Open XT™ Architecture Guide High-assurance isolation & security for virtual environments

Publication date 12 June 2014 (build 1550)

Table of Contents

Chapter 1. About this document

In this document we describe the software architectures implemented in OpenXT features. We cover the relevant design and implementation details as well as the technologies on which they depend. Where appropriate, we identify security critical technologies and design decisions to assist and facilitate 3rd party analysis. Particular attention is paid to the high-level security goals of the system, how these goals are achieved through software architecture and the relative strength of the mechanisms employed.

OpenXT is a system composed of both open and closed source components. As is the case with open source technologies not all of the architectures implemented originate from within OpenXT so we are not always the definitive source. Where appropriate we reference external documentation that the reader may consult for additional details.
This document is intended for software architects and security evaluators in their efforts to determine the appropriateness of the OpenXT platform for their respective use-cases. After reading this document you will have the requisite knowledge to evaluate the OpenXT architecture for use in your environment. The architectures detailed in this document are subject to change without notice. Please contact OpenXT with any comments or corrections to this document.

Chapter 2. Security Goals

Client virtualization with XenClient aims to provide end users with an experience that meets the expectations set by traditional client computing. We aim to provide the performance, graphics, networking and peripheral compatibility that users expect. This goal underpins all design and architecture in OpenXT Client Virtualization products.
The architecture of OpenXT is unique in that we extend this goal to formally include security properties. By.their nature both client and server virtualization platforms add a layer of highly privileged software that acts as an abstraction layer between the hardware and guest VMs. This additional software must be built in such a way as to introduce no additional threats to guest VMs beyond those posed by the hardware itself. This security goal is a simple extension of our desire to provide users with an experience indistinguishable from native client computing: We aim to provide the security properties that users have come to expect from an OS running on "bare metal".

This high-level security goal must be decomposed and made concrete to be useful in an architectural analysis. The remainder of this document will tie architecture and implementation details to the following goals:

2.1. Workload Isolation

Isolating guest VMs from the actions of other guests is a primary goal of the Xen Hypervisor. Great pains have been taken to create a flexible and fair scheduler in Xen and this work has been documented[*] by the community. Maintaining performance separation between guest OS's such that they appear to be the only OS running on the platform is paramount but performance is only half of the story. The Xen scheduler prevents denial of service caused by CPU resource consumption but OpenXT also implements mechanisms to prevent guest interference through other channels.

2.2. Mediated Access

Where Section 2.1: "Workload Isolation" professes our goal of isolating system components to prevent undesired interaction or tampering, this section is an admission that cooperation between VMs is often required to do meaningful work. On the surface this need may seem to contradict Section 2.1: "Workload Isolation" but it is really complementary. Strong isolation is a prerequisite of mediation as without isolation, mediation can not be guaranteed. In this way we depend on establishing isolation first and then selectively allow communication between VMs through mediated channels. This is a core property of mandatory access control and must be a pervasive design principle throughout the system.

2.3. Disaggregation of Security Critical Function

An architectural observation from early in the development of Xen was that the strong isolation properties that are applied to guest VMs or 'workloads' can equally be applied to core system functions to achieve an even stronger system security posture. The act of moving security critical function traditionally in the Xen dom0 to separate isolated VMs is termed 'disaggregation'. An academic exercise to disaggregate the Xen dom0 as far as possible has been done in the past` Later sections of this document will explicitly address the functions exported from dom0 and the protections put in place to protect the platform from them in the event of a compromise.

2.4. Strength of Mechanism

All of these goals aim to protect guest VMs from each other and from a compromised platform. The term 'strength of mechanism' is intended to describe the relative strength of a mechanism from those available. Thus the mechanisms used to achieve our security goals and described throughout this guide, should be the strongest mechanisms

[*]http://www.xenproject.org/component/mtree/research/115-comparison-of-the-three-cpu-schedulers-in-xen.html
[†]http://www.cs.ubc.ca/~andy/papers/xoar-sosp-final.pdf

available. In the event a relatively weaker mechanism is chosen over a stronger one the selection must be sufficiently justified. Wherever possible we aim to use mechanisms available in hardware both for strength and performance reasons.

2.5. Integrity and Measurement

The past fifteen years of numerous, stealthy exploits have taught the security community that knowing when you've been compromised is the most valuable piece of information you can have. The cycle of software attempting to protect itself from other equally privileged software was identified by industry experts. Under the auspices of the Trusted Computing Group (TCG)[‡], they attempted to break this cycle through the design and integration of a hardware mechanism that software could rely upon to accurately reflect the integrity state of the platform.

This mechanism, specifically designed to resist tampering at the software level, became known as the Trusted Platform Module (TPM)[4]. By using the TPM as a hardware root of trust for measurement (RTM), CPU manufacturers like Intel® have been able to provide support for bootstrapping trustworthy software environments directly in hardware. Properly using the available hardware mechanisms enables software systems to load into a pristine environment and have an unforgeable record of the integrity state of the system.

These integrity measurements can then be used as the basis for a number of mechanisms designed to ensure the system will only disclose sensitive data to software in the appropriate state. Integrity reporting mechanisms are perfectly aligned with our goal of posing no additional threat to guests.

2.6. Extensibility

Strong isolation and a disaggregated design is useful as part of the core platform but these mechanisms should be available for use in as many contexts as possible. With this in mind we aim to provide the benefits of these properties to 3rd parties such that their software can benefit from the same protections as the XT platform itself. Leveraging this extensible design, applications that had previously relied on week separation mechanisms provided by traditional operating systems can benefit from the security properties of OpenXT. Thus some 3rd party applications are able to integrate with the OpenXT platform and no longer risk compromise or circumvention by OS malware.

[‡]http://www.trustedcomputinggroup.org/
[4]http://www.trustedcomputinggroup.org/developers/trusted_platform_module/

Chapter 3. The OpenXT Measured Launch Process

When enabled, OpenXT Measured Launch measures security-critical components in the Engine Control Domain. We rely heavily on features in the hardware to assist in measuring software components early in the boot process and to provide protection from tampering by other system software. This requires support for Intel® vPro™ hardware, specifically Intel® Trusted Execution Technology® (TXT). A good introductory white paper to TXT is available on the Intel® website: here[*].

The remainder of this section briefly discusses foundational trusted computing technology required for understanding the Measured Launch process. These descriptions will be brief, with links to external documentation that can be read as background. We then go on to discuss the details of the Measured Launch process as implemented in OpenXT.

3.1. Software Measurements

Software Measurements in their most simple form are cryptographic hashes of executable files or configuration data. Individually a measurement of a piece of software is abstract, but in the context of the operating system boot sequence, a series of measurements that describes all software involved in the platform boot can be established. This measurement chain begins with the earliest software (BIOS & firmware) which measures itself and then the next component in the boot chain before transferring control.

As the boot process progresses, execution and measurement continues with each piece of software measuring the next before transferring control. We call the series of measurements produced during this process the "chain of trust" and the rationale behind the architecture is explained thoroughly in the TPM Main Specification: Design Principles[†].

3.2. Platform Configuration Registers & Sealed Storage

Sealed Storage is the name of the mechanism provided by the Trusted Platform Module (TPM) to protect secrets from malicious software. The TPM has storage locations called Platform Configuration Registers (PCRs). As the system executes, these PCRs are used to hold measurements (SHA1 hashes) of security relevant software. OpenXT is configured in such a way that measurements for the security-relevant components are stored in PCRs before they are executed, creating the "chain of trust" discussed above.

Sealed Storage is a TPM function that allows secrets to be protected by the TPM. A secret is said to be "sealed" to a set of PCR values when the TPM will only divulge the secret if the values present in the PCRs are the expected values. A more detailed discussion of PCR functions and Sealed Storage can be found here[‡].

3.3. Intel® Trusted Execution Technology® (TXT)

OpenXT uses Intel® TXT for a hardware-based, dynamic root of trust for measurement (DRTM) of the OpenXT launch environment. TXT is invoked prior to OpenXT boot, suspend, restart and shutdown. TXT uses open-source (tboot) and closed-source (SINIT) chipset-specific software that is developed and maintained by Intel®. Successful functioning of TXT also depends on processor support, the quality of an OEM's BIOS implementation, and Trusted Platform Module (TPM) firmware.

As part of executing the Intel® tboot program, the security relevant early boot components are measured. tboot is executed by the bootloader, before the OpenXT hypervisor and dom0 kernel. It performs a number of chipset operations which protect the OpenXT hypervisor from tampering by other system software.

[*]http://www.intel.com/content/www/us/en/trusted-execution-technology/trusted-execution-technology-security-paper.html
[†]http://www.trustedcomputinggroup.org/resources/tpm_main_specification
[‡]www.rsa.com/rsalabs/technotes/tpm/sealedstorage.pdf

Before transferring control to OpenXT, tboot measures the Xen hypervisor, the dom0 kernel, the dom0 initramfs, and all other modules loaded by the bootloader. These measurements are stored in the TPM PCRs as part of the chain of trust. In the next section Section 3.4: "OpenXT PCR Usage" we discuss the PCRs used to secure OpenXT and which measurements they contain. We then go on to describe how the measured launch and the TPM sealed storage mechanism are used to protect the OpenXT system.

3.4. OpenXT PCR Usage

We give only a brief description of the measurements stored in each PCR in this document. Where appropriate, a reference to the definitive documentation is given.

PCRs for BIOS configuration

The contents of these PCRs are documented in the *TCG PC Client Specific Implementation Specification For Conventional BIOS* document.

PCR[0]
Used to store the Core Root of Trust for Measurement (CRTM), POST BIOS, and Embedded Option ROMs

PCR[1]
Platform Configuration

PCR[2]
Option ROM Code

PCR[3]
Option ROM Configuration and Data

PCRs for OpenXT Engine root filesystem measurement

This PCR is extended with the SHA1 hash of the block device that contains the Engine root file system. This is a bulk measurement of all software running in the OpenXT Engine. The measurement is taken by the initramfs.

PCR[15]
OpenXT Engine root filesystem

PCRs specific to Intel® TXT and the tboot program

The measurements contained in these PCRs are defined in the *Intel® MLE Developers Manual* and the documentation packaged with the tboot program.

PCR[17]
TXT data

- SHA-1 hash of BIOS ACM: SinitMleData.BiosAcmID (20 bytes)
- STM opt-in indicator: SinitMleData.MsegValid (8 bytes)
- SHA-1 hash of the STM (or all 0s if opt-out): SinitMleData.StmHash (20 bytes)
- LCP Control Field of used policy (PD or PO): SinitMleData.PolicyControl (4 bytes)
- SHA-1 hash of used policy (or all 0s if chosen not to be extended): SinitMleData.LcpPolicyHash (20 bytes)

- MLE-chosen Capabilities (or all 0s if chosen not to be extended): OsSinitData.Capabilities (4 bytes)

PCR[18]
: SHA1 hash of the MLE

PCR[19]
: SHA1 hash of Xen Hypervisor, Kernel and command line, LCP Policy, initramfs, tboot ACMs and XSM policy

3.5. Protecting Secrets in OpenXT

With the supporting technology laid out, the method for protecting secrets in OpenXT can be put into context. All encryption keys and unique platform configurations are stored in an LVM partition on the guest which is separate from the root filesystem. We call this volume the "config partition." The config partition is encrypted and the key (256-bit AES) is sealed in the TPM using the Sealed Storage mechanism described in Section 3.2: “Platform Configuration Registers & Sealed Storage”.

The PCRs to which this data is sealed are described in Section 3.4: “OpenXT PCR Usage”. These PCRs store measurements taken during the boot sequence by the system BIOS, Intel® TXT and tboot, and the OpenXT Engine. When OpenXT boots, the Engine requires access to the encryption key for the config partition. Without this key, the Engine cannot access system configuration or encrypted guest disks. If the TPMs PCRs contain the expected values, the encryption key can be unsealed and the config partition can be accessed. If any of the measured components are changed on disk, the PCR values will be different and properties of sealed storage in the TPM will render the encryption key inaccessible.

While changes to a measured component will result in the config key being inaccessible, for the sake of usability the administrator is able to intervene. When the unseal operation fails on boot, the administrator can unlock the system with their password. When the prompt is displayed, you can choose for either a single boot to be allowed with the modification, or for the changes to be permanently allowed. Permanently allowing changes requires that the encryption key for the config partition be sealed to the new PCR values. If the changes are not permanently allowed the administrative password will be required on all subsequent boots until the changes are either reverted or the config key is sealed to the new measurements.

3.6. Measured Launch Configuration Process

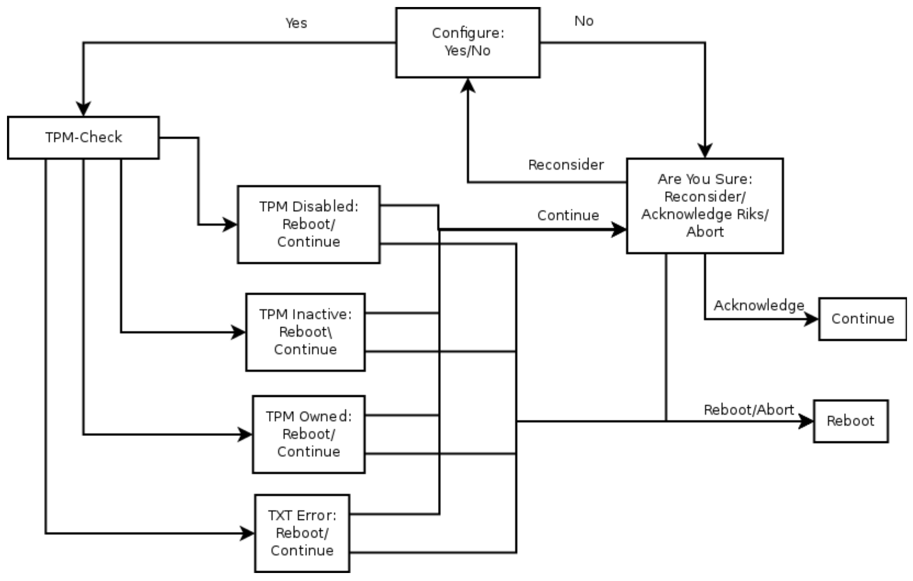
This section documents the control flow for the two methods of enabling OpenXT Measured Launch. This section also documents the method used to unseal the config partition and the remedial steps if the unseal fails.

3.6.1. Enabling Measured Launch for the Control Domain at Installation

The current preferred method for configuring Measured Launch is at system install time. This method has several desirable advantages, the most important of which is that the OpenXT system will be protected by the security offered by Measured Launch on first boot. Configuring Measured Launch after first boot leaves the system unprotected for a window of time. During this time, encryption keys for guests will be stored on disk unencrypted.

The Measured Launch configuration in the installer happens in three stages:

Compatibility Check
: This is likely the most complicated step in configuring Measured Launch, as it probes the platform's TPM and TXT to determine their state while interacting with the user. It is where configuration begins, and it is accessed in the installer through the Advanced installation option. It begins when the user is asked, "do you wish to configure XenClient Measured Launch?" If the user selects Yes, the installer then probes the TPM and TXT logs to determine whether or not the system is in a state where Measured Launch can be configured. A flow chart describing the paths through these screens is below.


Configure: Yes/No
Yes
No
TPM-Check
Reconsider
Continue
Are You Sure: Reconsider/ Acknowledge Riks/ Abort
TPM Disabled: Reboot/ Continue
TPM Inactive: Reboot\ Continue
TPM Owned: Reboot/ Continue
TXT Error: Reboot/ Continue
Acknowledge
Continue
Reboot/Abort
Reboot

Platform Configuration

If the user elects to configure Measured Launch and the compatibility check passes, the OpenXT installer will configure the platform to measure itself and finish the configuration on first boot. This is a procedural set of steps and doesn't warrant a diagram. It includes steps like:

- generating the 256-bit AES key for the config partition and adding it to the LUKS partition
- creating several flags on the file system which instruct the Measured Launch code to perform configuration steps

It is worth noting here that generating a 256-bit AES key often exhausts the entropy pool in the installer. In an interactive installation, the user is prompted to press random keys on the keyboard to repopulate the entropy pool. For unattended installations this isn't always possible.

One possible solution to this problem could be to perform installations on an intentionally noisy network. Interrupts caused by the networking hardware from received packets (ICMP/ping) has the effect of repopulating the entropy pool much like having a user press random keys on the keyboard.

TPM Configuration

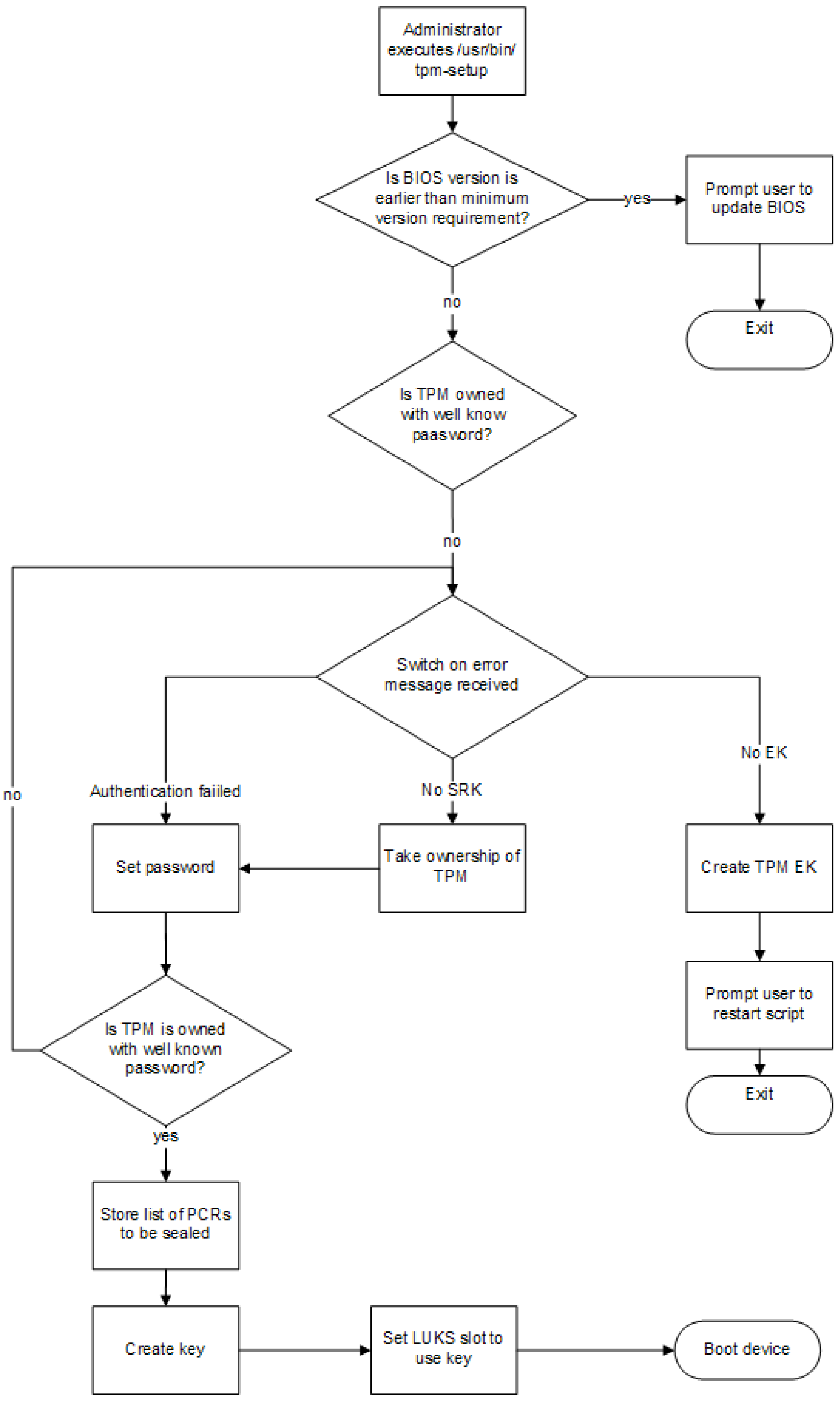
The final step is configuring the TPM itself so that a Measured Launch can be executed. This requires that the installer take ownership of the TPM providing it with an owner password. This password is 15 characters long including alphanumeric and special characters generated randomly. The password is saved to disk encrypted using the platform recovery key, which is itself protected by the administrative password. The TPM SRK is set to the well-known secret as specified by the TCG. This allows the system to automate seal and unseal operations on boot without requiring administrative intervention.

First Boot

On first boot, OpenXT completes the Measured Launch configuration process by performing the first system measurement. See Section 3.6.3: “Measured Launch Process: Measurement Verification on Boot” for details.

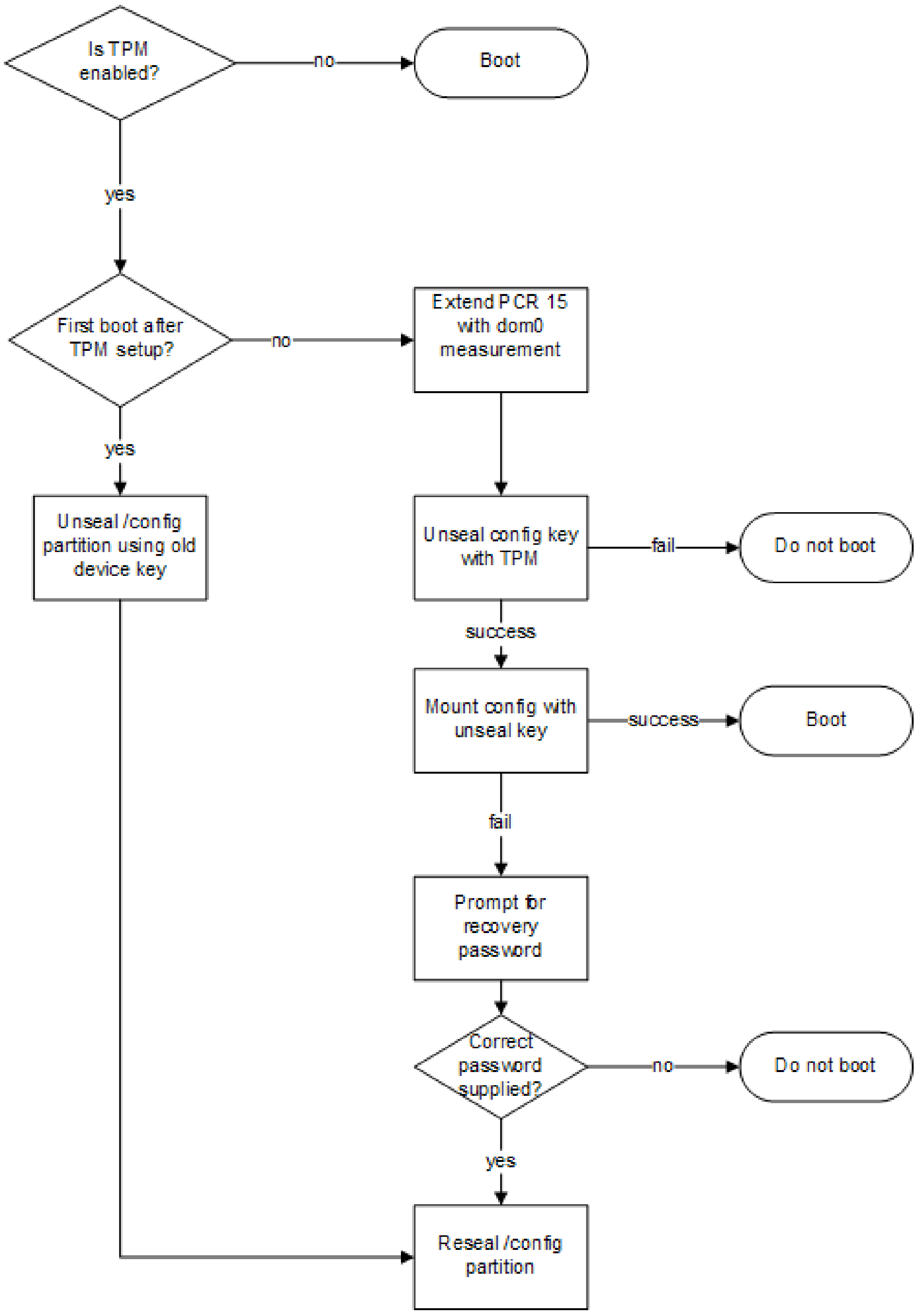
3.6.2. Enabling Measured Launch for the Control Domain After Installation

The following diagram shows the initial Measured Launch setup process when configured on the command line by an administrator through the **tpm-setup** command.


Administrator executes /usr/bin/ tpm-setup
Is BIOS version is earlier than minimum version requirement?
yes
Prompt user to update BIOS
Exit
no
Is TPM owned with well know paasword?
no
Switch on error message received
Authentication failed
No SRK
No EK
no
Set password
Take ownership of TPM
Create TPM EK
Prompt user to restart script
Exit
Is TPM is owned with well known password?
yes
Store list of PCRs to be sealed
Create key
Set LUKS slot to use key
Boot device

3.6.3. Measured Launch Process: Measurement Verification on Boot

The following diagram shows the verification process which occurs at device boot time.



Is TPM enabled?
no
Boot
yes
First boot after TPM setup?
no
Extend PCR 15 with dom0 measurement
yes
Unseal /config partition using old device key
Unseal config key with TPM
fail
Do not boot
success
Mount config with unseal key
success
Boot
fail
Prompt for recovery password
Correct password supplied?
no
Do not boot
yes
Reseal /config partition

3.7. Files Associated With Encrypted VHDs

The following files are associated with the encrypted VHDs of a VM:

- the VM configuration file, `/config/vms/<vm_uuid>`
- the VHD file, `/storage/disks/<disk_uuid>`
- the VHD encryption key, `/config/platform-crypto-keys/<disk_uuid>+other_text`

VHD keys are not themselves sealed to the TPM. Instead they are stored in the config partition. When the device is powered off, VHD keys are protected by the config key which encrypts the config partition and is sealed to the TPM.

VHDs and their associated encryption keys can be copied between machines running the same version of OpenXT. If the machines are running different versions, there are likely to be differences between the VM configuration formats.

3.8. Generation and Use of Keys for Synchronizer VMs

When creating encrypted VHDs, normally the toolstack creates a blank disk and subsequently a key is generated and used for the disk. The key is generated by reading an appropriate number of bits from `/dev/random`. This operation in completed in the Control Domain. When creating a VM using the user interface, when the key is being created, a message box appears indicating the reason for a delay, and suggests that the user move the mouse in order to generate more entropy. Typically, these disks and keys will be used as the basis for images being delivered by Synchronizer.

When a VM is downloaded from the Synchronizer, the default configuration causes the disk to be snapshotted on the client, and for the new, empty, snapshot to be encrypted (unencrypted snapshots can be used by setting the VM configuration item *synchronizer:encrypt_snapshots:false*). Any data written to the disk while the VM is running on the device is stored in this snapshot, and protected by the key that is generated for it if encrypted snapshots are being used. The key downloaded to the client protects only the original disk image.

The key used to encrypt the snapshot is generated when the snapshot is first created. It will be generated in the sync-client VM using random data from the Control Domain. The random data will come from either `/dev/random` (the default) or `/dev/urandom`, both in the Control Domain. These are accessed in the sync-client VM over V4V (inter-VM transport). To use `/dev/urandom` rather than `/dev/random`, the device configuration item *sync-client:use-pseudorandomness:true* can be set. To ensure that `/dev/random` is used, do not set a value for this property.

The VHD keys that are generated for the snapshot are unique to that snapshot. If the disk has the shared property set to *True* on the Synchronizer, then the snapshot will be shared between multiple VMs on the same device. Further discussion of the shared property can be found in the *OpenXT Synchronizer Administrator's Guide*.

When using `/dev/random`, synchronization may pause while the keys for the snapshot disk are being generated. As the user is using the device, more entropy will be generated and the Synchronizer client will become unblocked. This might cause a delay in VMs appearing on the client, although a progress bar is visible indicating that work is being done. This can be avoided by using `/dev/urandom` as indicated above, although this is a poorer-quality random number source. Since the keys are long-lived, it is recommended that pseudorandomness is not used for the best security.

Chapter 4. OpenXT Video Architecture

This chapter describes how OpenXT handles the graphics displays for a device, both physical and emulated.

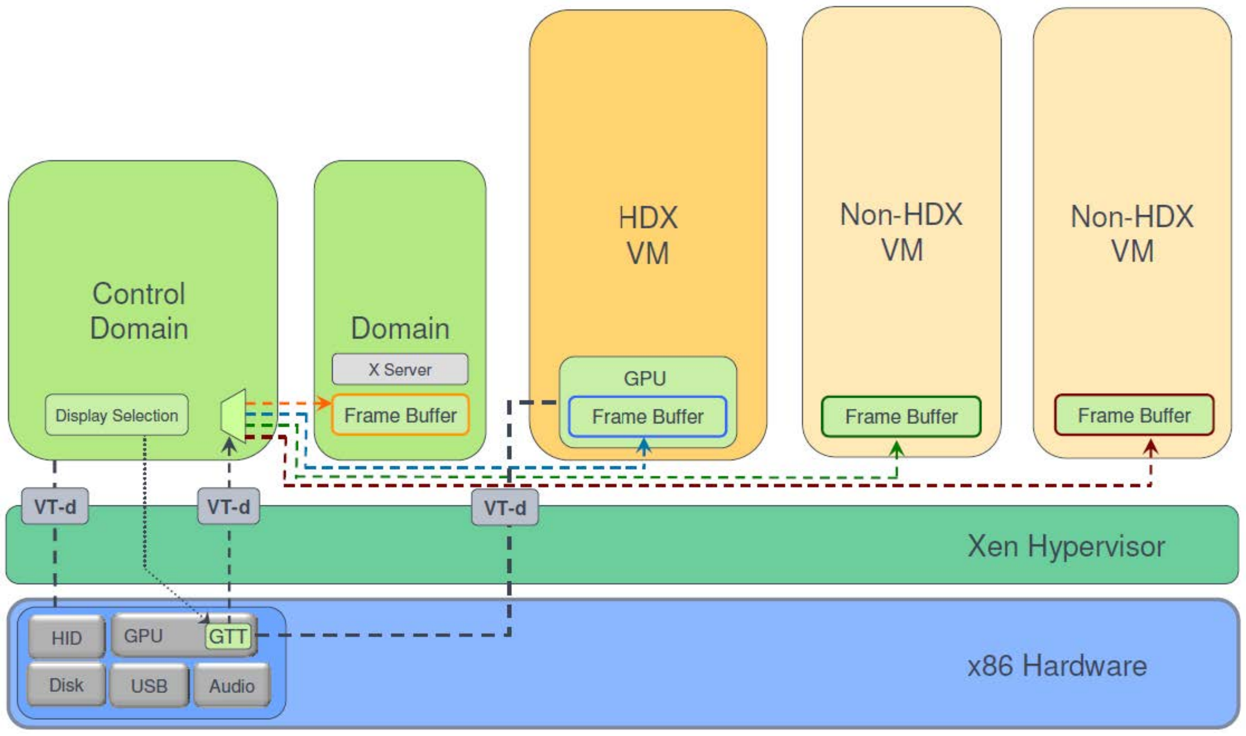
After a booting, or upon resuming from S3, OpenXT does the following:

- The graphics card is initialized
- The system probes and configures the laptop panel and external analog/digital monitors
- At this point, VMs (with native drivers) might or might not be running

If VMs are running, the physical and emulated GPUs interact as follows:

- If an HDX VM is running:
  - On display panels where the HDX VM's graphics are presented on the display, the HDX VM's native graphics driver writes directly to the physical GPU
  - On display panels where the HDX VM's graphics are not presented on the display, the HDX VM's native graphics driver writes to an emulated GPU. OpenXT coordinates state with the physical GPU.
  - On display panels where a non-HDX VM's graphics are presented on the display, OpenXT uses GTT to map the display of the emulated GPU frame buffer to the physical GPU
  - On display panels where a non-HDX VM's graphics are not presented on the display, the non-HDX VM's graphics driver writes to an emulated GPU with an off-screen frame buffer
- If no HDX VM is running, OpenXT optimizes GPU power usage based on non-HDX graphics workload

Figure 4.1. OpenXT Video Architecture:

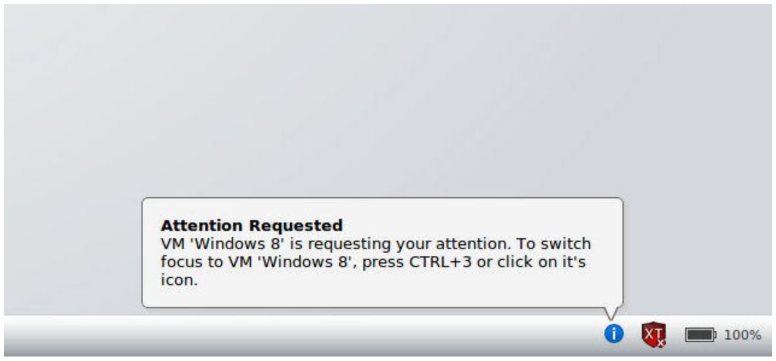


Chapter 5. OpenXT Disk Security

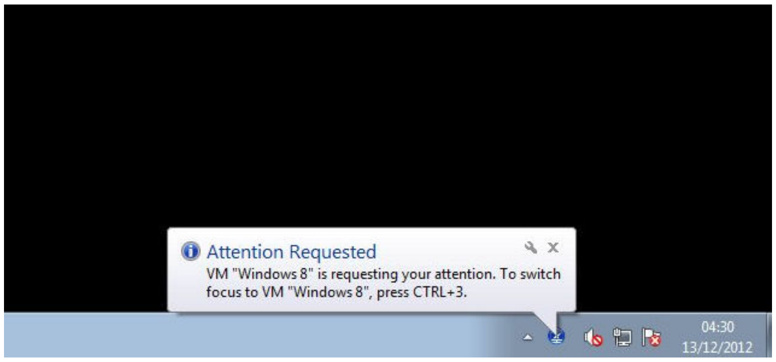
If you move a physical hard drive from one OpenXT system to another, you need to "bless" the disk. This is a security feature to keep somebody from stealing your hard drive and accessing your data. Hold down the **Shift** key during boot and select **OpenXT Technical Support option: console access**. You will be prompted to enter the administrative passphrase, which is the password you created when you installed OpenXT (the Control Domain password). You will then be asked, "Would you like to reseal the device with the current configuration?" Respond with a yes, then you can reboot the device and use your VMs.

Chapter 6. The request_attention method in the xenmgr interface

The xenmgr interface includes a method, **request_attention**. Executing this method from any guest VM (user VM or service VM) will cause a dialog box in both the UIVM and Windows guests to alert the user that the VM which sent it requires attention.



Attention Requested
VM 'Windows 8' is requesting your attention. To switch focus to VM 'Windows 8', press CTRL+3 or click on it's icon.



Attention Requested
VM "Windows 8" is requesting your attention. To switch focus to VM "Windows 8", press CTRL+3.

This is useful for the case where a service VM needs credentials from the user or has an issue which requires user intervention to resolve. Note that the dialog box means a VM wants you to switch to the VM; but it does NOT mean that we recommend that you take any action, including but not limited to giving your attention to any VM that is requesting said attention. A malicious VM can repeatedly call this function and cause this dialog box to appear repeatedly; there is no time delay that limits how often any VM can request attention.