR locations are determined by the host BIOS and reported to other software via ACPI tables.

Robustness (Medium Robustness and High Robustness)

A characterization of strength and assurance of security functions defined in Department of Defense Instruction 8500.2.

Scheduling Policy

A configuration encompassing a description of the subjects, and time frames involved and of how much time each subject executes upon all utilized processor cores on the target platform. A scheduling policy contains one major frame per processor core. A scheduling policy is the highest level of abstraction regarding scheduling. A scheduling policy consists of one or more major frames. Each major frame within a distinct scheduling policy is processor specific, meaning that each major frame is assigned to one and only one distinct processor. Only one scheduling policy is active at a time. A configuration vector can contain multiple scheduling policies.

Self-Assisted Virtualization (SAV)

A component in LynxSecure responsible for emulation of virtual devices. SAV executes in the context of the subject itself.

Separation Kernel

A software layer responsible for providing secure, non-bypassable isolation and separation of platform resources amongst entities wishing to utilize the platform resources. In LynxSecure, separation kernel and hypervisor are used interchangeably since both are realized by the same architectural component.

Separation Kernel / Hypervisor

A portion of the LynxSecure system responsible for providing separation kernel and virtualization (hypervisor-VMM). While providing different functionality, the terms are often combined due to their close intertwining into overlapping software layers within the overall architecture. Includes hardware and/or firmware and/or software mechanisms whose primary function is to establish, isolate and separate multiple *subjects* and control information flow between the subjects and exported resources allocated to those subjects.

SKH Runtime Package

A data format stored as a binary image which contains the runtime software and configuration to be loaded into memory by the *SKH* bootloader.

Subject

A subject is an active entity that causes operations to be performed, and executes within the context of the LynxSecure *Separation Kernel / Hypervisor*. Since the only information flows which are allowed are between subjects and exported resources, a subject is also considered a resource so that subject to subject information flows are possible.

Subject Execution State
A descriptive quality of the subject's current state in terms of its software execution, from the perspective of the SKH. Values include Running, Stopped and Suspended.

Synthetic Interrupt
Interrupts that are either generated internally by SKH and injected into a subject or initiated by a subject with right permissions and injected into another subject. The subject initiates the interrupt using a hypercall.

System Clock Tick
An event that causes each scheduler to be run in a near synchronous manner. SCTs occur at a periodic rate. The time between two SCTs is called a SCT interval. A SCT is an interrupt that is generated by a hardware timer. Also known as System Tick.

System Integrator
Develops code and configurations specific to a program or application to meet the needs of an end user.

System Timestamp Counter
A hardware dependent system counter, incrementing once for each specified hardware clock, that is started at, or soon after, power up or system reset, and which is not stopped or reset until the system is powered down or reset.

Target Board
The realized instance of a circuit board which executes the software.

Target Platform
The realized instance of a hardware platform which executes the LynxSecure software composed of a target board and many other hardware elements.

Timebase Calibration Value
A configuration value which describes length of one timebase cycle (i.e. the time between two subsequent timebase register increments) in nanoseconds. The timebase register is a platform register that increments at a constant speed. A system with a 2 GHz timebase register increment frequency would have a nominal Timebase Calibration Value of 0.5 nanoseconds. The subject that calibrates the system clock actually vary this value, speeding up and slowing down the apparent rate of time passage to synchronize the system to an external timebase in a software analogue of a phase locked loop.

Trusted Platform Module
A standard for a secure cryptoprocessor. LynxSecure bootloader and RIF can use the TPM device for reporting the metrics of the software modules, thus participating in the root of trust.

USB Emulation
Emulates USB controllers for fully virtualized subjects or provides a virtual USB controller for paravirtualized subjects.

Virtual KMA Switch
Provides support for switching the keyboard, mouse, audio and USB devices control between the fully virtualized subjects using key combinations.

Virtual KVM Switch
Provides support for switching the keyboard, mouse, video, audio and USB devices control between the fully virtualized subjects using key combinations.

Virtual Machine (VM)
The software that creates a virtualized environment upon which guest software executes. The following phrases are synonymous: "the Linux® and Windows® Guests," "the Linux and Windows guest OSs," and "the Linux and Windows virtual machines."

Virtual Network

An emulation of a network topology and node interfaces through software and wholly contained within the LynxSecure hardware platform itself. It uses well-known network protocols rather than custom network protocols. The virtual network can interface with an external network through software configuration and a physical NIC.

Virtualization

An added level of indirection and abstraction that hides physical characteristics of a resource from the way in which subjects use it. For LynxSecure, this allows multiple subjects (software) to utilize resources (hardware) where normally only a single subject could interface with the resource. This enables multiple operating systems to concurrently utilize a single hardware platform.

Wall Time

See *Absolute Time*.

Acronyms

Acronyms	Description
ABI	Application Binary Interface
ACPI	Advanced Control and Power Interface
AHCI	Advanced Host Controller Interface
API	Application Programming Interface
APIC	Advanced Programmable Interrupt Controller
BAR	Base Address Register
BCV	Binary Configuration Vector
BDE	Block Device Emulation
BIOS	Basic Input/Output System
BIT	Built-In Test
CA	Certificate Authority
CBIT	Continuously Running Built-In Test
CD	Compact Disc
CD-ROM	Compact Disc Read Only Memory
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DMA	Direct Memory Access
DVD	Digital Versatile Disc
FLR	Function Level Reset
FVS	Fully Virtualized Subject (LynxSecure component)
GPA	Guest Physical Address
HCV	Human Readable Configuration Vector
HPA	Host Physical Address
I/O	Input/Output
IBIT	Initiated by an Event Built-in Test
I/O MMU	Input Output Memory Management Unit
IP	Internet Protocol
IPI	Inter Processor Interrupt

Acronyms	Description
IRQ	Interrupt Request
KMA	Keyboard Mouse Audio
KVM	Keyboard Video Mouse
LPC	Low Pin Count (bus)
LSA	LynxSecure Application
LSK	LynxSecure Separation Kernel
MAC	Media Access Control
MSI	Message Signaled Interrupts
NAT	Network Address Translation
NIC	Network Interface Card
OS	Operating System
PCR	Platform Configuration Register
PCI	Peripheral Component Interconnect
PCIe	PCI Express
PSCI	Power State Coordination Interface
PXE	Preboot Execution Environment
RIF	Runtime Initialization Function
RMRR	Reserved Memory Range Region
SATA	Serial Advanced Technology Attachment
SAV	Self-Assisted Virtualization (LynxSecure component)
SBIT	Startup Initiated Built-In Test
SCT	System Clock Tick
SESM	Subject Execution State Manager
SKH	Separation Kernel Hypervisor
SMP	Symmetric Multiprocessing
SMT	Symmetric Multithreading
SR-IOV	Single Root I/O Virtualization
SRP	SKH Runtime Package
SW	Software
TCP	Transmission Control Protocol
TSC	Time Stamp Counter
USB	Universal Serial Bus
VDS	Virtual Device Server
VGA	Video Graphics Array
VF	(SR-IOV) Virtual Function
VM	Virtual Machine
VT	Virtualization Technology
VT-d	Intel's VT for devices (includes I/O MMU)
XML	Extensible Markup Language