Users' Manual

Xen v3.3

DISCLAIMER: This documentation is always under active development and as such there may be mistakes and omissions — watch out for these and please report any you find to the developers' mailing list, xen-devel@lists.xensource.com. The latest version is always available on-line. Contributions of material, suggestions and corrections are welcome.

Contents

Chapter 1

Introduction

Xen is an open-source *para-virtualizing* virtual machine monitor (VMM), or "hypervisor", for a variety of processor architectures including x86. Xen can securely execute multiple virtual machines on a single physical system with near native performance. Xen facilitates enterprise-grade functionality, including:

- Virtual machines with performance close to native hardware.
- Live migration of running virtual machines between physical hosts.
- Up to 32[1] virtual CPUs per guest virtual machine, with VCPU hotplug.
- x86/32 with PAE, x86/64, and IA64 platform support.
- Intel and AMD Virtualization Technology for unmodified guest operating systems (including Microsoft Windows).
- Excellent hardware support (supports almost all Linux device drivers).

1.1 Usage Scenarios

Usage scenarios for Xen include:

Server Consolidation. Move multiple servers onto a single physical host with performance and fault isolation provided at the virtual machine boundaries.

Hardware Independence. Allow legacy applications and operating systems to exploit new hardware.

Multiple OS configurations. Run multiple operating systems simultaneously, for development or testing purposes.

Kernel Development. Test and debug kernel modifications in a sand-boxed virtual machine — no need for a separate test machine.

[1] IA64 supports up to 64 virtual CPUs per guest virtual machine

Cluster Computing. Management at VM granularity provides more flexibility than separately managing each physical host, but better control and isolation than single-system image solutions, particularly by using live migration for load balancing.

Hardware support for custom OSes. Allow development of new OSes while benefiting from the wide-ranging hardware support of existing OSes such as Linux.

1.2 Operating System Support

Para-virtualization permits very high performance virtualization, even on architectures like x86 that are traditionally very hard to virtualize.

This approach requires operating systems to be *ported* to run on Xen. Porting an OS to run on Xen is similar to supporting a new hardware platform, however the process is simplified because the para-virtual machine architecture is very similar to the underlying native hardware. Even though operating system kernels must explicitly support Xen, a key feature is that user space applications and libraries *do not* require modification.

With hardware CPU virtualization as provided by Intel VT and AMD SVM technology, the ability to run an unmodified guest OS kernel is available. No porting of the OS is required, although some additional driver support is necessary within Xen itself. Unlike traditional full virtualization hypervisors, which suffer a tremendous performance overhead, the combination of Xen and VT or Xen and Pacifica technology complement one another to offer superb performance for para-virtualized guest operating systems and full support for unmodified guests running natively on the processor.

Paravirtualized Xen support is available for increasingly many operating systems: currently, mature Linux support is available and included in the standard distribution. Other OS ports, including NetBSD, FreeBSD and Solaris are also complete.

1.3 Hardware Support

Xen currently runs on the IA64 and x86 architectures. Multiprocessor machines are supported, and there is support for HyperThreading (SMT).

The default 32-bit Xen requires processor support for Physical Addressing Extensions (PAE), which enables the hypervisor to address up to 16GB of physical memory. Xen also supports x86/64 platforms such as Intel EM64T and AMD Opteron which can currently address up to 1TB of physical memory.

Xen offloads most of the hardware support issues to the guest OS running in the *Domain 0* management virtual machine. Xen itself contains only the code required to detect and start secondary processors, set up interrupt routing, and perform PCI bus enumeration. Device drivers run within a privileged guest OS rather than within Xen itself. This approach provides compatibility with the majority of device hardware supported by Linux. The default XenLinux build contains support for most server-class network and disk hardware, but you can add support for other hardware by configuring your XenLinux kernel in the normal way.

1.4 Structure of a Xen-Based System

A Xen system has multiple layers, the lowest and most privileged of which is Xen itself.

Xen may host multiple *guest* operating systems, each of which is executed within a secure virtual machine. In Xen terminology, a *domain*. Domains are scheduled by Xen to make effective use of the available physical CPUs. Each guest OS manages its own applications. This management includes the responsibility of scheduling each application within the time allotted to the VM by Xen.

The first domain, *domain 0*, is created automatically when the system boots and has special management privileges. Domain 0 builds other domains and manages their virtual devices. It also performs administrative tasks such as suspending, resuming and migrating other virtual machines.

Within domain 0, a process called *xend* runs to manage the system. Xend is responsible for managing virtual machines and providing access to their consoles. Commands are issued to xend over an HTTP interface, via a command-line tool.

1.5 History

Xen was originally developed by the Systems Research Group at the University of Cambridge Computer Laboratory as part of the XenoServers project, funded by the UK-EPSRC.

XenoServers aim to provide a "public infrastructure for global distributed computing". Xen plays a key part in that, allowing one to efficiently partition a single machine to enable multiple independent clients to run their operating systems and applications in an environment. This environment provides protection, resource isolation and accounting. The project web page contains further information along with pointers to papers and technical reports: `http://www.cl.cam.ac.uk/xeno`

Xen has grown into a fully-fledged project in its own right, enabling us to investigate interesting research issues regarding the best techniques for virtualizing resources such as the CPU, memory, disk and network. Project contributors now include Citrix, Intel, IBM, HP, AMD, Novell, RedHat, Sun, Fujitsu, and Samsung.

Xen was first described in a paper presented at SOSP in 2003[2], and the first public release (1.0) was made that October. Since then, Xen has significantly matured and is now used in production scenarios on many sites.

1.6 What's New

Xen 3.3.0 offers:

- IO Emulation (stub domains) for HVM IO performance and scailability
- Replacement of Intel VT vmxassist by new 16b emulation code
- Improved VT-d device pass-through e.g. for graphics devices
- Enhanced C and P state power management
- Exploitation of multi-queue support on modern NICs
- Removal of domain lock for improved PV guest scalability
- 2MB page support for HVM and PV guests
- CPU Portability

Xen 3.3 delivers the capabilities needed by enterprise customers and gives computing industry leaders a solid, secure platform to build upon for their virtualization solutions. This latest release establishes Xen as the definitive open source solution for virtualization.

[2] `http://www.cl.cam.ac.uk/netos/papers/2003-xensosp.pdf`

Part I

Installation

Chapter 2

Basic Installation

The Xen distribution includes three main components: Xen itself, ports of Linux and NetBSD to run on Xen, and the userspace tools required to manage a Xen-based system. This chapter describes how to install the Xen 3.3 distribution from source. Alternatively, there may be pre-built packages available as part of your operating system distribution.

2.1 Prerequisites

The following is a full list of prerequisites. Items marked '†' are required by the xend control tools, and hence required if you want to run more than one virtual machine; items marked '∗' are only required if you wish to build from source.

- A working Linux distribution using the GRUB bootloader and running on a P6-class or newer CPU.

† The `iproute2` package.

† The Linux bridge-utils[1] (e.g., `/sbin/brctl`)

† The Linux hotplug system[2] (e.g., `/sbin/hotplug` and related scripts). On newer distributions, this is included alongside the Linux udev system[3].

∗ Build tools (gcc v3.2.x or v3.3.x, binutils, GNU make).

∗ Development installation of zlib (e.g., zlib-dev).

∗ Development installation of Python v2.2 or later (e.g., python-dev).

∗ LaTeX and transfig are required to build the documentation.

[1] Available from `http://bridge.sourceforge.net`

[2] Available from `http://linux-hotplug.sourceforge.net/`

[3] See `http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev.html/`

Once you have satisfied these prerequisites, you can now install either a binary or source distribution of Xen.

2.2 Installing from Binary Tarball

Pre-built tarballs are available for download from the XenSource downloads page:

    http://www.xensource.com/downloads/

Once you've downloaded the tarball, simply unpack and install:

```
# tar zxvf xen-3.0-install.tgz
# cd xen-3.0-install
# sh ./install.sh
```

Once you've installed the binaries you need to configure your system as described in Section 2.5.

2.3 Installing from RPMs

Pre-built RPMs are available for download from the XenSource downloads page:

    http://www.xensource.com/downloads/

Once you've downloaded the RPMs, you typically install them via the RPM commands:

```
# rpm -iv rpmname
```

See the instructions and the Release Notes for each RPM set referenced at:

    http://www.xensource.com/downloads/.

2.4 Installing from Source

This section describes how to obtain, build and install Xen from source.

2.4.1 Obtaining the Source

The Xen source tree is available as either a compressed source tarball or as a clone of our master Mercurial repository.

Obtaining the Source Tarball

Stable versions and daily snapshots of the Xen source tree are available from the Xen download page:

    http://www.xensource.com/downloads/

Obtaining the source via Mercurial

The source tree may also be obtained via the public Mercurial repository at:

```
http://xenbits.xensource.com
```

See the instructions and the Getting Started Guide referenced at:

```
http://www.xensource.com/downloads/
```

2.4.2 Building from Source

The top-level Xen Makefile includes a target "world" that will do the following:

- Build Xen.
- Build the control tools, including xend.
- Download (if necessary) and unpack the Linux 2.6 source code, and patch it for use with Xen.
- Build a Linux kernel to use in domain 0 and a smaller unprivileged kernel, which can be used for unprivileged virtual machines.

After the build has completed you should have a top-level directory called `dist/` in which all resulting targets will be placed. Of particular interest are the two XenLinux kernel images, one with a "-xen0" extension which contains hardware device drivers and drivers for Xen's virtual devices, and one with a "-xenU" extension that just contains the virtual ones. These are found in `dist/install/boot/` along with the image for Xen itself and the configuration files used during the build.

To customize the set of kernels built you need to edit the top-level Makefile. Look for the line:

```
KERNELS ?= linux-2.6-xen0 linux-2.6-xenU
```

You can edit this line to include any set of operating system kernels which have configurations in the top-level `buildconfigs/` directory.

2.4.3 Custom Kernels

If you wish to build a customized XenLinux kernel (e.g. to support additional devices or enable distribution-required features), you can use the standard Linux configuration mechanisms, specifying that the architecture being built for is `xen`, e.g:

```
# cd linux-2.6.12-xen0
# make ARCH=xen xconfig
# cd ..
# make
```

You can also copy an existing Linux configuration (`.config`) into e.g. `linux-2.6.12-xen0` and execute:

```
# make ARCH=xen oldconfig
```

You may be prompted with some Xen-specific options. We advise accepting the defaults for these options.

Note that the only difference between the two types of Linux kernels that are built is the configuration file used for each. The "U" suffixed (unprivileged) versions don't contain any of the physical hardware device drivers, leading to a 30% reduction in size; hence you may prefer these for your non-privileged domains. The "0" suffixed privileged versions can be used to boot the system, as well as in driver domains and unprivileged domains.

2.4.4 Installing Generated Binaries

The files produced by the build process are stored under the `dist/install/` directory. To install them in their default locations, do:

```
# make install
```

Alternatively, users with special installation requirements may wish to install them manually by copying the files to their appropriate destinations.

The `dist/install/boot` directory will also contain the config files used for building the XenLinux kernels, and also versions of Xen and XenLinux kernels that contain debug symbols such as (`xen-syms-3.0.0` and `vmlinux-syms-2.6.12.6-xen0`) which are essential for interpreting crash dumps. Retain these files as the developers may wish to see them if you post on the mailing list.

2.5 Configuration

Once you have built and installed the Xen distribution, it is simple to prepare the machine for booting and running Xen.

2.5.1 GRUB Configuration

An entry should be added to `grub.conf` (often found under `/boot/` or `/boot/grub/`) to allow Xen / XenLinux to boot. This file is sometimes called `menu.lst`, depending on your distribution. The entry should look something like the following:

```
title Xen 3.0 / XenLinux 2.6
  kernel /boot/xen-3.0.gz dom0_mem=262144
  module /boot/vmlinuz-2.6-xen0 root=/dev/sda4 ro console=tty0
```

The kernel line tells GRUB where to find Xen itself and what boot parameters should be passed to it (in this case, setting the domain 0 memory allocation in kilobytes and the settings for the serial port). For more details on the various Xen boot parameters see Section 11.3.

The module line of the configuration describes the location of the XenLinux kernel that Xen should start and the parameters that should be passed to it. These are standard Linux parameters, identifying the root device and specifying it be initially mounted read only and instructing that console output be sent to the screen. Some distributions such as SuSE do not require the `ro` parameter.

To use an initrd, add another `module` line to the configuration, like:

```
module /boot/my_initrd.gz
```

When installing a new kernel, it is recommended that you do not delete existing menu options from `menu.lst`, as you may wish to boot your old Linux kernel in future, particularly if you have problems.

2.5.2 Serial Console (optional)

Serial console access allows you to manage, monitor, and interact with your system over a serial console. This can allow access from another nearby system via a null-modem ("LapLink") cable or remotely via a serial concentrator.

You system's BIOS, bootloader (GRUB), Xen, Linux, and login access must each be individually configured for serial console access. It is *not* strictly necessary to have each component fully functional, but it can be quite useful.

For general information on serial console configuration under Linux, refer to the "Remote Serial Console HOWTO" at The Linux Documentation Project: `http://www.tldp.org`

Serial Console BIOS configuration

Enabling system serial console output neither enables nor disables serial capabilities in GRUB, Xen, or Linux, but may make remote management of your system more convenient by displaying POST and other boot messages over serial port and allowing remote BIOS configuration.

Refer to your hardware vendor's documentation for capabilities and procedures to enable BIOS serial redirection.

Serial Console GRUB configuration

Enabling GRUB serial console output neither enables nor disables Xen or Linux serial capabilities, but may made remote management of your system more convenient by displaying GRUB prompts, menus, and actions over serial port and allowing remote GRUB management.

Adding the following two lines to your GRUB configuration file, typically either `/boot/grub/menu.lst` or `/boot/grub/grub.conf` depending on your distro, will enable GRUB serial output.

```
serial --unit=0 --speed=115200 --word=8 --parity=no --stop=1
terminal --timeout=10 serial console
```

Note that when both the serial port and the local monitor and keyboard are enabled, the text "*Press any key to continue*" will appear at both. Pressing a key on one device will cause GRUB to display to that device. The other device will see no output. If no key is pressed before the timeout period expires, the system will boot to the default GRUB boot entry.

Please refer to the GRUB documentation for further information.

Serial Console Xen configuration

Enabling Xen serial console output neither enables nor disables Linux kernel output or logging in to Linux over serial port. It does however allow you to monitor and log the Xen boot process via serial console and can be very useful in debugging.

In order to configure Xen serial console output, it is necessary to add a boot option to your GRUB config; e.g. replace the previous example kernel line with:

```
kernel /boot/xen.gz dom0_mem=131072 com1=115200,8n1 console=com1,vga
```

This configures Xen to output on COM1 at 115,200 baud, 8 data bits, no parity and 1 stop bit. Modify these parameters for your environment. See Section 11.3 for an explanation of all boot parameters.

One can also configure XenLinux to share the serial console; to achieve this append "`console=ttyS0`" to your module line.

Serial Console Linux configuration

Enabling Linux serial console output at boot neither enables nor disables logging in to Linux over serial port. It does however allow you to monitor and log the Linux boot process via serial console and can be very useful in debugging.

To enable Linux output at boot time, add the parameter `console=ttyS0` (or ttyS1, ttyS2, etc.) to your kernel GRUB line. Under Xen, this might be:

```
module /vmlinuz-2.6-xen0 ro root=/dev/VolGroup00/LogVol00 \
console=ttyS0, 115200
```

to enable output over ttyS0 at 115200 baud.

Serial Console Login configuration

Logging in to Linux via serial console, under Xen or otherwise, requires specifying a login prompt be started on the serial port. To permit root logins over serial console, the serial port must be added to `/etc/securetty`.

To automatically start a login prompt over the serial port, add the line:

```
c:2345:respawn:/sbin/mingetty ttyS0
```

to `/etc/inittab`. Run `init q` to force a reload of your inttab and start getty.

To enable root logins, add `ttyS0` to `/etc/securetty` if not already present.

Your distribution may use an alternate getty; options include getty, mgetty and agetty. Consult your distribution's documentation for further information.

2.5.3 TLS Libraries

Users of the XenLinux 2.6 kernel should disable Thread Local Storage (TLS) (e.g. by doing a `mv /lib/tls /lib/tls.disabled`) before attempting to boot a XenLinux kernel[4]. You can always reenable TLS by restoring the directory to its original location (i.e. `mv /lib/tls.disabled /lib/tls`).

The reason for this is that the current TLS implementation uses segmentation in a way that is not permissible under Xen. If TLS is not disabled, an emulation mode is used within Xen which reduces performance substantially. To ensure full performance you should install a 'Xen-friendly' (nosegneg) version of the library.

2.6 Booting Xen

It should now be possible to restart the system and use Xen. Reboot and choose the new Xen option when the Grub screen appears.

What follows should look much like a conventional Linux boot. The first portion of the output comes from Xen itself, supplying low level information about itself and the underlying hardware. The last portion of the output comes from XenLinux.

You may see some error messages during the XenLinux boot. These are not necessarily anything to worry about—they may result from kernel configuration differences between your XenLinux kernel and the one you usually use.

When the boot completes, you should be able to log into your system as usual. If you are unable to log in, you should still be able to reboot with your normal Linux kernel by selecting it at the GRUB prompt.

[4] If you boot without first disabling TLS, you will get a warning message during the boot process. In this case, simply perform the rename after the machine is up and then run `/sbin/ldconfig` to make it take effect.

Chapter 3

Booting a Xen System

Booting the system into Xen will bring you up into the privileged management domain, Domain0. At that point you are ready to create guest domains and "boot" them using the `xm create` command.

3.1 Booting Domain0

After installation and configuration is complete, reboot the system and and choose the new Xen option when the Grub screen appears.

What follows should look much like a conventional Linux boot. The first portion of the output comes from Xen itself, supplying low level information about itself and the underlying hardware. The last portion of the output comes from XenLinux.

When the boot completes, you should be able to log into your system as usual. If you are unable to log in, you should still be able to reboot with your normal Linux kernel by selecting it at the GRUB prompt.

The first step in creating a new domain is to prepare a root filesystem for it to boot. Typically, this might be stored in a normal partition, an LVM or other volume manager partition, a disk file or on an NFS server. A simple way to do this is simply to boot from your standard OS install CD and install the distribution into another partition on your hard drive.

To start the xend control daemon, type

```
# xend start
```

If you wish the daemon to start automatically, see the instructions in Section 4.1. Once the daemon is running, you can use the `xm` tool to monitor and maintain the domains running on your system. This chapter provides only a brief tutorial. We provide full details of the `xm` tool in the next chapter.

3.2 Booting Guest Domains

3.2.1 Creating a Domain Configuration File

Before you can start an additional domain, you must create a configuration file. We provide two example files which you can use as a starting point:

- `/etc/xen/xmexample1` is a simple template configuration file for describing a single VM.
- `/etc/xen/xmexample2` file is a template description that is intended to be reused for multiple virtual machines. Setting the value of the `vmid` variable on the `xm` command line fills in parts of this template.

There are also a number of other examples which you may find useful. Copy one of these files and edit it as appropriate. Typical values you may wish to edit include:

kernel Set this to the path of the kernel you compiled for use with Xen (e.g. `kernel = ''/boot/vmlinuz-2.6-xenU''`)

memory Set this to the size of the domain's memory in megabytes (e.g. `memory = 64`)

disk Set the first entry in this list to calculate the offset of the domain's root partition, based on the domain ID. Set the second to the location of `/usr` if you are sharing it between domains (e.g. `disk = ['phy:your_hard_drive%d,sda1,w' % (base_partition_number + vmid), 'phy:your_usr_partition,sda6,r' ]`

dhcp Uncomment the dhcp variable, so that the domain will receive its IP address from a DHCP server (e.g. `dhcp=''dhcp''`)

You may also want to edit the **vif** variable in order to choose the MAC address of the virtual ethernet interface yourself. For example:

```
vif = ['mac=00:16:3E:F6:BB:B3']
```

If you do not set this variable, xend will automatically generate a random MAC address from the range 00:16:3E:xx:xx:xx, assigned by IEEE to XenSource as an OUI (organizationally unique identifier). XenSource Inc. gives permission for anyone to use addresses randomly allocated from this range for use by their Xen domains.

For a list of IEEE OUI assignments, see `http://standards.ieee.org/regauth/oui/oui.txt`

3.2.2 Booting the Guest Domain

The `xm` tool provides a variety of commands for managing domains. Use the `create` command to start new domains. Assuming you've created a configuration file `myvmconf` based around /etc/xen/xmexample2, to start a domain with virtual machine ID 1 you should type:

```
# xm create -c myvmconf vmid=1
```

The -c switch causes xm to turn into the domain's console after creation. The vmid=1 sets the vmid variable used in the myvmconf file.

You should see the console boot messages from the new domain appearing in the terminal in which you typed the command, culminating in a login prompt.

3.3 Starting / Stopping Domains Automatically

It is possible to have certain domains start automatically at boot time and to have dom0 wait for all running domains to shutdown before it shuts down the system.

To specify a domain is to start at boot-time, place its configuration file (or a link to it) under /etc/xen/auto/.

A Sys-V style init script for Red Hat and LSB-compliant systems is provided and will be automatically copied to /etc/init.d/ during install. You can then enable it in the appropriate way for your distribution.

For instance, on Red Hat:

```
# chkconfig --add xendomains
```

By default, this will start the boot-time domains in runlevels 3, 4 and 5.

You can also use the service command to run this script manually, e.g:

```
# service xendomains start
```

Starts all the domains with config files under /etc/xen/auto/.

```
# service xendomains stop
```

Shuts down all running Xen domains.

Part II

Configuration and Management

Chapter 4

Domain Management Tools

This chapter summarizes the management software and tools available.

4.1 Xend

The Xend node control daemon performs system management functions related to virtual machines. It forms a central point of control of virtualized resources, and must be running in order to start and manage virtual machines. Xend must be run as root because it needs access to privileged system management functions.

An initialization script named `/etc/init.d/xend` is provided to start Xend at boot time. Use the tool appropriate (i.e. chkconfig) for your Linux distribution to specify the runlevels at which this script should be executed, or manually create symbolic links in the correct runlevel directories.

Xend can be started on the command line as well, and supports the following set of parameters:

`# xend start`	start xend, if not already running
`# xend stop`	stop xend if already running
`# xend restart`	restart xend if running, otherwise start it
`# xend status`	indicates xend status by its return code

A SysV init script called `xend` is provided to start xend at boot time. `make install` installs this script in `/etc/init.d`. To enable it, you have to make symbolic links in the appropriate runlevel directories or use the `chkconfig` tool, where available. Once xend is running, administration can be done using the `xm` tool.

4.1.1 Logging

As xend runs, events will be logged to `/var/log/xen/xend.log` and (less frequently) to `/var/log/xen/xend-debug.log`. These, along with the standard syslog files, are useful when troubleshooting problems.

4.1.2 Configuring Xend

Xend is written in Python. At startup, it reads its configuration information from the file `/etc/xen/xend-config.sxp`. The Xen installation places an example `xend-config.sxp` file in the `/etc/xen` subdirectory which should work for most installations.

See the example configuration file `xend-debug.sxp` and the section 5 man page `xend-config.sxp` for a full list of parameters and more detailed information. Some of the most important parameters are discussed below.

An HTTP interface and a Unix domain socket API are available to communicate with Xend. This allows remote users to pass commands to the daemon. By default, Xend does not start an HTTP server. It does start a Unix domain socket management server, as the low level utility `xm` requires it. For support of cross-machine migration, Xend can start a relocation server. This support is not enabled by default for security reasons.

Note: the example `xend` configuration file modifies the defaults and starts up Xend as an HTTP server as well as a relocation server.

From the file:

```
#(xend-http-server no)
(xend-http-server yes)
#(xend-unix-server yes)
#(xend-relocation-server no)
(xend-relocation-server yes)
```

Comment or uncomment lines in that file to disable or enable features that you require.

Connections from remote hosts are disabled by default:

```
# Address xend should listen on for HTTP connections, if xend-http-server
# set.
# Specifying 'localhost' prevents remote connections.
# Specifying the empty string '' (the default) allows all connections.
#(xend-address '')
(xend-address localhost)
```

It is recommended that if migration support is not needed, the `xend-relocation-server` parameter value be changed to "`no`" or commented out.

4.2 Xm

The xm tool is the primary tool for managing Xen from the console. The general format of an xm command line is:

```
# xm command [switches] [arguments] [variables]
```

The available *switches* and *arguments* are dependent on the *command* chosen. The *variables* may be set using declarations of the form `variable=value` and command line declarations override any of the values in the configuration file being used, including the standard variables described above and any custom variables (for instance, the `xmdefconfig` file uses a `vmid` variable).

For online help for the commands available, type:

```
# xm help
```

This will list the most commonly used commands. The full list can be obtained using `xm help --long`. You can also type `xm help <command>` for more information on a given command.

4.2.1 Basic Management Commands

One useful command is `# xm list` which lists all domains running in rows of the following format:

```
name domid memory vcpus state cputime
```

The meaning of each field is as follows:

name The descriptive name of the virtual machine.

domid The number of the domain ID this virtual machine is running in.

memory Memory size in megabytes.

vcpus The number of virtual CPUs this domain has.

state Domain state consists of 5 fields:

- **r** running
- **b** blocked
- **p** paused
- **s** shutdown
- **c** crashed

cputime How much CPU time (in seconds) the domain has used so far.

The `xm list` command also supports a long output format when the `-l` switch is used. This outputs the full details of the running domains in xend's SXP configuration format.

If you want to know how long your domains have been running for, then you can use the `# xm uptime` command.

You can get access to the console of a particular domain using the `# xm console` command (e.g. `# xm console myVM`).

4.2.2 Domain Scheduling Management Commands

The credit CPU scheduler automatically load balances guest VCPUs across all available physical CPUs on an SMP host. The user need not manually pin VCPUs to load balance the system. However, she can restrict which CPUs a particular VCPU may run on using the `xm vcpu-pin` command.

Each guest domain is assigned a `weight` and a `cap`.

A domain with a weight of 512 will get twice as much CPU as a domain with a weight of 256 on a contended host. Legal weights range from 1 to 65535 and the default is 256.

The cap optionally fixes the maximum amount of CPU a guest will be able to consume, even if the host system has idle CPU cycles. The cap is expressed in percentage of one physical CPU: 100 is 1 physical CPU, 50 is half a CPU, 400 is 4 CPUs, etc... The default, 0, means there is no upper cap.

When you are running with the credit scheduler, you can check and modify your domains' weights and caps using the `xm sched-credit` command:

`xm sched-credit -d <domain>`	lists weight and cap
`xm sched-credit -d <domain> -w <weight>`	sets the weight
`xm sched-credit -d <domain> -c <cap>`	sets the cap

Chapter 5

Domain Configuration

The following contains the syntax of the domain configuration files and description of how to further specify networking, driver domain and general scheduling behavior.

5.1 Configuration Files

Xen configuration files contain the following standard variables. Unless otherwise stated, configuration items should be enclosed in quotes: see the configuration scripts in `/etc/xen/` for concrete examples.

kernel Path to the kernel image.

ramdisk Path to a ramdisk image (optional).

memory Memory size in megabytes.

vcpus The number of virtual CPUs.

console Port to export the domain console on (default 9600 + domain ID).

vif Network interface configuration. This may simply contain an empty string for each desired interface, or may override various settings, e.g.

```
vif = [ 'mac=00:16:3E:00:00:11, bridge=xen-br0',
        'bridge=xen-br1' ]
```

to assign a MAC address and bridge to the first interface and assign a different bridge to the second interface, leaving xend to choose the MAC address. The settings that may be overridden in this way are type, mac, bridge, ip, script, backend, and vifname.

disk List of block devices to export to the domain e.g. `disk = [ 'phy:hda1,sda1,r' ]` exports physical device `/dev/hda1` to the domain as `/dev/sda1` with read-only access. Exporting a disk read-write which is currently mounted is dangerous – if you are *certain* you wish to do this, you can specify `w!` as the mode.

dhcp Set to `'dhcp'` if you want to use DHCP to configure networking.

netmask Manually configured IP netmask.

gateway Manually configured IP gateway.

hostname Set the hostname for the virtual machine.

root Specify the root device parameter on the kernel command line.

nfs_server IP address for the NFS server (if any).

nfs_root Path of the root filesystem on the NFS server (if any).

extra Extra string to append to the kernel command line (if any)

Additional fields are documented in the example configuration files (e.g. to configure virtual TPM functionality).

For additional flexibility, it is also possible to include Python scripting commands in configuration files. An example of this is the `xmexample2` file, which uses Python code to handle the `vmid` variable.

5.2 Network Configuration

For many users, the default installation should work "out of the box". More complicated network setups, for instance with multiple Ethernet interfaces and/or existing bridging setups will require some special configuration.

The purpose of this section is to describe the mechanisms provided by xend to allow a flexible configuration for Xen's virtual networking.

5.2.1 Xen virtual network topology

Each domain network interface is connected to a virtual network interface in dom0 by a point to point link (effectively a "virtual crossover cable"). These devices are named `vif<domid>.<vifid>` (e.g. `vif1.0` for the first interface in domain 1, `vif3.1` for the second interface in domain 3).

Traffic on these virtual interfaces is handled in domain 0 using standard Linux mechanisms for bridging, routing, rate limiting, etc. Xend calls on two shell scripts to perform initial configuration of the network and configuration of new virtual interfaces. By default, these scripts configure a single bridge for all the virtual interfaces. Arbitrary routing / bridging configurations can be configured by customizing the scripts, as described in the following section.

5.2.2 Xen networking scripts

Xen's virtual networking is configured by two shell scripts (by default `network-bridge` and `vif-bridge`). These are called automatically by xend when certain events occur, with arguments to the scripts providing further contextual information. These scripts are found by default in `/etc/xen/scripts`. The names and locations of the scripts can be configured in `/etc/xen/xend-config.sxp`.

network-bridge: This script is called whenever xend is started or stopped to respectively initialize or tear down the Xen virtual network. In the default configuration initialization creates the bridge 'xen-br0' and moves eth0 onto that bridge, modifying the routing accordingly. When xend exits, it deletes the Xen bridge and removes eth0, restoring the normal IP and routing configuration.

vif-bridge: This script is called for every domain virtual interface and can configure firewalling rules and add the vif to the appropriate bridge. By default, this adds and removes VIFs on the default Xen bridge.

Other example scripts are available (`network-route` and `vif-route`, `network-nat` and `vif-nat`). For more complex network setups (e.g. where routing is required or integrate with existing bridges) these scripts may be replaced with customized variants for your site's preferred configuration.

5.3 Driver Domain Configuration

5.3.1 PCI

Individual PCI devices can be assigned to a given domain (a PCI driver domain) to allow that domain direct access to the PCI hardware.

While PCI Driver Domains can increase the stability and security of a system by addressing a number of security concerns, there are some security issues that remain that you can read about in Section 9.2.

Compile-Time Setup

To use this functionality, ensure that the PCI Backend is compiled in to a privileged domain (e.g. domain 0) and that the domains which will be assigned PCI devices have the PCI Frontend compiled in. In XenLinux, the PCI Backend is available under the Xen configuration section while the PCI Frontend is under the architecture-specific "Bus Options" section. You may compile both the backend and the frontend into the same kernel; they will not affect each other.

PCI Backend Configuration - Binding at Boot

The PCI devices you wish to assign to unprivileged domains must be ”hidden” from your backend domain (usually domain 0) so that it does not load a driver for them. Use the `pciback.hide` kernel parameter which is specified on the kernel command-line and is configurable through GRUB (see Section 2.5). Note that devices are not really hidden from the backend domain. The PCI Backend appears to the Linux kernel as a regular PCI device driver. The PCI Backend ensures that no other device driver loads for the devices by binding itself as the device driver for those devices. PCI devices are identified by hexadecimal slot/function numbers (on Linux, use `lspci` to determine slot/function numbers of your devices) and can be specified with or without the PCI domain:

(*bus*:*slot*.*func*) example (02:1d.3)

(*domain*:*bus*:*slot*.*func*) example (0000:02:1d.3)

An example kernel command-line which hides two PCI devices might be:

```
root=/dev/sda4 ro console=tty0 pciback.hide=(02:01.f)(0000:04:1d.0)
```

PCI Backend Configuration - Late Binding

PCI devices can also be bound to the PCI Backend after boot through the manual binding/unbinding facilities provided by the Linux kernel in sysfs (allowing for a Xen user to give PCI devices to driver domains that were not specified on the kernel command-line). There are several attributes with the PCI Backend's sysfs directory (`/sys/bus/pci/drivers/pciback`) that can be used to bind/unbind devices:

slots lists all of the PCI slots that the PCI Backend will try to seize (or ”hide” from Domain 0). A PCI slot must appear in this list before it can be bound to the PCI Backend through the `bind` attribute.

new_slot write the name of a slot here (in 0000:00:00.0 format) to have the PCI Backend seize the device in this slot.

remove_slot write the name of a slot here (same format as `new_slot`) to have the PCI Backend no longer try to seize devices in this slot. Note that this does not unbind the driver from a device it has already seized.

bind write the name of a slot here (in 0000:00:00.0 format) to have the Linux kernel attempt to bind the device in that slot to the PCI Backend driver.

unbind write the name of a skit here (same format as `bind`) to have the Linux kernel unbind the device from the PCI Backend. DO NOT unbind a device while it is currently given to a PCI driver domain!

Some examples:

Bind a device to the PCI Backend which is not bound to any other driver.

```
# # Add a new slot to the PCI Backend's list
# echo -n 0000:01:04.d > /sys/bus/pci/drivers/pciback/new_slot
# # Now that the backend is watching for the slot, bind to it
# echo -n 0000:01:04.d > /sys/bus/pci/drivers/pciback/bind
```

Unbind a device from its driver and bind to the PCI Backend.

```
# # Unbind a PCI network card from its network driver
# echo -n 0000:05:02.0 > /sys/bus/pci/drivers/3c905/unbind
# # And now bind it to the PCI Backend
# echo -n 0000:05:02.0 > /sys/bus/pci/drivers/pciback/new_slot
# echo -n 0000:05:02.0 > /sys/bus/pci/drivers/pciback/bind
```

Note that the "-n" option in the example is important as it causes echo to not output a new-line.

PCI Backend Configuration - User-space Quirks

Quirky devices (such as the Broadcom Tigon 3) may need write access to their configuration space registers. Xen can be instructed to allow specified PCI devices write access to specific configuration space registers. The policy may be found in:

`/etc/xen/xend-pci-quirks.sxp`

The policy file is heavily commented and is intended to provide enough documentation for developers to extend it.

PCI Backend Configuration - Permissive Flag

If the user-space quirks approach doesn't meet your needs you may want to enable the permissive flag for that device. To do so, first get the PCI domain, bus, slot, and function information from dom0 via `lspci`. Then augment the user-space policy for permissive devices. The permissive policy can be found in:

`/etc/xen/xend-pci-permissive.sxp`

Currently, the only way to reset the permissive flag is to unbind the device from the PCI Backend driver.

PCI Backend - Checking Status

There two important sysfs nodes that provide a mechanism to view specifics on quirks and permissive devices:

`/sys/bus/drivers/pciback/permissive`
: Use `cat` on this file to view a list of permissive slots.

/sys/bus/drivers/pciback/quirks
: Use `cat` on this file view a hierarchical view of devices bound to the PCI backend, their PCI vendor/device ID, and any quirks that are associated with that particular slot.

You may notice that every device bound to the PCI backend has 17 quirks standard ”quirks” regardless of `xend-pci-quirks.sxp`. These default entries are necessary to support interactions between the PCI bus manager and the device bound to it. Even non-quirky devices should have these standard entries.

In this case, preference was given to accuracy over aesthetics by choosing to show the standard quirks in the quirks list rather than hide them from the inquiring user

PCI Frontend Configuration

To configure a domU to receive a PCI device:

Command-line: Use the *pci* command-line flag. For multiple devices, use the option multiple times.

```
xm create netcard-dd pci=01:00.0 pci=02:03.0
```

Flat Format configuration file: Specify all of your PCI devices in a python list named *pci*.

```
pci=['01:00.0','02:03.0']
```

SXP Format configuration file: Use a single PCI device section for all of your devices (specify the numbers in hexadecimal with the preceding '0x'). Note that *domain* here refers to the PCI domain, not a virtual machine within Xen.

```
(device (pci
    (dev (domain 0x0)(bus 0x3)(slot 0x1a)(func 0x1)
    (dev (domain 0x0)(bus 0x1)(slot 0x5)(func 0x0)
)
```

5.4 Support for virtual Trusted Platform Module (vTPM)

Paravirtualized domains can be given access to a virtualized version of a TPM. This enables applications in these domains to use the services of the TPM device for example through a TSS stack [1]. The Xen source repository provides the necessary software components to enable virtual TPM access. Support is provided through several different pieces. First, a TPM emulator has been modified to provide TPM's functionality for the virtual TPM subsystem. Second, a virtual TPM Manager coordinates the virtual TPMs efforts, manages their creation, and provides protected key storage using

[1] Trousers TSS stack: http://sourceforge.net/projects/trousers

the TPM. Third, a device driver pair providing a TPM front- and backend is available for XenLinux to deliver TPM commands from the domain to the virtual TPM manager, which dispatches it to a software TPM. Since the TPM Manager relies on a HW TPM for protected key storage, therefore this subsystem requires a Linux-supported hardware TPM. For development purposes, a TPM emulator is available for use on non-TPM enabled platforms.

Compile-Time Setup

To enable access to the virtual TPM, the virtual TPM backend driver must be compiled for a privileged domain (e.g. domain 0). Using the XenLinux configuration, the necessary driver can be selected in the Xen configuration section. Unless the driver has been compiled into the kernel, its module must be activated using the following command:

```
modprobe tpmbk
```

Similarly, the TPM frontend driver must be compiled for the kernel trying to use TPM functionality. Its driver can be selected in the kernel configuration section Device Driver / Character Devices / TPM Devices. Along with that the TPM driver for the built-in TPM must be selected. If the virtual TPM driver has been compiled as module, it must be activated using the following command:

```
modprobe tpm_xenu
```

Furthermore, it is necessary to build the virtual TPM manager and software TPM by making changes to entries in Xen build configuration files. The following entry in the file Config.mk in the Xen root source directory must be made:

```
VTPM_TOOLS ?= y
```

After a build of the Xen tree and a reboot of the machine, the TPM backend drive must be loaded. Once loaded, the virtual TPM manager daemon must be started before TPM-enabled guest domains may be launched. To enable being the destination of a virtual TPM Migration, the virtual TPM migration daemon must also be loaded.

```
vtpm_managerd
```

```
vtpm_migratord
```

Once the VTPM manager is running, the VTPM can be accessed by loading the front end driver in a guest domain.

Development and Testing TPM Emulator

For development and testing on non-TPM enabled platforms, a TPM emulator can be used in replacement of a platform TPM. First, the entry in the file tools/vtpm/Rules.mk must look as follows:

```
BUILD_EMULATOR = y
```

Second, the entry in the file tool/vtpm_manager/Rules.mk must be uncommented as follows:

```
# TCS talks to fifo's rather than /dev/tpm. TPM Emulator assumed on fifos
CFLAGS += -DDUMMY_TPM
```

Before starting the virtual TPM Manager, start the emulator by executing the following in dom0:

```
tpm_emulator clear
```

vTPM Frontend Configuration

To provide TPM functionality to a user domain, a line must be added to the virtual TPM configuration file using the following format:

```
vtpm = ['instance=<instance number>, backend=<domain id>']
```

The *instance number* reflects the preferred virtual TPM instance to associate with the domain. If the selected instance is already associated with another domain, the system will automatically select the next available instance. An instance number greater than zero must be provided. It is possible to omit the instance parameter from the configuration file.

The *domain id* provides the ID of the domain where the virtual TPM backend driver and virtual TPM are running in. It should currently always be set to '0'.

Examples for valid vtpm entries in the configuration file are

```
vtpm = ['instance=1, backend=0']
```

and

```
vtpm = ['backend=0'].
```

Using the virtual TPM

Access to TPM functionality is provided by the virtual TPM frontend driver. Similar to existing hardware TPM drivers, this driver provides basic TPM status information through the *sysfs* filesystem. In a Xen user domain the sysfs entries can be found in /sys/devices/xen/vtpm-0.

Commands can be sent to the virtual TPM instance using the character device /dev/tpm0 (major 10, minor 224).

Chapter 6

Storage and File System Management

Storage can be made available to virtual machines in a number of different ways. This chapter covers some possible configurations.

The most straightforward method is to export a physical block device (a hard drive or partition) from dom0 directly to the guest domain as a virtual block device (VBD).

Storage may also be exported from a filesystem image or a partitioned filesystem image as a *file-backed VBD*.

Finally, standard network storage protocols such as NBD, iSCSI, NFS, etc., can be used to provide storage to virtual machines.

6.1 Exporting Physical Devices as VBDs

One of the simplest configurations is to directly export individual partitions from domain 0 to other domains. To achieve this use the `phy:` specifier in your domain configuration file. For example a line like

```
disk = ['phy:hda3,sda1,w']
```

specifies that the partition `/dev/hda3` in domain 0 should be exported read-write to the new domain as `/dev/sda1`; one could equally well export it as `/dev/hda` or `/dev/sdb5` should one wish.

In addition to local disks and partitions, it is possible to export any device that Linux considers to be "a disk" in the same manner. For example, if you have iSCSI disks or GNBD volumes imported into domain 0 you can export these to other domains using the `phy:` disk syntax. E.g.:

```
disk = ['phy:vg/lvm1,sda2,w']
```

Warning: Block device sharing

Block devices should typically only be shared between domains in a read-only fashion otherwise the Linux kernel's file systems will get very confused as the file system structure may change underneath them (having the same ext3 partition mounted `rw` twice is a sure fire way to cause irreparable damage)! Xend will attempt to prevent you from doing this by checking that the device is not mounted read-write in domain 0, and hasn't already been exported read-write to another domain. If you want read-write sharing, export the directory to other domains via NFS from domain 0 (or use a cluster file system such as GFS or ocfs2).

6.2 Using File-backed VBDs

It is also possible to use a file in Domain 0 as the primary storage for a virtual machine. As well as being convenient, this also has the advantage that the virtual block device will be *sparse* — space will only really be allocated as parts of the file are used. So if a virtual machine uses only half of its disk space then the file really takes up half of the size allocated.

For example, to create a 2GB sparse file-backed virtual block device (actually only consumes no disk space at all):

```
# dd if=/dev/zero of=vm1disk bs=1k seek=2048k count=0
```

Make a file system in the disk file:

```
# mkfs -t ext3 vm1disk
```

(when the tool asks for confirmation, answer 'y')

Populate the file system e.g. by copying from the current root:

```
# mount -o loop vm1disk /mnt
# cp -ax /{root,dev,var,etc,usr,bin,sbin,lib} /mnt
# mkdir /mnt/{proc,sys,home,tmp}
```

Tailor the file system by editing `/etc/fstab`, `/etc/hostname`, etc. Don't forget to edit the files in the mounted file system, instead of your domain 0 filesystem, e.g. you would edit `/mnt/etc/fstab` instead of `/etc/fstab`. For this example put `/dev/sda1` to root in fstab.

Now unmount (this is important!):

```
# umount /mnt
```

In the configuration file set:

```
disk = ['tap:aio:/full/path/to/vm1disk,sda1,w']
```

As the virtual machine writes to its 'disk', the sparse file will be filled in and consume more space up to the original 2GB.

Note: Users that have worked with file-backed VBDs on Xen in previous versions will be interested to know that this support is now provided through the blktap driver instead of the loopback driver. This change results in file-based block devices that are higher-performance, more scalable, and which provide better safety properties for VBD data. All that is required to update your existing file-backed VM configurations is to change VBD configuration lines from:

```
disk = ['file:/full/path/to/vm1disk,sda1,w']
```

to:

```
disk = ['tap:aio:/full/path/to/vm1disk,sda1,w']
```

6.2.1 Loopback-mounted file-backed VBDs (deprecated)

Note: *Loopback mounted VBDs have now been replaced with blktap-based support for raw image files, as described above. This section remains to detail a configuration that was used by older Xen versions.*

Raw image file-backed VBDs may also be attached to VMs using the Linux loopback driver. The only required change to the raw file instructions above are to specify the configuration entry as:

```
disk = ['file:/full/path/to/vm1disk,sda1,w']
```

Note that loopback file-backed VBDs may not be appropriate for backing I/O-intensive domains. This approach is known to experience substantial slowdowns under heavy I/O workloads, due to the I/O handling by the loopback block device used to support file-backed VBDs in dom0. Loopback support remains for old Xen installations, and users are strongly encouraged to use the blktap-based file support (using "`tap:aio`" as described above).

Additionally, Linux supports a maximum of eight loopback file-backed VBDs across all domains by default. This limit can be statically increased by using the *max_loop* module parameter if CONFIG_BLK_DEV_LOOP is compiled as a module in the dom0 kernel, or by using the *max_loop=n* boot option if CONFIG_BLK_DEV_LOOP is compiled directly into the dom0 kernel. Again, users are encouraged to use the blktap-based file support described above which scales to much larger number of active VBDs.

6.3 Using LVM-backed VBDs

A particularly appealing solution is to use LVM volumes as backing for domain filesystems since this allows dynamic growing/shrinking of volumes as well as snapshot and other features.

To initialize a partition to support LVM volumes:

```
# pvcreate /dev/sda10
```

Create a volume group named 'vg' on the physical partition:

```
# vgcreate vg /dev/sda10
```

Create a logical volume of size 4GB named 'myvmdisk1':

```
# lvcreate -L4096M -n myvmdisk1 vg
```

You should now see that you have a `/dev/vg/myvmdisk1` Make a filesystem, mount it and populate it, e.g.:

```
# mkfs -t ext3 /dev/vg/myvmdisk1
# mount /dev/vg/myvmdisk1 /mnt
# cp -ax / /mnt
# umount /mnt
```

Now configure your VM with the following disk configuration:

```
disk = [ 'phy:vg/myvmdisk1,sda1,w' ]
```

LVM enables you to grow the size of logical volumes, but you'll need to resize the corresponding file system to make use of the new space. Some file systems (e.g. ext3) now support online resize. See the LVM manuals for more details.

You can also use LVM for creating copy-on-write (CoW) clones of LVM volumes (known as writable persistent snapshots in LVM terminology). This facility is new in Linux 2.6.8, so isn't as stable as one might hope. In particular, using lots of CoW LVM disks consumes a lot of dom0 memory, and error conditions such as running out of disk space are not handled well. Hopefully this will improve in future.

To create two copy-on-write clones of the above file system you would use the following commands:

```
# lvcreate -s -L1024M -n myclonedisk1 /dev/vg/myvmdisk1
# lvcreate -s -L1024M -n myclonedisk2 /dev/vg/myvmdisk1
```

Each of these can grow to have 1GB of differences from the master volume. You can grow the amount of space for storing the differences using the lvextend command, e.g.:

```
# lvextend +100M /dev/vg/myclonedisk1
```

Don't let the 'differences volume' ever fill up otherwise LVM gets rather confused. It may be possible to automate the growing process by using `dmsetup wait` to spot the volume getting full and then issue an `lvextend`.

In principle, it is possible to continue writing to the volume that has been cloned (the changes will not be visible to the clones), but we wouldn't recommend this: have the cloned volume as a 'pristine' file system install that isn't mounted directly by any of the virtual machines.

6.4 Using NFS Root

First, populate a root filesystem in a directory on the server machine. This can be on a distinct physical machine, or simply run within a virtual machine on the same node.

Now configure the NFS server to export this filesystem over the network by adding a line to `/etc/exports`, for instance:

```
/export/vm1root      192.0.2.4/24 (rw,sync,no_root_squash)
```

Finally, configure the domain to use NFS root. In addition to the normal variables, you should make sure to set the following values in the domain's configuration file:

```
root       = '/dev/nfs'
nfs_server = '2.3.4.5'       # substitute IP address of server
nfs_root   = '/path/to/root' # path to root FS on the server
```

The domain will need network access at boot time, so either statically configure an IP address using the config variables `ip`, `netmask`, `gateway`, `hostname`; or enable DHCP (`dhcp='dhcp'`).

Note that the Linux NFS root implementation is known to have stability problems under high load (this is not a Xen-specific problem), so this configuration may not be appropriate for critical servers.

Chapter 7

CPU Management

Xen allows a domain's virtual CPU(s) to be associated with one or more host CPUs. This can be used to allocate real resources among one or more guests, or to make optimal use of processor resources when utilizing dual-core, hyperthreading, or other advanced CPU technologies.

Xen enumerates physical CPUs in a 'depth first' fashion. For a system with both hyperthreading and multiple cores, this would be all the hyperthreads on a given core, then all the cores on a given socket, and then all sockets. I.e. if you had a two socket, dual core, hyperthreaded Xeon the CPU order would be:

socket0				socket1			
core0		core1		core0		core1	
ht0	ht1	ht0	ht1	ht0	ht1	ht0	ht1
#0	#1	#2	#3	#4	#5	#6	#7

Having multiple vcpus belonging to the same domain mapped to the same physical CPU is very likely to lead to poor performance. It's better to use 'vcpus-set' to hot-unplug one of the vcpus and ensure the others are pinned on different CPUs.

If you are running IO intensive tasks, its typically better to dedicate either a hyperthread or whole core to running domain 0, and hence pin other domains so that they can't use CPU 0. If your workload is mostly compute intensive, you may want to pin vcpus such that all physical CPU threads are available for guest domains.

Chapter 8

Migrating Domains

8.1 Domain Save and Restore

The administrator of a Xen system may suspend a virtual machine's current state into a disk file in domain 0, allowing it to be resumed at a later time.

For example you can suspend a domain called "VM1" to disk using the command:

```
# xm save VM1 VM1.chk
```

This will stop the domain named "VM1" and save its current state into a file called `VM1.chk`.

To resume execution of this domain, use the `xm restore` command:

```
# xm restore VM1.chk
```

This will restore the state of the domain and resume its execution. The domain will carry on as before and the console may be reconnected using the `xm console` command, as described earlier.

8.2 Migration and Live Migration

Migration is used to transfer a domain between physical hosts. There are two varieties: regular and live migration. The former moves a virtual machine from one host to another by pausing it, copying its memory contents, and then resuming it on the destination. The latter performs the same logical functionality but without needing to pause the domain for the duration. In general when performing live migration the domain continues its usual activities and—from the user's perspective—the migration should be imperceptible.

To perform a live migration, both hosts must be running Xen / xend and the destination host must have sufficient resources (e.g. memory capacity) to accommodate the domain after the move. Furthermore we currently require both source and destination machines to be on the same L2 subnet.

Currently, there is no support for providing automatic remote access to filesystems stored on local disk when a domain is migrated. Administrators should choose an appropriate storage solution (i.e. SAN, NAS, etc.) to ensure that domain filesystems are also available on their destination node. GNBD is a good method for exporting a volume from one machine to another. iSCSI can do a similar job, but is more complex to set up.

When a domain migrates, it's MAC and IP address move with it, thus it is only possible to migrate VMs within the same layer-2 network and IP subnet. If the destination node is on a different subnet, the administrator would need to manually configure a suitable etherip or IP tunnel in the domain 0 of the remote node.

A domain may be migrated using the `xm migrate` command. To live migrate a domain to another machine, we would use the command:

```
# xm migrate --live mydomain destination.ournetwork.com
```

Without the `--live` flag, xend simply stops the domain and copies the memory image over to the new node and restarts it. Since domains can have large allocations this can be quite time consuming, even on a Gigabit network. With the `--live` flag xend attempts to keep the domain running while the migration is in progress, resulting in typical down times of just 60–300ms.

For now it will be necessary to reconnect to the domain's console on the new machine using the `xm console` command. If a migrated domain has any open network connections then they will be preserved, so SSH connections do not have this limitation.

Chapter 9

Securing Xen

This chapter describes how to secure a Xen system. It describes a number of scenarios and provides a corresponding set of best practices. It begins with a section devoted to understanding the security implications of a Xen system.

9.1 Xen Security Considerations

When deploying a Xen system, one must be sure to secure the management domain (Domain-0) as much as possible. If the management domain is compromised, all other domains are also vulnerable. The following are a set of best practices for Domain-0:

1. **Run the smallest number of necessary services.** The less things that are present in a management partition, the better. Remember, a service running as root in the management domain has full access to all other domains on the system.
2. **Use a firewall to restrict the traffic to the management domain.** A firewall with default-reject rules will help prevent attacks on the management domain.
3. **Do not allow users to access Domain-0.** The Linux kernel has been known to have local-user root exploits. If you allow normal users to access Domain-0 (even as unprivileged users) you run the risk of a kernel exploit making all of your domains vulnerable.

9.2 Driver Domain Security Considerations

Driver domains address a range of security problems that exist regarding the use of device drivers and hardware. On many operating systems in common use today, device drivers run within the kernel with the same privileges as the kernel. Few or no mechanisms exist to protect the integrity of the kernel from a misbehaving (read ”buggy”) or malicious device driver. Driver domains exist to aid in isolating a device driver within its own virtual machine where it cannot affect the stability and integrity of other domains. If a driver crashes, the driver domain can be restarted rather than have the entire machine crash (and restart) with it. Drivers written by unknown or untrusted third-parties can be confined to an isolated space. Driver domains thus address a number of security and stability issues with device drivers.

However, due to limitations in current hardware, a number of security concerns remain that need to be considered when setting up driver domains (it should be noted that the following list is not intended to be exhaustive).

1. **Without an IOMMU, a hardware device can DMA to memory regions outside of its controlling domain.** Architectures which do not have an IOMMU (e.g. most x86-based platforms) to restrict DMA usage by hardware are vulnerable. A hardware device which can perform arbitrary memory reads and writes can read/write outside of the memory of its controlling domain. A malicious or misbehaving domain could use a hardware device it controls to send data overwriting memory in another domain or to read arbitrary regions of memory in another domain.

2. **Shared buses are vulnerable to sniffing.** Devices that share a data bus can sniff (and possible spoof) each others' data. Device A that is assigned to Domain A could eavesdrop on data being transmitted by Domain B to Device B and then relay that data back to Domain A.

3. **Devices which share interrupt lines can either prevent the reception of that interrupt by the driver domain or can trigger the interrupt service routine of that guest needlessly.** A devices which shares a level-triggered interrupt (e.g. PCI devices) with another device can raise an interrupt and never clear it. This effectively blocks other devices which share that interrupt line from notifying their controlling driver domains that they need to be serviced. A device which shares an any type of interrupt line can trigger its interrupt continually which forces execution time to be spent (in multiple guests) in the interrupt service routine (potentially denying time to other processes within that guest). System architectures which allow each device to have its own interrupt line (e.g. PCI's Message Signaled Interrupts) are less vulnerable to this denial-of-service problem.

4. **Devices may share the use of I/O memory address space.** Xen can only restrict access to a device's physical I/O resources at a certain granularity. For interrupt lines and I/O port address space, that granularity is very fine (per interrupt line and per I/O port). However, Xen can only restrict access to I/O memory address space on a page size basis. If more than one device shares use of a page in I/O memory address space, the domains to which those devices are assigned will be able to access the I/O memory address space of each other's devices.

9.3 Security Scenarios

9.3.1 The Isolated Management Network

In this scenario, each node has two network cards in the cluster. One network card is connected to the outside world and one network card is a physically isolated management network specifically for Xen instances to use.

As long as all of the management partitions are trusted equally, this is the most secure scenario. No additional configuration is needed other than forcing Xend to bind to the management interface for relocation.

9.3.2 A Subnet Behind a Firewall

In this scenario, each node has only one network card but the entire cluster sits behind a firewall. This firewall should do at least the following:

1. Prevent IP spoofing from outside of the subnet.
2. Prevent access to the relocation port of any of the nodes in the cluster except from within the cluster.

The following iptables rules can be used on each node to prevent migrations to that node from outside the subnet assuming the main firewall does not do this for you:

```
# this command disables all access to the Xen relocation
# port:
iptables -A INPUT -p tcp --destination-port 8002 -j REJECT

# this command enables Xen relocations only from the specific
# subnet:
iptables -I INPUT -p tcp -{}-source 192.0.2.0/24 \
    --destination-port 8002 -j ACCEPT
```

9.3.3 Nodes on an Untrusted Subnet

Migration on an untrusted subnet is not safe in current versions of Xen. It may be possible to perform migrations through a secure tunnel via an VPN or SSH. The only safe option in the absence of a secure tunnel is to disable migration completely. The easiest way to do this is with iptables:

```
# this command disables all access to the Xen relocation port
iptables -A INPUT -p tcp -{}-destination-port 8002 -j REJECT
```

Chapter 10

sHype/Xen Access Control

The Xen mandatory access control framework is an implementation of the sHype Hypervisor Security Architecture (www.research.ibm.com/ssd_shype). It permits or denies communication and resource access of domains based on a security policy. The mandatory access controls are enforced in addition to the Xen core controls, such as memory protection. They are designed to remain transparent during normal operation of domains (policy-conform behavior) but to intervene when domains move outside their intended sharing behavior. This chapter will describe how the sHype access controls in Xen can be configured to prevent viruses from spilling over from one into another workload type and secrets from leaking from one workload type to another. sHype/Xen depends on the correct behavior of Domain-0 (cf previous chapter).

Benefits of configuring sHype/ACM in Xen include:

- robust workload and resource protection effective against rogue user domains
- simple, platform- and operating system-independent security policies (ideal for heterogeneous distributed environments)
- safety net with minimal performance overhead in case operating system security is missing, does not scale, or fails

These benefits are very valuable because today's operating systems become increasingly complex and often have no or insufficient mandatory access controls. (Discretionary access controls, supported by most operating systems, are not effective against viruses or misbehaving programs.) Where mandatory access control exists (e.g., SELinux), they usually deploy platform-specific, complex, and difficult to understand security policies. Multi-tier applications in business environments typically require different operating systems (e.g., AIX, Windows, Linux) in different tiers. Related distributed transactions and workloads cannot be easily protected on the OS level. The Xen access control framework steps in to offer a coarse-grained but very robust and consistent security layer and safety net across different platforms and operating systems.

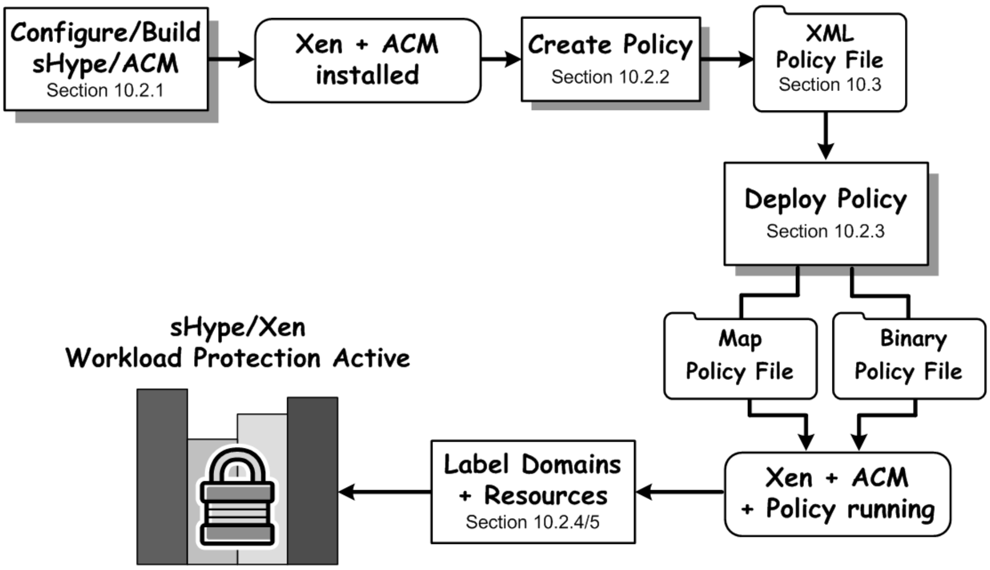

Figure 10.1: Overview of activating sHype workload protection in Xen. Section numbers point to representative examples.

To control sharing between domains, Xen mediates all inter-domain communication (shared memory, events) as well as the access of domains to resources such as storage disks. Thus, Xen can confine distributed workloads (domain payloads) by permitting sharing among domains running the same type of workload and denying sharing between pairs of domains that run different workload types. We assume that–from a Xen perspective–only one workload type is running per user domain. To enable Xen to associate domains and resources with workload types, security labels including the workload types are attached to domains and resources. These labels and the hypervisor sHype controls cannot be manipulated or bypassed by user domains and are effective even against compromised or rogue domains.

10.1 Overview

This section gives an overview of how workloads can be protected using the sHype mandatory access control framework in Xen. Figure 10.1 shows the necessary steps in activating the Xen workload protection. These steps are described in detail in Section 10.2.

First, the sHype/ACM access control must be enabled in the Xen distribution and the distribution must be built and installed (cf Subsection 10.2.1). Before we can enforce security, a Xen security policy must be created (cf Subsection 10.2.2) and deployed (cf Subsection 10.2.3). This policy defines the workload types differentiated during access control. It also defines the rules that compare workload types of domains and resources to decide about access requests. Workload types are represented by security labels that can be securely associated to domains and resources (cf Subsections 10.2.4 and 10.2.5). The functioning of the active sHype/Xen workload protection is demonstrated using simple resource assignment, and domain creation tests in Subsection 10.2.6. Section 10.3 describes the syntax and semantics of the sHype/Xen security policy in detail and introduces briefly the tools that are available to help you create your own sHype security policies.

The next section describes all the necessary steps to create, deploy, and test a simple workload protection policy. It is meant to enable Xen users and developers to quickly try out the sHype/Xen workload protection. Those readers who are interested in learning more about how the sHype access control in Xen works and how it is configured using the XML security policy should read Section 10.3 as well. Section 10.4 concludes this chapter with current limitations of the sHype implementation for Xen.

10.2 Xen Workload Protection Step-by-Step

You are about to configure and deploy the Xen sHype workload protection by following 5 simple steps:

- configure and install sHype/Xen
- create a simple workload protection security policy
- deploy the sHype/Xen security policy
- associate domains and resources with workload labels,
- test the workload protection

The essential commands to create and deploy an sHype/Xen security policy are numbered throughout the following sections. If you want a quick-guide or return at a later time to go quickly through this demonstration, simply look for the numbered commands and apply them in order.

10.2.1 Configuring/Building sHype Support into Xen

First, we need to configure the access control module in Xen and install the ACM-enabled Xen hypervisor. This step installs security tools and compiles sHype/ACM controls into the Xen hypervisor.

To enable sHype/ACM in Xen, please edit the Config.mk file in the top Xen directory.

```
(1) In Config.mk
      Change: XSM_ENABLE ?= n
          To: XSM_ENABLE ?= y

      Change: ACM_SECURITY ?= n
```

```
To: ACM_SECURITY ?= y
```

Then install the security-enabled Xen environment as follows:

```
(2) # make world
    # make install
```

Reboot into the security-enabled Xen hypervisor.

```
(3) # reboot
```

Xen will boot into the default security policy. After reboot, you can explore the simple DEFAULT policy.

```
# xm getpolicy
Supported security subsystems   : ACM
Policy name           : DEFAULT
Policy type           : ACM
Version of XML policy : 1.0
Policy configuration  : loaded

# xm labels
SystemManagement

# xm list --label
Name       ID   Mem VCPUs      State   Time(s) Label
Domain-0    0   941     1     r-----      38.1  ACM:DEFAULT:SystemManagement
```

In this state, no domains can be started. Now, a policy can be created and loaded into the hypervisor.

10.2.2 Creating A WLP Policy in 3 Simple Steps with ezPolicy

We will use the ezPolicy tool to quickly create a policy that protects workloads. You will need both the Python and wxPython packages to run this tool. To run the tool in Domain-0, you can download the wxPython package from www.wxpython.org or use the command `yum install wxPython` in Redhat/Fedora. To run the tool on MS Windows, you also need to download the Python package from www.python.org. After these packages are installed, start the ezPolicy tool with the following command:

```
(4) # xensec_ezpolicy
```

Figure 10.2 shows a screen-shot of the tool. The following steps illustrate how you can create the workload definition shown in Figure 10.2. You can use `<CTRL>-h` to pop up a help window at any time. The indicators (a), (b), and (c) in Figure 10.2 show the buttons that are used during the 3 steps of creating a policy:

1. defining workloads
2. defining run-time conflicts
3. translating the workload definition into an sHype/Xen access control policy

Defining workloads. Workloads are defined for each organization and department that you enter in the left panel.

To ease the transition from an unlabeled to a fully labeled workload-protection environment, we have added support to sHype/Xen to run unlabeled domains accessing unlabeled resources in addition to labeled domains accessing labeled resources.

Support for running unlabeled domains on sHype/Xen is enabled by adding the predefined workload type and label `__UNLABELED__` to the security policy. (This is a double underscore followed by the string ”`UNLABELED`” followed by a double underscore.) The ezPolicy tool automatically adds this organization-level workload type to a new workload definition (cf Figure 10.2). It can simply be deleted from the workload definition if no such support is desired. If unlabeled domains are supported in the policy, then any domain or resource that has no label will implicitly inherit this label when access control decisions are made. In effect, unlabeled domains and resources define a new workload type `__UNLABELED__`, which is confined from any other labeled workload.

Please use now the “New Org” button to add the organization workload types “A-Bank”, “B-Bank”, and “AutoCorp”.

You can refine an organization to differentiate between multiple department workloads by right-clicking the organization and selecting `Add Department` (or selecting an organization and pressing `<CRTL>-a`). Create department workloads “SecurityUnderwriting”, and “MarketAnalysis” for the “A-Bank”. The resulting layout of the tool should be similar to the left panel shown in Figure 10.2.

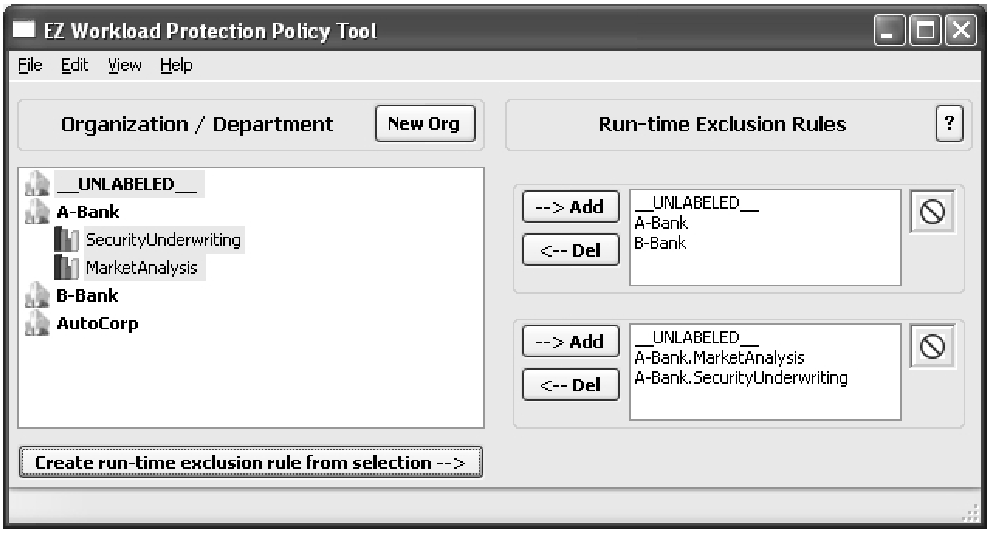

Figure 10.2: Final layout including workload definition and Run-time Exclusion rules.

Defining run-time conflicts. Workloads that shall be prohibited from running concurrently on the same hypervisor platform are grouped into “Run-time Exclusion rules” on the right panel of the window. Cautious users should include the `__UNLABELED__` workload type in all run-time exclusion rules because any workload could run inside unlabeled domains.

To prevent A-Bank and B-Bank workloads (including their departmental workloads) from running simultaneously on the same hypervisor system, select the organization “A-Bank” and, while pressing the `<CTRL>`-key, select the organization “B-Bank”. Being cautious, we also prevent unlabeled workloads from running with any of those workloads by pressing the `<CTRL>`-key and selecting “__UNLABELED__”. Now press the button named “Create run-time exclusion rule from selection”. A popup window will ask for the name for this run-time exclusion rule (enter a name or just hit `<ENTER>`). A rule will appear on the right panel. The name is used as reference only and does not affect access control decisions.

Please repeat this process to create another run-time exclusion rule for the department workloads “A-Bank.SecurityUnderwriting”, “A-Bank.MarketAnalysis”. Also add the “__UNLABELED__” workload type to this conflict set.

The resulting layout of your window should be similar to Figure 10.2. Save this workload definition by selecting “Save Workload Definition as ...” in the “File” menu. This workload definition can be later refined if required.

Translating the workload definition into an sHype/Xen access control policy. To translate the workload definition into a access control policy understood by Xen, please select the “Save as Xen ACM Security Policy” in the “File” menu. Enter the following policy name in the popup window: `mytest`. If you are running ezPolicy in Domain-0, the resulting policy file mytest_security-policy.xml will automatically be placed into the right directory (/etc/xen/acm-security/policies/). If you run the tool on another system, then you need to copy the resulting policy file into Domain-0 before continuing. See Section 10.3.1 for naming conventions of security policies.

Note: The support for `__UNLABELED__` domains and resources is meant to help transitioning from an uncontrolled environment to a workload-protected environment by starting with unlabeled domains and resources and then step-by-step labeling domains and resources. Once all workloads are labeled, the `__UNLABELED__` type can simply be removed from the Domain-0 label or from the policy through a policy update. Section 10.3.5 will show how unlabeled domains can be disabled by updating the `mytest` policy at run-time.

10.2.3 Deploying a WLP Policy

To deploy the workload protection policy we created in Section 10.2.2, we create a policy representation (mytest.bin), load it into the Xen hypervisor, and configure Xen to also load this policy during reboot.

The following command translates the source policy representation into a format that can be loaded into Xen with sHype/ACM support, activates the policy, and configures this policy for future boot cycles into the boot sequence. Please refer to the `xm` man page for further details:

```
(5) # xm setpolicy ACM mytest
    Successfully set the new policy.
    Supported security subsystems   : ACM
    Policy name           : mytest
    Policy type           : ACM
    Version of XML policy : 1.0
    Policy configuration  : loaded, activated for boot
```

Alternatively, if installing the policy fails (e.g., because it cannot identify the Xen boot entry), you can manually install the policy in 3 steps a-c.

(*Alternatively to 5 - step a*) Manually copy the policy binary file into the boot directory:

```
# cp /etc/xen/acm-security/policies/mytest.bin /boot/mytest.bin
```

(*Alternatively to 5 - step b*) Manually add a module line to your Xen boot entry so that grub loads this policy file during startup:

```
title XEN Devel with 2.6.18.8
     kernel /xen.gz
     module /vmlinuz-2.6.18.8-xen root=/dev/sda3 ro console=tty0
     module /initrd-2.6.18.8-xen.img
     module /mytest.bin
```

(*Alternatively to 5 - step c*) Reboot. Xen will choose the bootstrap label defined in the policy as Domain-0 label during reboot. After reboot, you can re-label Domain-0 at run-time, cf Section 10.2.7.

Assuming that command (5) succeeded or you followed the alternative instructions above, you should see the new policy and label appear when listing domains:

```
# xm list --label
Name       ID   Mem VCPUs     State   Time(s) Label
Domain-0    0   941     1     r-----     81.5  ACM:mytest:SystemManagement
```

If the security label at the end of the line says “INACTIVE” then the security is not enabled. Verify the previous steps. Note: Domain-0 is assigned a default label (see `bootstrap` policy attribute explained in Section 10.3). All other domains must be explicitly labeled, which we describe in detail below.

10.2.4 Labeling Unmanaged User Domains

Unmanaged domains are started in Xen by using a configuration file. Please refer to Section 10.2.8 if you are using managed domains.

The following configuration file defines `domain1`:

```
# cat domain1.xm
kernel= "/boot/vmlinuz-2.6.18.8-xen"
memory = 128
name = "domain1"
```

```
vif = ['']
dhcp = "dhcp"
disk = ['file:/home/xen/dom_fc5/fedora.fc5.img,sda1,w', \
        'file:/home/xen/dom_fc5/fedora.fc5.swap,sda2,w']
root = "/dev/sda1 ro xencons=tty"
```

Every domain must be associated with a security label before it can start on sHype/Xen. Otherwise, sHype/Xen would not be able to enforce the policy consistently. Our `mytest` policy is configured so that Xen assigns a default label `__UNLABELED__` to domains and resources that have no label and supports them in a controlled manner. Since neither the domain, nor the resources are (yet) labeled, this domain can start under the `mytest` policy:

```
# xm create domain1.xm
Using config file "./domain1.xm".
Started domain domain1

# xm list --label
Name      ID   Mem VCPUs      State   Time(s) Label
domain1    1   128     1     -b----       0.7  ACM:mytest:__UNLABELED__
Domain-0   0   875     1     r-----      84.6  ACM:mytest:SystemManagement
```

Please shutdown domain1 so that we can move it into the protection domain of workload `A-Bank`.

```
# xm shutdown domain1
(wait some seconds until the domain has shut down)

#xm list --label
Name       ID   Mem VCPUs      State   Time(s) Label
Domain-0    0   875     1     r-----      86.4  ACM:mytest:SystemManagement
```

We assume that the processing in domain1 contributes to the `A-Bank` workload. We explore now how to transition this domain into the “A-Bank” workload-protection. The following command prints all domain labels available in the active policy:

```
# xm labels
A-Bank
A-Bank.MarketAnalysis
A-Bank.SecurityUnderwriting
AutoCorp
B-Bank
SystemManagement
__UNLABELED__
```

Now label `domain1` with the A-Bank label and another `domain2` with the B-Bank label. Please refer to the xm man page for further information.

```
(6) # xm addlabel A-Bank dom domain1.xm
    # xm addlabel B-Bank dom domain2.xm
```

Let us try to start the domain again:

```
# xm create domain1.xm
Using config file "./domain1.xm".
Error: VM's access to block device 'file:/home/xen/dom_fc5/fedora.fc5.img' denied
```

This error indicates that `domain1`, if started, would not be able to access its image and swap files because they are not labeled. This makes sense because to confine workloads, access of domains to resources must be controlled. Otherwise, domains that are not allowed to communicate or run simultaneously could share data through storage resources.

10.2.5 Labeling Resources

You can use the `xm labels type=res` command to list available resource labels. Let us assign the A-Bank resource label to the `domain1` image file representing `/dev/sda1` and to its swap file:

```
(7) # xm addlabel A-Bank res \
    file:/home/xen/dom_fc5/fedora.fc5.img

    # xm addlabel A-Bank res \
    file:/home/xen/dom_fc5/fedora.fc5.swap
```

The following command lists all labeled resources on the system, e.g., to lookup or verify the labeling:

```
# xm resources
file:/home/xen/dom_fc5/fedora.fc5.swap
      type: ACM
    policy: mytest
     label: A-Bank
file:/home/xen/dom_fc5/fedora.fc5.img
      type: ACM
    policy: mytest
     label: A-Bank
```

Starting `domain1` will now succeed:

```
# xm create domain1.xm
Using config file "./domain1.xm".
Started domain domain1

# xm list --label
Name       ID   Mem VCPUs      State   Time(s) Label
domain1     3   128     1     -b----       0.8  ACM:mytest:A-Bank
Domain-0    0   875     1     r-----      90.9  ACM:mytest:SystemManagement
```

Currently, if a labeled resource is moved to another location, the label must first be manually removed, and after the move re-attached using the xm commands `rmlabel` and `addlabel` respectively. Please see Section 10.4 for further details.

```
(8) Label the resources of domain2 as B-Bank
    but please do not start this domain yet.
```

10.2.6 Testing The Xen Workload Protection

We are about to demonstrate the sHype/Xen workload protection by verifying

- that user domains with conflicting workloads cannot run simultaneously
- that user domains cannot access resources of workloads other than the one they are associated with
- that user domains cannot exchange network packets if they are not associated with the same workload type (not yet supported in Xen)

Test 1: Run-time exclusion rules. We assume that `domain1` with the A-Bank label is still running. While `domain1` is running, the run-time exclusion set of our policy implies that `domain2` cannot start because the label of `domain1` includes the CHWALL type A-Bank and the label of `domain2` includes the CHWALL type B-Bank. The run-time exclusion rule of our policy enforces that A-Bank and B-Bank cannot run at the same time on the same hypervisor platform. Once domain1 is stopped, saved, or migrated to another platform, `domain2` can start. Once `domain2` is started, however, `domain1` can no longer start or resume on this system. When creating the Chinese Wall types for the workload labels, the ezPolicy tool policy translation component ensures that department workloads inherit all the organization types (and with it any organization exclusions).

```
# xm list --label
Name       ID   Mem VCPUs      State   Time(s) Label
domain1     3   128     1     -b----      0.8  ACM:mytest:A-Bank
Domain-0    0   875     1     r-----     90.9  ACM:mytest:SystemManagement

# xm create domain2.xm
Using config file "./domain2.xm".
Error: 'Domain in conflict set with running domains'

# xm shutdown domain1
(wait some seconds until domain 1 is shut down)

# xm list --label
Name       ID   Mem VCPUs      State   Time(s) Label
Domain-0    0   873     1     r-----     95.3  ACM:mytest:SystemManagement

# xm create domain2.xm
Using config file "./domain2.xm".
Started domain domain2

# xm list --label
Name       ID   Mem VCPUs      State   Time(s) Label
domain2     5   164     1     -b----      0.3  ACM:mytest:B-Bank
Domain-0    0   839     1     r-----     96.4  ACM:mytest:SystemManagement

# xm create domain1.xm
Using config file "domain1.xm".
Error: 'Domain in conflict with running domains'

# xm shutdown domain2
# xm list --label
Name       ID   Mem VCPUs      State   Time(s) Label
Domain-0    0   839     1     r-----     97.8  ACM:mytest:SystemManagement
```

You can verify that domains with AutoCorp label can run together with domains labeled A-Bank or B-Bank.

Test2: Resource access. In this test, we will re-label the swap file for `domain1` with the `B-Bank` resource label. In a real environment, the swap file must be sanitized (scrubbed/zeroed) before it is reassigned to prevent data leaks from the A-Bank to the B-Bank workload through the swap file.

We expect that `domain1` will no longer start because it cannot access this resource. This test checks the sharing abilities of domains, which are defined by the Simple Type Enforcement Policy component.

```
# xm rmlabel res file:/home/xen/dom_fc5/fedora.fc5.swap

# xm addlabel B-Bank res file:/home/xen/dom_fc5/fedora.fc5.swap

# xm resources
file:/home/xen/dom_fc5/fedora.fc5.swap
      type: ACM
    policy: mytest
     label: B-Bank
file:/home/xen/dom_fc5/fedora.fc5.img
      type: ACM
    policy: mytest
     label: A-Bank

# xm create domain1.xm
Using config file "./domain1.xm".
Error:
VM's access to block device 'file:/home/xen/dom_fc5/fedora.fc5.swap' denied
```

The resource authorization checks are performed before the domain is actually started so that failures during the startup are prevented. A domain is only started if all the resources specified in its configuration are accessible.

Test 3: Communication. In this test we would verify that two domains with labels A-Bank and B-Bank cannot exchange network packets by using the 'ping' connectivity test. It is also related to the STE policy. **Note:** sHype/Xen does control direct communication between domains. However, domains associated with different workloads can currently still communicate through the Domain-0 virtual network. We are working on the sHype/ACM controls for local and remote network traffic through Domain-0. Please monitor the xen-devel mailing list for updated information.

10.2.7 Labeling Domain-0 –or– Restricting System Authorization

The major use case for explicitly labeling or relabeling Domain-0 is to restrict or extend which workload types can run on a virtualized Xen system. This enables flexible partitioning of the physical infrastructure as well as the workloads running on it in a multi-platform environment.

In case no Domain-0 label is explicitly stated, we automatically assigned Domain-0 the `SystemManagement` label, which includes all STE (workload) types that are known to the policy. In effect, the Domain-0 label authorizes the Xen system to run only those workload types, whose STE types are included in the Domain-0 label. Hence, choosing the `SystemManagement` label for Domain-0 permits any labeled domain to run. Resetting the label for Domain-0 at boot or run-time to a label with a subset of the known STE workload types restricts which user domains can run on this system. If Domain-0 is relabeled at run-time, then the new label must at least include all STE types of those domains that are currently running. The operation fails otherwise. This requirement ensures that the system remains in a valid security configuration after relabelling.

Restricting the Domain-0 authorization through the label creates a flexible policy-driven way to strongly partition the physical infrastructure and the workloads running on it. This partitioning will be automatically enforced during migration, start, or resume of domains and simplifies the security management considerably. Strongly competing workloads can be forced to run on separate physical infrastructure and become less depend on the domain isolation capabilities of the hypervisor.
First, we relabel the swap image back to A-Bank and then start up domain1:

```
# xm rmlabel res file:/home/xen/dom_fc5/fedora.fc5.swap

# xm addlabel A-Bank res file:/home/xen/dom_fc5/fedora.fc5.swap

# xm create domain1.xm
Using config file "./domain1.xm".
Started domain domain1

# xm list --label
Name        ID   Mem VCPUs      State   Time(s) Label
domain1      7   128     1     -b----       0.7  ACM:mytest:A-Bank
Domain-0     0   839     1     r-----     103.1  ACM:mytest:SystemManagement
```

The following command will restrict the Xen system to only run STE types included in the A-Bank label.

```
# xm addlabel A-Bank mgt Domain-0
Successfully set the label of domain 'Domain-0' to 'A-Bank'.

# xm list --label
Name        ID   Mem VCPUs      State   Time(s) Label
Domain-0     0   839     1     r-----     103.7  ACM:mytest:A-Bank
domain1      7   128     1     -b----       0.7  ACM:mytest:A-Bank
```

In our example policy in Figure 10.4, this means that only `A-Bank` domains and workloads (types) can run after the successful completion of this command because the `A-Bank` label includes only a single STE type, namely `A-Bank`. This command fails if any running domain has an STE type in its label that is not included in the A-Bank label.

If we now label a domain3 with AutoCorp, it cannot start because Domain-0 is no longer authorized to run the workload type `AutoCorp`.

```
# xm addlabel AutoCorp dom domain3.xm
  (remember to label its resources, too)

# xm create domain3.xm
Using config file "./domain3.xm".
Error: VM is not authorized to run.

# xm list --label
Name       ID   Mem VCPUs      State   Time(s) Label
Domain-0    0   839     1     r-----    104.7  ACM:mytest:A-Bank
domain1     7   128     1     -b----      0.7  ACM:mytest:A-Bank
```

At this point, unlabeled domains cannot start either. Let domain4.xm describe an unlabeled domain, then trying to start domain4 will fail:

```
# xm getlabel dom domain4.xm
Error: 'Domain not labeled'

# xm create domain4.xm
Using config file "./domain4.xm".
Error: VM is not authorized to run.
```

Relabeling Domain-0 with the SystemManagement label will enable domain3 to start.

```
# xm addlabel SystemManagement mgt Domain-0
Successfully set the label of domain 'Domain-0' to 'SystemManagement'.

# xm list --label
Name       ID   Mem VCPUs      State   Time(s) Label
domain1     7   128     1     -b----      0.8  ACM:mytest:A-Bank
Domain-0    0   839     1     r-----    106.6  ACM:mytest:SystemManagement

# xm create domain3.xm
Using config file "./domain3.xm".
Started domain domain3

# xm list --label
Name       ID   Mem VCPUs      State   Time(s) Label
domain1     7   128     1     -b----      0.8 ACM:mytest:A-Bank
domain3     8   164     1     -b----      0.3 ACM:mytest:AutoCorp
Domain-0    0   711     1     r-----    107.6 ACM:mytest:SystemManagement
```

10.2.8 Labeling Managed User Domains

Xend has been extended with functionality to manage domains along with their configuration information. Such domains are configured and started via Xen-API calls. Since managed domains do not have an associated xm configuration file, the existing `addlabel` command, which adds the security label into a domain's configuration file, will not work for such managed domains.

Therefore, we have extended the `xm addlabel` and `xm rmlabel` subcommands to enable adding security labels to and removing security labels from managed domain configurations. The following example shows how the `A-Bank` label can be assigned to the xend-managed domain configuration of `domain1`. Removing labels from managed user domain configurations works similarly.

Below, we show a dormant configuration of the managed domain1 with ID `"-1"` and state `"-----"` before labeling:

```
# xm list --label
Name       ID   Mem VCPUs      State   Time(s) Label
domain1    -1   128     1     ------       0.0 ACM:mytest:__UNLABELED__
Domain-0    0   711     1     r-----     128.4 ACM:mytest:SystemManagement
```

Now we label the managed domain:

```
# xm addlabel A-Bank mgt domain1
Successfully set the label of the dormant domain 'domain1' to 'A-Bank'.
```

After labeling, you can see that the security label is part of the domain configuration:

```
# xm list --label
Name       ID   Mem VCPUs      State   Time(s) Label
domain1    -1   128     1     ------       0.0  ACM:mytest:A-Bank
Domain-0    0   711     1     r-----     129.7  ACM:mytest:SystemManagement
```

This command extension does not support relabeling of individual running user domains for several reasons. For one, because of the difficulty to revoke resources in cases where a running domain's new label does not permit access to resources that were accessible under the old label. Another reason is that changing the label of a single domain of a workload is rarely a good choice and will affect the workload isolation properties of the overall workload.

However, the name and contents of the label associated with running domains can be indirectly changed through a global policy change, which will update the whole workload consistently (domains and resources), cf. Section 10.3.5.

10.3 Xen Access Control Policy

This section describes the sHype/Xen access control policy in detail. It gives enough information to enable the reader to write custom access control policies and to use the available Xen policy tools. The policy language is expressive enough to specify most symmetric access relationships between domains and resources efficiently.

The Xen access control policy consists of two policy components. The first component, called Simple Type Enforcement (STE) policy, controls the sharing between running domains, i.e., communication or access to shared resources. The second component, called Chinese Wall (CHWALL) policy, controls which domains can run simultaneously on the same virtualized platform. The CHWALL and STE policy components complement each other. The XML policy file includes all information needed by Xen to enforce those policies.

Figures 10.3 and 10.4 show the fully functional but very simple example Xen security policy that is created by ezPolicy as shown in Figure 10.2. The policy can distinguish the 6 workload types shown in lines 11-17 in Fig. 10.3. The whole XML Security Policy consists of four parts:

1. Policy header including the policy name
2. Simple Type Enforcement block

```
01  <?xml version="1.0" ?>
02  <!-- Auto-generated by ezPolicy        -->
03  <SecurityPolicyDefinition ...">
04      <PolicyHeader>
05          <PolicyName>mytest</PolicyName>
06          <Date>Mon Nov 19 22:51:56 2007</Date>
07          <Version>1.0</Version>
08      </PolicyHeader>
09      <SimpleTypeEnforcement>
10          <SimpleTypeEnforcementTypes>
11              <Type>SystemManagement</Type>
12              <Type>__UNLABELED__</Type>
13              <Type>A-Bank</Type>
14              <Type>A-Bank.SecurityUnderwriting</Type>
15              <Type>A-Bank.MarketAnalysis</Type>
16              <Type>B-Bank</Type>
17              <Type>AutoCorp</Type>
18          </SimpleTypeEnforcementTypes>
19      </SimpleTypeEnforcement>
20      <ChineseWall priority="PrimaryPolicyComponent">
21          <ChineseWallTypes>
22              <Type>SystemManagement</Type>
23              <Type>__UNLABELED__</Type>
24              <Type>A-Bank</Type>
25              <Type>A-Bank.SecurityUnderwriting</Type>
26              <Type>A-Bank.MarketAnalysis</Type>
27              <Type>B-Bank</Type>
28              <Type>AutoCorp</Type>
29          </ChineseWallTypes>
30          <ConflictSets>
31              <Conflict name="RER">
32                  <Type>A-Bank</Type>
33                  <Type>B-Bank</Type>
34                  <Type>__UNLABELED__</Type>
35              </Conflict>
36              <Conflict name="RER">
37                  <Type>A-Bank.MarketAnalysis</Type>
38                  <Type>A-Bank.SecurityUnderwriting</Type>
39                  <Type>__UNLABELED__</Type>
40              </Conflict>
41         </ConflictSets>
42      </ChineseWall>
```

Figure 10.3: Example XML security policy file – Part I: Types and Rules Definition.

3. Chinese Wall Policy block
4. Label definition block

10.3.1 Policy Header and Policy Name

Lines 1-2 (cf Figure 10.3) include the usual XML header. The security policy definition starts in Line 3 and refers to the policy schema. The XML-Schema definition for the Xen policy can be found in the file */etc/xen/acm-security/policies/security-policy.xsd*. Examples for security policies can be found in the example subdirectory. The acm-security directory is only installed if ACM security is configured during installation (cf Section 10.2.1).

The `Policy Header` spans lines 4-8. It includes a date field and defines the policy name mytest as well as the version of the XML. It can also include optional fields that are not shown and are for future use (see schema definition).

The policy name serves two purposes: First, it provides a unique name for the security policy. This name is also exported by the Xen hypervisor to the Xen management tools in order to ensure that both the Xen hypervisor and Domain-0 enforce the same policy. We plan to extend the policy name with a digital fingerprint of the policy contents to better protect this correlation. Second, it implicitly points the xm tools to the location where the XML policy file is stored on the Xen system. Replacing the colons in the policy name by slashes yields the local path to the policy file starting from the global policy directory /etc/xen/acm-security/policies. The last part of the policy name is the prefix for the XML policy file name, completed by -security_policy.xml. Our example policy with the name mytest can be found in the XML policy file named mytest-security_policy.xml that is stored under the global policy directory. Another, preinstalled example policy named example.test can be found in the test-security_policy.xml under /etc/xen/acm-security/policies/example.

10.3.2 Simple Type Enforcement Policy Component

The Simple Type Enforcement (STE) policy controls which domains can communicate or share resources. This way, Xen can enforce confinement of workload types by confining the domains running those workload types and their resources. The mandatory access control framework enforces its policy when domains access intended communication or cooperation means (shared memory, events, shared resources such as block devices). It builds on top of the core hypervisor isolation, which restricts the ways of inter-communication to those intended means. STE does not protect or intend to protect from covert channels in the hypervisor or hardware; this is an orthogonal problem that can be mitigated by using the Run-time Exclusion rules described above or by fixing the problem leading to those covert channels in the core hypervisor or hardware platform.

Xen controls sharing between domains on the resource and domain level because this is the abstraction the hypervisor and its management understand naturally. While this is coarse-grained, it is also very reliable and robust and it requires minimal changes to implement mandatory access controls in the hypervisor. It enables platform- and operating system-independent policies as part of a layered security approach.

Lines 11-17 (cf Figure 10.3) define the Simple Type Enforcement policy component. Essentially, they define the workload type names SystemManagement, A-Bank, AutoCorp etc. that are available in the STE policy component. The policy rules are implicit: Xen permits two domains to communicate with each other if and only if their security labels have at least one STE type in common. Similarly, Xen permits a user domain to access a resource if and only if the labels of the domain and the resource have at least one STE workload type in common.

10.3.3 Chinese Wall Policy Component

The Chinese Wall security policy interpretation of sHype enables users to prevent certain workloads from running simultaneously on the same hypervisor platform. Run-time Exclusion rules (RER), also called Conflict Sets or Anti-Collocation rules, define a set of workload types that are not permitted to run simultaneously on the same virtualized platform. Of all the workloads specified in a Run-time Exclusion rule, at most one type can run on the same hypervisor platform at a time. Run-time Exclusion Rules implement a less rigorous variant of the original Chinese Wall security component. They do not implement the *-property of the policy, which would require to restrict also types that are not part of an exclusion rule once they are running together with a type in an exclusion rule (http://www.gammassl.co.uk/topics/chinesewall.html provides more information on the original Chinese Wall policy).

Xen considers the `ChineseWallTypes` part of the label for the enforcement of the Run-time Exclusion rules. It is illegal to define labels including conflicting Chinese Wall types.

Lines 20-41 (cf Figure 10.3) define the Chinese Wall policy component. Lines 22-28 define the known Chinese Wall types, which coincide here with the STE types defined above. This usually holds if the criteria for sharing among domains and sharing of the hardware platform are the same. Lines 30-41 define one Run-time Exclusion rules, the first of which is depicted below:

```
31  <Conflict name="RER">
32    <Type>A-Bank</Type>
33    <Type>B-Bank</Type>
34    <Type>__UNLABELED__</Type>
35  </Conflict>
```

Based on this rule, Xen enforces that only one of the types `A-Bank`, `B-Bank`, or `__UNLABELED__` will run on a single hypervisor platform at a time. For example, once a domain assigned a `A-Bank` workload type is started, domains with the `B-Bank` type or unlabeled domains will be denied to start. When the former domain stops and no other domains with the `A-Bank` type are running, then domains with the `B-Bank` type or unlabeled domains can start.

Xen maintains reference counts on each running workload type to keep track of which workload types are running. Every time a domain starts or resumes, the reference count on those Chinese Wall types that are referenced in the domain's label are incremented. Every time a domain is destroyed or saved, the reference counts of its Chinese Wall types are decremented. sHype in Xen fully supports migration and live-migration, which is subject to access control the same way as saving a domain on the source platform and resuming it on the destination platform.

Here are some reasons why users might want to restrict workloads or domains from sharing the system hardware simultaneously:

- Imperfect resource management or control might enable a compromised user domain to starve other domains and the workload running in them.
- Redundant user domains might run the same workload to increase availability; such domains should not run on the same hardware to avoid single points of failure.
- Imperfect Xen core domain isolation might enable two rogue domains running different workload types to use unintended and unknown ways (covert channels) to exchange some bits of information. This way, they bypass the policed Xen access control mechanisms. Such imperfections cannot be completely eliminated and are a result of trade-offs between security and other design requirements. For a simple example of a covert channel see http://www.multicians.org/timing-chn.html. Such covert channels exist also between workloads running on different platforms if they are connected through networks. The Xen Chinese Wall policy provides an approximated "air-gap" between selected workload types.

10.3.4 Security Labels

To enable Xen to associate domains with workload types running in them, each domain is assigned a security label that includes the workload types of the domain.

The `SecurityLabelTemplate` (cf Figure 10.4) defines the security labels that can be associated with domains and resources when this policy is active (use the `xm labels type=any` command described in Section 10.2.4 to list all available labels).

The domain labels include Chinese Wall types while resource labels do not include Chinese Wall types. The `SubjectLabels` policy section defines the labels that can be assigned to domains. The VM label `A-Bank.SecurityUnderwriting` in Figure 10.4) associates the domain that carries it with the workload STE type `A-Bank.SecurityUnderwriting` and with the CHWALL types `A-Bank` and `A-Bank.SecurityUnderwriting`. The ezPolicy tool assumes that any department workload will inherit any conflict set that is specified for its organization, i.e., if `B-Bank` is running, not only `A-Bank` but also all its departmental workloads are prevented from running by this first run-time exclusion set. The separation of STE and CHWALL types in the label definition ensures that all departmental workloads are isolated from each other and from their generic organization workloads, while they are sharing CHWALL types to simplify the formulation of run-time exclusion sets.

The `bootstrap` attribute of the `<SubjectLabels>` XML node in our example policy shown in Figure 10.4 names the label `SystemManagement` as the label that Xen will assign to Domain-0 at boot time (if this policy is installed as boot policy). The

```
<SecurityLabelTemplate>
  <SubjectLabels bootstrap="SystemManagement">
  <VirtualMachineLabel>
    <Name>SystemManagement</Name>
    <SimpleTypeEnforcementTypes>
      <Type>SystemManagement</Type>
      <Type>__UNLABELED__</Type>
      <Type>A-Bank</Type>
      <Type>A-Bank.SecurityUnderwriting</Type>
      <Type>A-Bank.MarketAnalysis</Type>
      <Type>B-Bank</Type>
      <Type>AutoCorp</Type>
    </SimpleTypeEnforcementTypes>
    <ChineseWallTypes>
      <Type>SystemManagement</Type>
    </ChineseWallTypes>
  </VirtualMachineLabel>
  <VirtualMachineLabel>
    <Name>__UNLABELED__</Name>
    <SimpleTypeEnforcementTypes>
      <Type>__UNLABELED__</Type>
    </SimpleTypeEnforcementTypes>
    <ChineseWallTypes>
      <Type>__UNLABELED__</Type>
    </ChineseWallTypes>
  </VirtualMachineLabel>
  <VirtualMachineLabel>
    <Name>A-Bank</Name>
    <SimpleTypeEnforcementTypes>
      <Type>A-Bank</Type>
    </SimpleTypeEnforcementTypes>
    <ChineseWallTypes>
      <Type>A-Bank</Type>
    </ChineseWallTypes>
  </VirtualMachineLabel>
  <VirtualMachineLabel>
    <Name>A-Bank.SecurityUnderwriting</Name>
    <SimpleTypeEnforcementTypes>
      <Type>A-Bank.SecurityUnderwriting</Type>
    </SimpleTypeEnforcementTypes>
    <ChineseWallTypes>
      <Type>A-Bank</Type>
      <Type>A-Bank.SecurityUnderwriting</Type>
    </ChineseWallTypes>
  </VirtualMachineLabel>
  <VirtualMachineLabel>
    <Name>A-Bank.MarketAnalysis</Name>
    <SimpleTypeEnforcementTypes>
      <Type>A-Bank.MarketAnalysis</Type>
    </SimpleTypeEnforcementTypes>
    <ChineseWallTypes>
      <Type>A-Bank</Type>
      <Type>A-Bank.MarketAnalysis</Type>
    </ChineseWallTypes>
  </VirtualMachineLabel>
  <VirtualMachineLabel>
    <Name>B-Bank</Name>
    <SimpleTypeEnforcementTypes>
      <Type>B-Bank</Type>
    </SimpleTypeEnforcementTypes>
    <ChineseWallTypes>
      <Type>B-Bank</Type>
    </ChineseWallTypes>
  </VirtualMachineLabel>
  <VirtualMachineLabel>
    <Name>AutoCorp</Name>
    <SimpleTypeEnforcementTypes>
      <Type>AutoCorp</Type>
    </SimpleTypeEnforcementTypes>
    <ChineseWallTypes>
      <Type>AutoCorp</Type>
    </ChineseWallTypes>
  </VirtualMachineLabel>
  </SubjectLabels>
  <ObjectLabels>
  <ResourceLabel>
    <Name>SystemManagement</Name>
    <SimpleTypeEnforcementTypes>
      <Type>SystemManagement</Type>
    </SimpleTypeEnforcementTypes>
  </ResourceLabel>
  <ResourceLabel>
    <Name>__UNLABELED__</Name>
    <SimpleTypeEnforcementTypes>
      <Type>__UNLABELED__</Type>
    </SimpleTypeEnforcementTypes>
  </ResourceLabel>
  <ResourceLabel>
    <Name>A-Bank</Name>
    <SimpleTypeEnforcementTypes>
      <Type>A-Bank</Type>
    </SimpleTypeEnforcementTypes>
  </ResourceLabel>
  <ResourceLabel>
    <Name>A-Bank.SecurityUnderwriting</Name>
    <SimpleTypeEnforcementTypes>
      <Type>A-Bank.SecurityUnderwriting</Type>
    </SimpleTypeEnforcementTypes>
  </ResourceLabel>
  <ResourceLabel>
    <Name>A-Bank.MarketAnalysis</Name>
    <SimpleTypeEnforcementTypes>
      <Type>A-Bank.MarketAnalysis</Type>
    </SimpleTypeEnforcementTypes>
  </ResourceLabel>
  <ResourceLabel>
    <Name>B-Bank</Name>
    <SimpleTypeEnforcementTypes>
      <Type>B-Bank</Type>
    </SimpleTypeEnforcementTypes>
  </ResourceLabel>
  <ResourceLabel>
    <Name>AutoCorp</Name>
    <SimpleTypeEnforcementTypes>
      <Type>AutoCorp</Type>
    </SimpleTypeEnforcementTypes>
  </ResourceLabel>
  </ObjectLabels>
</SecurityLabelTemplate>
</SecurityPolicyDefinition>
```

Figure 10.4: Example XML security policy file – Part II: Label Definition.

label of Domain-0 can be persistently changed at run-time with the `addlabel` command, which adds an overriding option to the grub.conf boot entry (cf Section 10.2.7). All user domains are assigned labels according to their domain configuration (see Section 10.2.4 for examples of how to label domains).

The `ObjectLabels` depicted in Figure 10.4 can be assigned to resources when this policy is active.

In general, user domains should be assigned labels that have only a single SimpleTypeEnforcement workload type. This way, workloads remain confined even if user domains become rogue. Any domain that is assigned a label with multiple STE types must be trusted to keep information belonging to the different STE types separate (confined). For example, Domain-0 is assigned the bootstrap label `SystemManagement`, which includes all existing STE types. Therefore, Domain-0 must take care not to enable unauthorized information flow (eg. through block devices or virtual networking) between domains or resources that are assigned different STE types.

Security administrators simply use the name of a label (specified in the `<Name>` field) to associate a label with a domain (cf. Section 10.2.4). The types inside the label are used by the Xen access control enforcement. While the name can be arbitrarily chosen (as long as it is unique), it is advisable to choose the label name in accordance to the security types included. Similarly, the STE and CHWALL types should be named according to the workloads they represent. While the XML representation of the label in the above example seems unnecessary flexible, labels in general must be able to include multiple types.

We assume in the following example, that `A-Bank.SecurityUnderwriting` and `A-Bank.MarketAnalysis` workloads use virtual disks that are provided by a virtual I/O domain hosting a physical storage device and carrying the following label:

```
<VirtualMachineLabel>
  <Name>VIOServer</Name>
  <SimpleTypeEnforcementTypes>
      <Type>A-Bank</Type>
      <Type>A-Bank.SecurityUnderwriting</Type>
      <Type>A-Bank.MarketAnalysis</Type>
      <Type>VIOServer</Type>
  </SimpleTypeEnforcementTypes>
  <ChineseWallTypes>
      <Type>VIOServer</Type>
  </ChineseWallTypes>
</VirtualMachineLabel>
```

This Virtual I/O domain (VIO) exports its virtualized disks by communicating to all domains labeled with the `A-Bank.SecurityUnderwriting`, the `A-Bank`, or the `A-Bank.MarketAnalysis` label. This requires the VIO domain to carry those STE types. In addition, this label includes a new `VIOServer` type that can be used to restrict direct access to the physical storage resource to the VIODomain.

In this example, the confinement of these A-Bank workloads depends on the VIO domain that must keep the data of those different workloads separate. The virtual disks are labeled as well to keep track of their assignments to workload types (see Section 10.2.5 for labeling resources) and enforcement functions inside the VIO domain must ensure that the labels of the domain mounting a virtual disk and the virtual disk label share a common STE type. The VIO label carrying its own VIOServer CHWALL type introduces the flexibility to permit the trusted VIO server to run together with `A-Bank.SecurityUnderwriting` or `A-Bank.MarketAnalysis` workloads.

Alternatively, a system that has two hard-drives does not need a VIO domain but can directly assign one hardware storage device to each of the workloads if the platform offers an IO-MMU, cf Section 9.2. Sharing hardware through virtualized devices is a trade-off between the amount of trusted code (size of the trusted computing base) and the amount of acceptable over-provisioning. This holds both for peripherals and for system platforms.

10.3.5 Managing sHype/Xen Security Policies at Run-time

Removing the sHype/Xen Security Policy

When resetting the policy, no labeled domains can be running. Please stop or shutdown all running labeled domains. Then you can reset the policy to the default policy using the `resetpolicy` command:

```
# xm getpolicy
Supported security subsystems   : ACM
Policy name           : mytest
Policy type           : ACM
Version of XML policy : 1.0
Policy configuration  : loaded, activated for boot

# xm resetpolicy
Successfully reset the system's policy.

# xm getpolicy
Supported security subsystems   : ACM
Policy name           : DEFAULT
Policy type           : ACM
Version of XML policy : 1.0
Policy configuration  : loaded

# xm resources
file:/home/xen/dom_fc5/fedora.fc5.swap
      type: INV_ACM
    policy: mytest
     label: A-Bank
file:/home/xen/dom_fc5/fedora.fc5.img
      type: INV_ACM
    policy: mytest
     label: A-Bank
```

As the `xm resources` output shows, all resource labels have invalidated type information but their semantics remain associated with the resources so that they can later on either be relabeled with semantically equivalent labels or sanitized and reused (storage resources).

At this point, the system is in the same initial state as after configuring XSM and sHype/ACM and rebooting the system without a specific policy. No user domains can run.

Changing to a Different sHype/Xen Security Policy

The easiest way to change to a different, unrelated policy is to reset the system policy and then set the new policy. Please consider that the existing domain and resource labels become invalid at this point. Please refer to the next section for an example of how to seamlessly update an active policy at run-time without invalidating labels.

```
# xm resetpolicy
Successfully reset the system's policy.

# xm setpolicy ACM example.test
Successfully set the new policy.
Supported security subsystems   : ACM
Policy name           : example.test
Policy type           : ACM
Version of XML policy : 1.0
Policy configuration  : loaded, activated for boot

# xm labels
CocaCola
PepsiCo
SystemManagement
VIO
# xm list --label
Name      ID   Mem VCPUs      State   Time(s) Label
Domain-0   0   873     1     r-----     56.3  ACM:example.test:SystemManagement

# xm resetpolicy
Successfully reset the system's policy.

# xm getpolicy
Supported security subsystems   : ACM
Policy name           : DEFAULT
Policy type           : ACM
Version of XML policy : 1.0
Policy configuration  : loaded

# xm list --label
Name      ID   Mem VCPUs      State   Time(s) Label
Domain-0   0   873     1     r-----     57.2  ACM:DEFAULT:SystemManagement

# xm setpolicy ACM mytest
Successfully set the new policy.
Supported security subsystems   : ACM
Policy name           : mytest
Policy type           : ACM
Version of XML policy : 1.0
Policy configuration  : loaded, activated for boot

# xm labels
A-Bank
A-Bank.MarketAnalysis
A-Bank.SecurityUnderwriting
AutoCorp
B-Bank
SystemManagement
__UNLABELED__

# xm list --label
Name      ID   Mem VCPUs      State   Time(s) Label
Domain-0   0   873     1     r-----     58.0  ACM:mytest:SystemManagement
```

The described way of changing policies by resetting the existing policy is useful for testing different policies. For real deployment environments, a policy update as described in the following section is more appropriate and can be applied seamlessly at run-time while user domains are running.

Update an sHype/Xen Security Policy at Run-time

Once an ACM security policy is activated (loaded into the Xen hypervisor), the policy may be updated at run-time without the need to re-boot the system. The XML update-policy contains several additional information fields that are required to safely link the new policy contents to the old policy and ensure a consistent transformation of the system security state from the old to the new policy. Those additional fields are required for policies that are updating an existing policy at run-time.

The major benefit of policy updates is the ability to add, delete, or rename workload types, labels, and conflict sets (run-time exclusion rules) to accommodate changes in the managed virtual environment without the need to reboot the Xen system. When a new policy renames labels of the current policy, the labels attached to resources and domains are automatically updated during a successful policy update.

We have manually crafted an update policy for the `mytest` security policy and stored it in the file mytest_update-security_policy.xml in the policies directory. We will discuss this policy in detail before using it to update a running sHype/Xen system. The following figures contain the whole contents of the update policy file.

Figure 10.5 shows the policy header of an update-policy and the new `FromPolicy` XML node. For the policy update to succeed, the policy name and the policy version fields of the `FromPolicy` XML node must exactly match those of the currently enforced policy. This ensures a controlled update path of the policy.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Auto-generated by ezPolicy        -->
<SecurityPolicyDefinition xmlns="http://www.ibm.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ibm.com ../../security_policy.xsd ">
    <PolicyHeader>
        <PolicyName>mytest</PolicyName>
        <Date>Tue Nov 27 21:53:45 2007</Date>
        <Version>1.1</Version>
        <FromPolicy>
            <PolicyName>mytest</PolicyName>
            <Version>1.0</Version>
        </FromPolicy>
    </PolicyHeader>
```

Figure 10.5: XML security policy update – Part I: Updated Policy Header.

The version number of the new policy, which is shown in the node following the `Date` node, must be a logical increment to the current policy's version. Therefore at least the minor number of the policy version must be incremented. This ensures that a policy update is applied only to exactly the policy for which this update was created and minimizes unforseen side-effects of policy updates.

Types and Conflic Sets The type names and the assignment of types to labels or conflict sets (run-time exclusion rules) can simply be changed consistently throughout the policy. Types, as opposed to labels, are not directly associated or referenced outside the policy so they do not need to carry their history in a "From" field. The figure below shows the update for the types and conflict sets. The `__UNLABELED__` type is removed to disable support for running unlabeled domains. Additionally, we have renamed the two `A-Bank` department types with abbreviated names `A-Bank.SU` and `A-Bank.MA`. You can also see how those type names are consistently changed within the conflict set definition.

```
<SimpleTypeEnforcement>
    <SimpleTypeEnforcementTypes>
        <Type>SystemManagement</Type>
        <Type>A-Bank</Type>
        <Type>A-Bank.SU</Type>
        <Type>A-Bank.MA</Type>
        <Type>B-Bank</Type>
        <Type>AutoCorp</Type>
    </SimpleTypeEnforcementTypes>
</SimpleTypeEnforcement>

<ChineseWall priority="PrimaryPolicyComponent">
    <ChineseWallTypes>
        <Type>SystemManagement</Type>
        <Type>A-Bank</Type>
        <Type>A-Bank.SU</Type>
        <Type>A-Bank.MA</Type>
        <Type>B-Bank</Type>
        <Type>AutoCorp</Type>
    </ChineseWallTypes>

    <ConflictSets>
        <Conflict name="RER">
            <Type>A-Bank</Type>
            <Type>B-Bank</Type>
        </Conflict>
        <Conflict name="RER">
            <Type>A-Bank.MA</Type>
            <Type>A-Bank.SU</Type>
        </Conflict>
   </ConflictSets>
</ChineseWall>
```

Figure 10.6: XML security policy update – Part II: Updated Types and Conflict Sets.

In the same way, new types can be introduced and new conflict sets can be defined by simply adding the types or conflict sets to the update policy.

Labels Virtual machine and resource labels of an existing policy can be deleted through a policy update simply by omitting them in the update-policy. However, if a currently running virtual machine or a currently used resource is labeled with a label not stated in the update-policy, then the policy update is rejected. This ensures that a policy update leaves the system in a consistent security state.

A policy update also enables the renaming of virtual machine and resource labels. Linking the old label name with the new label name is achieved through the `from` attribute in the `VirtualMachineLabel` or `ResourceLabel` nodes in the update-policy. Figure 10.7 shown how subject and resource labels are updated from their old name `A-Bank.SecurityUnterwriting` to their new name `A-Bank.SU` using the `from` attribute.

```
<SecurityLabelTemplate>
  <SubjectLabels bootstrap="SystemManagement">
  <VirtualMachineLabel>
    <Name>SystemManagement</Name>
    <SimpleTypeEnforcementTypes>
      <Type>SystemManagement</Type>
      <Type>A-Bank</Type>
      <Type>A-Bank.SU</Type>
      <Type>A-Bank.MA</Type>
      <Type>B-Bank</Type>
      <Type>AutoCorp</Type>
    </SimpleTypeEnforcementTypes>
    <ChineseWallTypes>
      <Type>SystemManagement</Type>
    </ChineseWallTypes>
  </VirtualMachineLabel>
  <VirtualMachineLabel>
    <Name>A-Bank-WL</Name>
    <SimpleTypeEnforcementTypes>
      <Type>SystemManagement</Type>
      <Type>A-Bank</Type>
      <Type>A-Bank.SU</Type>
      <Type>A-Bank.MA</Type>
    </SimpleTypeEnforcementTypes>
    <ChineseWallTypes>
      <Type>SystemManagement</Type>
    </ChineseWallTypes>
  </VirtualMachineLabel>
  <VirtualMachineLabel>
    <Name>A-Bank</Name>
    <SimpleTypeEnforcementTypes>
      <Type>A-Bank</Type>
    </SimpleTypeEnforcementTypes>
    <ChineseWallTypes>
      <Type>A-Bank</Type>
    </ChineseWallTypes>
  </VirtualMachineLabel>
  <VirtualMachineLabel>
    <Name from="A-Bank.SecurityUnderwriting">
            A-Bank.SU</Name>
    <SimpleTypeEnforcementTypes>
      <Type>A-Bank.SU</Type>
    </SimpleTypeEnforcementTypes>
    <ChineseWallTypes>
      <Type>A-Bank</Type>
      <Type>A-Bank.SU</Type>
    </ChineseWallTypes>
   </VirtualMachineLabel>
  <VirtualMachineLabel>
    <Name from="A-Bank.MarketAnalysis">
            A-Bank.MA</Name>
    <SimpleTypeEnforcementTypes>
      <Type>A-Bank.MA</Type>
    </SimpleTypeEnforcementTypes>
    <ChineseWallTypes>
      <Type>A-Bank</Type>
      <Type>A-Bank.MA</Type>
    </ChineseWallTypes>
  </VirtualMachineLabel>
  <VirtualMachineLabel>
    <Name>B-Bank</Name>
    <SimpleTypeEnforcementTypes>
      <Type>B-Bank</Type>
    </SimpleTypeEnforcementTypes>
    <ChineseWallTypes>
      <Type>B-Bank</Type>
    </ChineseWallTypes>
  </VirtualMachineLabel>
  <VirtualMachineLabel>
    <Name>AutoCorp</Name>
    <SimpleTypeEnforcementTypes>
      <Type>AutoCorp</Type>
    </SimpleTypeEnforcementTypes>
    <ChineseWallTypes>
      <Type>AutoCorp</Type>
    </ChineseWallTypes>
  </VirtualMachineLabel>
</SubjectLabels>

<ObjectLabels>
  <ResourceLabel>
    <Name>SystemManagement</Name>
    <SimpleTypeEnforcementTypes>
      <Type>SystemManagement</Type>
    </SimpleTypeEnforcementTypes>
  </ResourceLabel>
  <ResourceLabel>
    <Name>A-Bank</Name>
    <SimpleTypeEnforcementTypes>
      <Type>A-Bank</Type>
    </SimpleTypeEnforcementTypes>
  </ResourceLabel>
  <ResourceLabel>
    <Name from="A-Bank.SecurityUnderwriting">
           A-Bank.SU</Name>
    <SimpleTypeEnforcementTypes>
      <Type>A-Bank.SU</Type>
    </SimpleTypeEnforcementTypes>
  </ResourceLabel>
  <ResourceLabel>
    <Name from="A-Bank.MarketAnalysis">
           A-Bank.MA</Name>
    <SimpleTypeEnforcementTypes>
      <Type>A-Bank.MA</Type>
    </SimpleTypeEnforcementTypes>
  </ResourceLabel>
  <ResourceLabel>
    <Name>B-Bank</Name>
    <SimpleTypeEnforcementTypes>
      <Type>B-Bank</Type>
    </SimpleTypeEnforcementTypes>
  </ResourceLabel>
  <ResourceLabel>
    <Name>AutoCorp</Name>
    <SimpleTypeEnforcementTypes>
      <Type>AutoCorp</Type>
    </SimpleTypeEnforcementTypes>
  </ResourceLabel>
  </ObjectLabels>
</SecurityLabelTemplate>
</SecurityPolicyDefinition>
```

Figure 10.7: XML security policy update – Part III: Updated Label Definition.

The updated label definition also includes a new label `A-Bank-WL` that includes all STE types related to A-Bank. Its CHWALL type is `SystemManagement`. This indicates that this label is designed as Domain-0 label. A Xen system can be restricted to only run A-Bank related workloads by relabeling Domain-0 with the `A-Bank-WL` label.

We assume that the update-policy shown in Figures 10.5, 10.6 and 10.7 is stored in the

XML file mytest_update-security_policy.xml located in the ACM policy directory. See Section 10.3.1 for information about policy names and locations.

The following `xm setpolicy` command updates the active ACM security policy at run-time.

```
# xm list --label
Name       ID   Mem VCPUs    State  Time(s) Label
domain1     2   128     1   -b----      0.6  ACM:mytest:A-Bank
domain4     3   164     1   -b----      0.3  ACM:mytest:A-Bank.SecurityUnderwriting
Domain-0    0   711     1   r-----     71.8  ACM:mytest:SystemManagement

# xm resources
file:/home/xen/dom_fc5/fedora.fc5.swap
      type: ACM
    policy: mytest
    label:  A-Bank
file:/home/xen/dom_fc5/fedora.fc5.img
      type: ACM
    policy: mytest
    label:  A-Bank

# xm setpolicy ACM mytest_update
Successfully set the new policy.
Supported security subsystems   : ACM
Policy name           : mytest
Policy type           : ACM
Version of XML policy : 1.1
Policy configuration  : loaded, activated for boot

# xm list --label
Name       ID   Mem VCPUs      State   Time(s) Label
domain1     2   128     1     -b----      0.7  ACM:mytest:A-Bank
domain4     3   164     1     -b----      0.3  ACM:mytest:A-Bank.SU
Domain-0    0   711     1     r-----     72.8  ACM:mytest:SystemManagement

# xm labels
A-Bank
A-Bank-WL
A-Bank.MA
A-Bank.SU
AutoCorp
B-Bank

# xm resources
file:/home/xen/dom_fc5/fedora.fc5.swap
      type: ACM
    policy: mytest
     label: A-Bank
file:/home/xen/dom_fc5/fedora.fc5.img
      type: ACM
    policy: mytest
     label: A-Bank
```

After successful completion of this command, `xm list --label` shows that the labels of running domains changed to their new names. `xm labels` shows that new labels `A-Bank.SU` and `A-Bank.AM` are now available in the policy. The resource labels remain valid after the successful update as `xm resources` confirms.

The `setpolicy` command fails if the new policy is inconsistent with the current one or the policy is inconsistent internally (e.g., types are renamed in the type definition but not in the label definition part of the policy). In this case, the old policy remains active.

After relabeling Domain-0 with the new `A-Bank-WL` label, we can no longer run domains labeled `B-Bank` or `AutoCorp` since their STE types are not a subset of the new Domain-0 label.

```
# xm addlabel A-Bank-WL mgt Domain-0
Successfully set the label of domain 'Domain-0' to 'A-Bank-WL'.

# xm list --label
Name       ID   Mem VCPUs      State   Time(s) Label
domain1     2   128     1     -b----       0.8  ACM:mytest:A-Bank
Domain-0    0   711     1     r-----      74.5  ACM:mytest:A-Bank-WL
domain4     3   164     1     -b----       0.3  ACM:mytest:A-Bank.SU

# xm getlabel dom domain3.xm
policytype=ACM,policy=mytest,label=AutoCorp

# xm create domain3.xm
Using config file "./domain3.xm".
Error: VM is not authorized to run.

# xm addlabel SystemManagement mgt Domain-0
Successfully set the label of domain 'Domain-0' to 'SystemManagement'.

# xm list --label
Name       ID   Mem VCPUs      State   Time(s) Label
domain1     2   128     1     -b----       0.8  ACM:mytest:A-Bank
domain4     3   164     1     -b----       0.3  ACM:mytest:A-Bank.SU
Domain-0    0   709     1     r-----      76.4  ACM:mytest:SystemManagement

# xm create domain3.xm
Using config file "./domain3.xm".
Started domain domain3

# xm list --label
Name       ID   Mem VCPUs      State   Time(s) Label
domain1     2   128     1     -b----       0.8  ACM:mytest:A-Bank
domain4     3   164     1     -b----       0.3  ACM:mytest:A-Bank.SU
domain3     4   164     1     -b----       0.3  ACM:mytest:AutoCorp
Domain-0    0   547     1     r-----      77.5  ACM:mytest:SystemManagement
```

In the same manner, you can add new labels to support new workloads and add, delete, or rename workload types (STE and/or CHWALL types) simply by changing the composition of labels. Another use case is to add new workload types to the current Domain-0 label to enable them to run. Conflict sets (run-time exclusion rules) can be simply omitted or added. The policy and label changes become active at once and new workloads can be run in protected mode without rebooting the Xen system.

In all these cases, if any running user domain would–under the new policy–not be allowed to run or would not be allowed to access any of the resources it currently uses, then the policy update is rejected. In this case, you can stop domains that conflict with the new policy and update the policy afterwards. The old policy remains active until a policy update succeeds or Xen is re-booted into a new policy.

10.3.6 Tools For Creating sHype/Xen Security Policies

To create a security policy for Xen, you can use one of the following tools:

- `ezPolicy` GUI tool – start writing policies
- `xensec_gen` tool – refine policies created with `ezPolicy`
- text or XML editor

We use the `ezPolicy` tool in Section 10.2.2 to quickly create a workload protection policy. If desired, the resulting XML policy file can be loaded into the `xensec_gen` tool to refine it. It can also be directly edited using an XML editor. Any XML policy file is verified against the security policy schema when it is translated (see Subsection 10.2.3).

10.4 Current Limitations

The sHype/ACM configuration for Xen is work in progress. There is ongoing work for protecting virtualized resources and planned and ongoing work for protecting access to remote resources and domains. The following sections describe limitations of some of the areas into which access control is being extended.

10.4.1 Network Traffic

Local and remote network traffic is currently not controlled. Solutions to add sHype/ACM policy enforcement to the virtual network exist but need to be discussed before they can become part of Xen. Subjecting external network traffic to the ACM security policy is work in progress. Manually setting up filters in domain 0 is required for now but does not scale well.

10.4.2 Resource Access and Usage Control

Enforcing the security policy across multiple hypervisor systems and on access to remote shared resources is work in progress. Extending access control to new types of resources is ongoing work (e.g. network storage).

On a single Xen system, information about the association of resources and security labels is stored in `/var/lib/xend/security/policies/resource_labels`. This file relates a full resource path with a security label. This association is weak and will break if resources are moved or renamed without adapting the label file. Improving the protection of label-resource relationships is ongoing work.

Controlling resource usage and enforcing resource limits in general is ongoing work in the Xen community.

10.4.3 Domain Migration

Labels on domains are enforced during domain migration and the destination hypervisor will ensure that the domain label is valid and the domain is permitted to run (considering the Chinese Wall policy rules) before it accepts the migration. However, the network between the source and destination hypervisor as well as both hypervisors must be trusted. Architectures and prototypes exist that both protect the network connection and ensure that the hypervisors enforce access control consistently but patches are not yet available for the main stream.

10.4.4 Covert Channels

The sHype access control aims at system independent security policies. It builds on top of the core hypervisor isolation. Any covert channels that exist in the core hypervisor or in the hardware (e.g., shared processor cache) will be inherited. If those covert channels are not the result of trade-offs between security and other system properties, then they are most effectively minimized or eliminated where they are caused. sHype offers however some means to mitigate their impact, e.g., run-time exclusion rules (cf Section 10.2.2) or limiting the system authorization (cf Section 10.2.7).

Part III

Reference

Chapter 11

Build and Boot Options

This chapter describes the build- and boot-time options which may be used to tailor your Xen system.

11.1 Top-level Configuration Options

Top-level configuration is achieved by editing one of two files: `Config.mk` and `Makefile`.

The former allows the overall build target architecture to be specified. You will typically not need to modify this unless you are cross-compiling. Additional configuration options are documented in the `Config.mk` file.

The top-level `Makefile` is chiefly used to customize the set of kernels built. Look for the line:

```
KERNELS ?= linux-2.6-xen0 linux-2.6-xenU
```

Allowable options here are any kernels which have a corresponding build configuration file in the `buildconfigs/` directory.

11.2 Xen Build Options

Xen provides a number of build-time options which should be set as environment variables or passed on make's command-line.

verbose=y Enable debugging messages when Xen detects an unexpected condition. Also enables console output from all domains.

debug=y Enable debug assertions. Implies **verbose=y**. (Primarily useful for tracing bugs in Xen).

debugger=y Enable the in-Xen debugger. This can be used to debug Xen, guest OSes, and applications.

perfc=y Enable performance counters for significant events within Xen. The counts can be reset or displayed on Xen's console via console control keys.

11.3 Xen Boot Options

These options are used to configure Xen's behaviour at runtime. They should be appended to Xen's command line, either manually or by editing `grub.conf`.

noreboot Don't reboot the machine automatically on errors. This is useful to catch debug output if you aren't catching console messages via the serial line.

nosmp Disable SMP support. This option is implied by 'ignorebiostables'.

watchdog Enable NMI watchdog which can report certain failures.

noirqbalance Disable software IRQ balancing and affinity. This can be used on systems such as Dell 1850/2850 that have workarounds in hardware for IRQ-routing issues.

badpage=<page number>,<page number>, ... Specify a list of pages not to be allocated for use because they contain bad bytes. For example, if your memory tester says that byte 0x12345678 is bad, you would place 'badpage=0x12345' on Xen's command line.

serial_tx_buffer=<size> Size of serial transmit buffers. Default is 16kB.

com1=<baud>,DPS,<io_base>,<irq> com2=<baud>,DPS,<io_base>,<irq>

Xen supports up to two 16550-compatible serial ports. For example: 'com1=9600, 8n1, 0x408, 5' maps COM1 to a 9600-baud port, 8 data bits, no parity, 1 stop bit, I/O port base 0x408, IRQ 5. If some configuration options are standard (e.g., I/O base and IRQ), then only a prefix of the full configuration string need be specified. If the baud rate is pre-configured (e.g., by the bootloader) then you can specify 'auto' in place of a numeric baud rate.

console=<specifier list> Specify the destination for Xen console I/O. This is a comma-separated list of, for example:

vga Use VGA console (until domain 0 boots, unless **vga=...keep** is specified).

com1 Use serial port com1.

com2H Use serial port com2. Transmitted chars will have the MSB set. Received chars must have MSB set.

com2L Use serial port com2. Transmitted chars will have the MSB cleared. Received chars must have MSB cleared.

The latter two examples allow a single port to be shared by two subsystems (e.g. console and debugger). Sharing is controlled by MSB of each transmitted/received character. [NB. Default for this option is ‘com1,vga’]

vga=<mode>(,keep) The mode is one of the following options:

> **ask** Display a vga menu allowing manual selection of video mode.
>
> **current** Use existing vga mode without modification.
>
> **text-<mode>** Select text-mode resolution, where mode is one of 80x25, 80x28, 80x30, 80x34, 80x43, 80x50, 80x60.
>
> **gfx-<mode>** Select VESA graphics mode <width>x<height>x<depth> (e.g., ‘vga=gfx-1024x768x32’).
>
> **mode-<mode>** Specify a mode number as discovered by ‘vga ask’. Note that the numbers are displayed in hex and hence must be prefixed by ‘0x’ here (e.g., ‘vga=mode-0x0335’).

The mode may optionally be followed by ‘**,keep**’ to cause Xen to keep writing to the VGA console after domain 0 starts booting (e.g., ‘vga=text-80x50,keep’).

no-real-mode (x86 only) Do not execute real-mode bootstrap code when booting Xen. This option should not be used except for debugging. It will effectively disable the **vga** option, which relies on real mode to set the video mode.

edid=no,force (x86 only) Either force retrieval of monitor EDID information via VESA DDC, or disable it (edid=no). This option should not normally be required except for debugging purposes.

edd=off,on,skipmbr (x86 only) Control retrieval of Extended Disc Data (EDD) from the BIOS during boot.

console_to_ring Place guest console output into the hypervisor console ring buffer. This is disabled by default. When enabled, both hypervisor output and guest console output is available from the ring buffer. This can be useful for logging and/or remote presentation of console data.

sync_console Force synchronous console output. This is useful if you system fails unexpectedly before it has sent all available output to the console. In most cases Xen will automatically enter synchronous mode when an exceptional event occurs, but this option provides a manual fallback.

conswitch=<switch-char><auto-switch-char> Specify how to switch serial-console input between Xen and DOM0. The required sequence is CTRL-<switch-char> pressed three times. Specifying the backtick character disables switching. The <auto-switch-char> specifies whether Xen should auto-switch input to DOM0 when it boots — if it is ‘x’ then auto-switching is disabled. Any other value, or omitting the character, enables auto-switching. [NB. Default switch-char is ‘a’.]

loglvl=<level> / <level> Specify logging level. Messages of the specified severity level (and higher) will be printed to the Xen console. Valid levels are ‘none’, ‘error', 'warning', 'info', 'debug', and 'all'. The second level specifier is optional: it is used to specify message severities which are to be rate limited. Default is 'loglvl=warning'.

guest_loglvl=<level> / <level> As for loglvl, but applies to messages relating to guests. Default is 'guest_loglvl=none/warning'.

console_timestamps Adds a timestamp prefix to each line of Xen console output.

nmi=xxx Specify what to do with an NMI parity or I/O error.
'nmi=fatal': Xen prints a diagnostic and then hangs.
'nmi=dom0': Inform DOM0 of the NMI.
'nmi=ignore': Ignore the NMI.

mem=xxx Set the physical RAM address limit. Any RAM appearing beyond this physical address in the memory map will be ignored. This parameter may be specified with a B, K, M or G suffix, representing bytes, kilobytes, megabytes and gigabytes respectively. The default unit, if no suffix is specified, is kilobytes.

dom0_mem=<specifier list> Set the amount of memory to be allocated to domain 0. This is a comma-separated list containing the following optional components:

min:<min_amt> Minimum amount to allocate to domain 0

max:<min_amt> Maximum amount to allocate to domain 0

<amt> Precise amount to allocate to domain 0

Each numeric parameter may be specified with a B, K, M or G suffix, representing bytes, kilobytes, megabytes and gigabytes respectively; if no suffix is specified, the parameter defaults to kilobytes. Negative values are subtracted from total available memory. If <amt> is not specified, it defaults to all available memory less a small amount (clamped to 128MB) for uses such as DMA buffers.

dom0_vcpus_pin Pins domain 0 VCPUs on their respective physical CPUS (default=false).

tbuf_size=xxx Set the size of the per-cpu trace buffers, in pages (default 0).

sched=xxx Select the CPU scheduler Xen should use. The current possibilities are 'credit' (default), and 'sedf'.

apic_verbosity=debug,verbose Print more detailed information about local APIC and IOAPIC configuration.

lapic Force use of local APIC even when left disabled by uniprocessor BIOS.

nolapic Ignore local APIC in a uniprocessor system, even if enabled by the BIOS.

apic=bigsmp,default,es7000,summit Specify NUMA platform. This can usually be probed automatically.

dma_bits=xxx Specify width of DMA addresses in bits. This is used in NUMA systems to prevent this special DMA memory from being exhausted in one node when remote nodes have available memory.

vcpu_migration_delay=<minimum_time> Set minimum time of vcpu migration in microseconds (default 0). This parameter avoids agressive vcpu migration. For example, the linux kernel uses 0.5ms by default.

In addition, the following options may be specified on the Xen command line. Since domain 0 shares responsibility for booting the platform, Xen will automatically propagate these options to its command line. These options are taken from Linux's command-line syntax with unchanged semantics.

acpi=off,force,strict,ht,noirq,... Modify how Xen (and domain 0) parses the BIOS ACPI tables.

acpi_skip_timer_override Instruct Xen (and domain 0) to ignore timer-interrupt override instructions specified by the BIOS ACPI tables.

noapic Instruct Xen (and domain 0) to ignore any IOAPICs that are present in the system, and instead continue to use the legacy PIC.

11.4 XenLinux Boot Options

In addition to the standard Linux kernel boot options, we support:

xencons=xxx Specify the device node to which the Xen virtual console driver is attached. The following options are supported:

'xencons=off': disable virtual console
'xencons=tty': attach console to /dev/tty1 (tty0 at boot-time)
'xencons=ttyS': attach console to /dev/ttyS0
'xencons=xvc': attach console to /dev/xvc0

The default is ttyS for dom0 and xvc for all other domains.

Chapter 12

Further Support

If you have questions that are not answered by this manual, the sources of information listed below may be of interest to you. Note that bug reports, suggestions and contributions related to the software (or the documentation) should be sent to the Xen developers' mailing list (address below).

12.1 Other Documentation

For developers interested in porting operating systems to Xen, the *Xen Interface Manual* is distributed in the `docs/` directory of the Xen source distribution.

12.2 Online References

The official Xen web site can be found at:

`http://www.xen.org`

This contains links to the latest versions of all online documentation, including the latest version of the FAQ.

Information regarding Xen is also available at the Xen Wiki at

`http://wiki.xensource.com/xenwiki/`

The Xen project uses Bugzilla as its bug tracking system. You'll find the Xen Bugzilla at http://bugzilla.xensource.com/bugzilla/.

12.3 Mailing Lists

There are several mailing lists that are used to discuss Xen related topics. The most widely relevant are listed below. An official page of mailing lists and subscription information can be found at

`http://lists.xensource.com/`

xen-devel@lists.xensource.com Used for development discussions and bug reports. Subscribe at:
`http://lists.xensource.com/xen-devel`

xen-users@lists.xensource.com Used for installation and usage discussions and requests for help. Subscribe at:
`http://lists.xensource.com/xen-users`

xen-announce@lists.xensource.com Used for announcements only. Subscribe at:
`http://lists.xensource.com/xen-announce`

xen-changelog@lists.xensource.com Changelog feed from the unstable and 3.x trees - developer oriented. Subscribe at:
`http://lists.xensource.com/xen-changelog`

Appendix A

Unmodified (HVM) guest domains in Xen with Hardware support for Virtualization

Xen supports guest domains running unmodified guest operating systems using virtualization extensions available on recent processors. Currently processors featuring the Intel Virtualization Extension (Intel-VT) or the AMD extension (AMD-V) are supported. The technology covering both implementations is called HVM (for Hardware Virtual Machine) in Xen. More information about the virtualization extensions are available on the respective websites: `http://www.intel.com/technology/computing/vptech`

`http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/24593.pdf`

A.1 Building Xen with HVM support

The following packages need to be installed in order to build Xen with HVM support. Some Linux distributions do not provide these packages by default.

Package	Description
dev86	The dev86 package provides an assembler and linker for real mode 80x86 instructions. You need to have this package installed in order to build the BIOS code which runs in (virtual) real mode. If the dev86 package is not available on the x86_64 distribution, you can install the i386 version of it. The dev86 rpm package for various distributions can be found at `http://www.rpmfind.net/linux/rpm2html/search.php?query=dev86&submit=Search`
SDL-devel, SDL	Simple DirectMedia Layer (SDL) is another way of virtualizing the unmodified guest console. It provides an X window for the guest console. If the SDL and SDL-devel packages are not installed by default on the build system, they can be obtained from `http://www.rpmfind.net/linux/rpm2html/search.php?query=SDL&submit=Search` `http://www.rpmfind.net/linux/rpm2html/search.php?query=SDL-devel&submit=Search`

A.2 Configuration file for unmodified HVM guests

The Xen installation includes a sample configuration file, `/etc/xen/xmexample.hvm`. There are comments describing all the options. In addition to the common options that are the same as those for paravirtualized guest configurations, HVM guest configurations have the following settings:

Parameter	Description
kernel	The HVM firmware loader, `/usr/lib/xen/boot/hvmloader`
builder	The domain build function. The HVM domain uses the 'hvm' builder.
acpi	Enable HVM guest ACPI, default=1 (enabled)
apic	Enable HVM guest APIC, default=1 (enabled)
pae	Enable HVM guest PAE, default=1 (enabled)
hap	Enable hardware-assisted paging support, such as AMD-V's nested paging or Intel®VT's extended paging. If available, Xen will use hardware-assisted paging instead of shadow paging for this guest's memory management.
vif	Optionally defines MAC address and/or bridge for the network interfaces. Random MACs are assigned if not given. `type=ioemu` means ioemu is used to virtualize the HVM NIC. If no type is specified, vbd is used, as with paravirtualized guests.

disk	Defines the disk devices you want the domain to have access to, and what you want them accessible as. If using a physical device as the HVM guest's disk, each disk entry is of the form
	`phy:UNAME,ioemu:DEV,MODE,`
	where UNAME is the host device file, DEV is the device name the domain will see, and MODE is r for read-only, w for read-write. ioemu means the disk will use ioemu to virtualize the HVM disk. If not adding ioemu, it uses vbd like paravirtualized guests.
	If using disk image file, its form should be like
	`file:FILEPATH,ioemu:DEV,MODE`
	Optical devices can be emulated by appending cdrom to the device type
	`',hdc:cdrom,r'`
	If using more than one disk, there should be a comma between each disk entry. For example:
	`disk = ['file:/var/images/image1.img,ioemu:hda,w',`
	`'phy:hda1,hdb1,w', 'file:/var/images/install1.iso,hdc:cdrom,r']`

boot	Boot from floppy (a), hard disk (c) or CD-ROM (d). For example, to boot from CD-ROM and fallback to HD, the entry should be: boot='dc'
device_model	The device emulation tool for HVM guests. This parameter should not be changed.
sdl	Enable SDL library for graphics, default = 0 (disabled)
vnc	Enable VNC library for graphics, default = 1 (enabled)
vncconsole	Enable spawning of the vncviewer (only valid when vnc=1), default = 0 (disabled) If vnc=1 and vncconsole=0, user can use vncviewer to manually connect HVM from remote. For example: `vncviewer domain0_IP_address:HVM_domain_id`
serial	Enable redirection of HVM serial output to pty device

usb	Enable USB support without defining a specific USB device. This option defaults to 0 (disabled) unless the option usbdevice is specified in which case this option then defaults to 1 (enabled).
usbdevice	Enable USB support and also enable support for the given device. Devices that can be specified are `mouse` (a PS/2 style mouse), `tablet` (an absolute pointing device) and `host:id1:id2` (a physical USB device on the host machine whose ids are `id1` and `id2`). The advantage of `tablet` is that Windows guests will automatically recognize and support this device so specifying the config line `usbdevice='tablet'` will create a mouse that works transparently with Windows guests under VNC. Linux doesn't recognize the USB tablet yet so Linux guests under VNC will still need the Summagraphics emulation. Details about mouse emulation are provided in section **A.4.3**.
localtime	Set the real time clock to local time [default=0, that is, set to UTC].
soundhw	Enable sound card support and specify the hardware to emulate. Values can be sb16, es1370 or all. Default is none.
full-screen	Start in full screen.
nographic	Another way to redirect serial output. If enabled, no 'sdl' or 'vnc' can work. Not recommended.

A.3 Creating virtual disks from scratch

A.3.1 Using physical disks

If you are using a physical disk or physical disk partition, you need to install a Linux OS on the disk first. Then the boot loader should be installed in the correct place. For example `dev/sda` for booting from the whole disk, or `/dev/sda1` for booting from partition 1.

A.3.2 Using disk image files

You need to create a large empty disk image file first; then, you need to install a Linux OS onto it. There are two methods you can choose. One is directly installing it using a HVM guest while booting from the OS installation CD-ROM. The other is copying an installed OS into it. The boot loader will also need to be installed.

To create the image file:

The image size should be big enough to accommodate the entire OS. This example assumes the size is 1G (which is probably too small for most OSes).

```
# dd if=/dev/zero of=hd.img bs=1M count=0 seek=1024
```

To directly install Linux OS into an image file using a HVM guest:

Install Xen and create HVM with the original image file with booting from CD-ROM. Then it is just like a normal Linux OS installation. The HVM configuration file should have a stanza for the CD-ROM as well as a boot device specification:

```
disk=['file:/var/images/your-hd.img,hda,w', ',hdc:cdrom,r' ] boot='d'
```

If this method does not succeed, you can choose the following method of copying an installed Linux OS into an image file.

To copy a installed OS into an image file:

Directly installing is an easier way to make partitions and install an OS in a disk image file. But if you want to create a specific OS in your disk image, then you will most likely want to use this method.

1. **Install a normal Linux OS on the host machine**
   You can choose any way to install Linux, such as using yum to install Red Hat Linux or YAST to install Novell SuSE Linux. The rest of this example assumes the Linux OS is installed in `/var/guestos/`.
2. **Make the partition table**
   The image file will be treated as hard disk, so you should make the partition table in the image file. For example:
   ```
   # losetup /dev/loop0 hd.img
   # fdisk -b 512 -C 4096 -H 16 -S 32 /dev/loop0
   press 'n' to add new partition
   press 'p' to choose primary partition
   press '1' to set partition number
   press "Enter" keys to choose default value of "First Cylinder" parameter.
   press "Enter" keys to choose default value of "Last Cylinder" parameter.
   press 'w' to write partition table and exit
   # losetup -d /dev/loop0
   ```
3. **Make the file system and install grub**
   ```
   # ln -s /dev/loop0 /dev/loop
   # losetup /dev/loop0 hd.img
   # losetup -o 16384 /dev/loop1 hd.img
   ```

```
# mkfs.ext3 /dev/loop1
# mount /dev/loop1 /mnt
# mkdir -p /mnt/boot/grub
# cp /boot/grub/stage* /boot/grub/e2fs_stage1_5 /mnt/boot/grub
# umount /mnt
# grub
grub> device (hd0) /dev/loop
grub> root (hd0,0)
grub> setup (hd0)
grub> quit
# rm /dev/loop
# losetup -d /dev/loop0
# losetup -d /dev/loop1
```

The `losetup` option `-o 16384` skips the partition table in the image file. It is the number of sectors times 512. We need `/dev/loop` because grub is expecting a disk device *name*, where *name* represents the entire disk and *name1* represents the first partition.

4. **Copy the OS files to the image**
   If you have Xen installed, you can easily use `lomount` instead of `losetup` and `mount` when coping files to some partitions. `lomount` just needs the partition information.

```
# lomount -t ext3 -diskimage hd.img -partition 1 /mnt/guest
# cp -ax /var/guestos/{root,dev,var,etc,usr,bin,sbin,lib} /mnt/guest
# mkdir /mnt/guest/{proc,sys,home,tmp}
```

5. **Edit the `/etc/fstab` of the guest image**
   The fstab should look like this:

```
# vim /mnt/guest/etc/fstab
/dev/hda1 / ext3 defaults 1 1
none /dev/pts devpts gid=5,mode=620 0 0
none /dev/shm tmpfs defaults 0 0
none /proc proc defaults 0 0
none /sys sysfs efaults 0 0
```

6. **umount the image file**
   `# umount /mnt/guest`

Now, the guest OS image `hd.img` is ready. You can also reference `http://free.oszoo.org` for quickstart images. But make sure to install the boot loader.

A.4 HVM Guests

A.4.1 Editing the Xen HVM config file

Make a copy of the example HVM configuration file `/etc/xen/xmexample.hvm` and edit the line that reads

`disk = [ 'file:/var/images/`*`min-el3-i386.img`*`,hda,w' ]`

replacing *min-el3-i386.img* with the name of the guest OS image file you just made.

A.4.2 Creating HVM guests

Simply follow the usual method of creating the guest, providing the filename of your HVM configuration file:

```
# xend start
# xm create /etc/xen/hvmguest.hvm
```

In the default configuration, VNC is on and SDL is off. Therefore VNC windows will open when HVM guests are created. If you want to use SDL to create HVM guests, set `sdl=1` in your HVM configuration file. You can also turn off VNC by setting `vnc=0`.

A.4.3 Mouse issues, especially under VNC

Mouse handling when using VNC is a little problematic. The problem is that the VNC viewer provides a virtual pointer which is located at an absolute location in the VNC window and only absolute coordinates are provided. The HVM device model converts these absolute mouse coordinates into the relative motion deltas that are expected by the PS/2 mouse driver running in the guest. Unfortunately, it is impossible to keep these generated mouse deltas accurate enough for the guest cursor to exactly match the VNC pointer. This can lead to situations where the guest's cursor is in the center of the screen and there's no way to move that cursor to the left (it can happen that the VNC pointer is at the left edge of the screen and, therefore, there are no longer any left mouse deltas that can be provided by the device model emulation code.)

To deal with these mouse issues there are 4 different mouse emulations available from the HVM device model:

PS/2 mouse over the PS/2 port. This is the default mouse that works perfectly well under SDL. Under VNC the guest cursor will get out of sync with the VNC pointer. When this happens you can re-synchronize the guest cursor to the VNC pointer by holding down the **left-ctl** and **left-alt** keys together. While these keys are down VNC pointer motions will not be reported to the guest so that the

VNC pointer can be moved to a place where it is possible to move the guest cursor again.

Summagraphics mouse over the serial port. The device model also provides emulation for a Summagraphics tablet, an absolute pointer device. This emulation is provided over the second serial port, **/dev/ttyS1** for Linux guests and **COM2** for Windows guests. Unfortunately, neither Linux nor Windows provides default support for the Summagraphics tablet so the guest will have to be manually configured for this mouse.

Linux configuration.

First, configure the GPM service to use the Summagraphics tablet. This can vary between distributions but, typically, all that needs to be done is modify the file `/etc/sysconfig/mouse` to contain the lines:

```
MOUSETYPE="summa"
XMOUSETYPE="SUMMA"
DEVICE=/dev/ttyS1
```

and then restart the GPM daemon.

Next, modify the X11 config `/etc/X11/xorg.conf` to support the Summgraphics tablet by replacing the input device stanza with the following:

```
Section "InputDevice"
    Identifier "Mouse0"
    Driver "summa"
    Option "Device" "/dev/ttyS1"
    Option "InputFashion" "Tablet"
    Option "Mode" "Absolute"
    Option "Name" "EasyPen"
    Option "Compatible" "True"
    Option "Protocol" "Auto"
    Option "SendCoreEvents" "on"
    Option "Vendor" "GENIUS"
EndSection
```

Restart X and the X cursor should now properly track the VNC pointer.

Windows configuration.

Get the file `http://www.cad-plan.de/files/download/tw2k.exe` and execute that file on the guest, answering the questions as follows:

1. When the program asks for **model**, scroll down and select **SummaSketch (MM Compatible)**.
2. When the program asks for **COM Port** specify **com2**.
3. When the programs asks for a **Cursor Type** specify **4 button cursor/puck**.

4. The guest system will then reboot and, when it comes back up, the guest cursor will now properly track the VNC pointer.

PS/2 mouse over USB port. This is just the same PS/2 emulation except it is provided over a USB port. This emulation is enabled by the configuration flag:

```
usbdevice='mouse'
```

USB tablet over USB port. The USB tablet is an absolute pointing device that has the advantage that it is automatically supported under Windows guests, although Linux guests still require some manual configuration. This mouse emulation is enabled by the configuration flag:

```
usbdevice='tablet'
```

Linux configuration.

Unfortunately, there is no GPM support for the USB tablet at this point in time. If you intend to use a GPM pointing device under VNC you should configure the guest for Summagraphics emulation.

Support for X11 is available by following the instructions at `http://stz-softwaretechnik.com/~ke/touchscreen/evtouch.html` with one minor change. The `xorg.conf` given in those instructions uses the wrong values for the X & Y minimums and maximums, use the following config stanza instead:

```
Section "InputDevice"
    Identifier      "Tablet"
    Driver          "evtouch"
    Option          "Device" "/dev/input/event2"
    Option          "DeviceName" "touchscreen"
    Option          "MinX" "0"
    Option          "MinY" "0"
    Option          "MaxX" "32256"
    Option          "MaxY" "32256"
    Option          "ReportingMode" "Raw"
    Option          "Emulate3Buttons"
    Option          "Emulate3Timeout" "50"
    Option          "SendCoreEvents" "On"
EndSection
```

Windows configuration.

Just enabling the USB tablet in the guest's configuration file is sufficient, Windows will automatically recognize and configure device drivers for this pointing device.

A.4.4 USB Support

There is support for an emulated USB mouse, an emulated USB tablet and physical low speed USB devices (support for high speed USB 2.0 devices is still under development).

USB PS/2 style mouse. Details on the USB mouse emulation are given in sections **A.2** and **A.4.3**. Enabling USB PS/2 style mouse emulation is just a matter of adding the line

```
usbdevice='mouse'
```

to the configuration file.

USB tablet. Details on the USB tablet emulation are given in sections **A.2** and **A.4.3**. Enabling USB tablet emulation is just a matter of adding the line

```
usbdevice='tablet'
```

to the configuration file.

USB physical devices. Access to a physical (low speed) USB device is enabled by adding a line of the form

```
usbdevice='host:vid:pid'
```

into the the configuration file.[1] **vid** and **pid** are a product id and vendor id that uniquely identify the USB device. These ids can be identified in two ways:

1. Through the control window. As described in section **A.4.6** the control window is activated by pressing **ctl-alt-2** in the guest VGA window. As long as USB support is enabled in the guest by including the config file line

   ```
   usb=1
   ```

   then executing the command

   ```
   info usbhost
   ```

   in the control window will display a list of all usb devices and their ids. For example, this output:

   ```
   Device 1.3, speed 1.5 Mb/s
     Class 00: USB device 04b3:310b
   ```

   was created from a USB mouse with vendor id **04b3** and product id **310b**. This device could be made available to the HVM guest by including the config file entry

   ```
   usbdevice='host:04be:310b'
   ```

[1] There is an alternate way of specifying a USB device that uses the syntax **host:bus.addr** but this syntax suffers from a major problem that makes it effectively useless. The problem is that the **addr** portion of this address changes every time the USB device is plugged into the system. For this reason this addressing scheme is not recommended and will not be documented further.

It is also possible to enable access to a USB device dynamically through the control window. The control window command

```
usb_add host:vid:pid
```

will also allow access to a USB device with vendor id **vid** and product id **pid**.

2. Through the `/proc` file system. The contents of the pseudo file `/proc/bus/usb/devices` can also be used to identify vendor and product ids. Looking at this file, the line starting with **P:** has a field **Vendor** giving the vendor id and another field **ProdID** giving the product id. The contents of `/proc/bus/usb/devices` for the example mouse is as follows:

```
T:  Bus=01 Lev=01 Prnt=01 Port=01 Cnt=02 Dev#=  3 Spd=1.5 MxCh= 0
D:  Ver= 2.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=04b3 ProdID=310b Rev= 1.60
C:* #Ifs= 1 Cfg#= 1 Atr=a0 MxPwr=100mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=03(HID  ) Sub=01 Prot=02 Driver=(none)
E:  Ad=81(I) Atr=03(Int.) MxPS=   4 Ivl=10ms
```

Note that the **P:** line correctly identifies the vendor id and product id for this mouse as **04b3:310b**.

There is one other issue to be aware of when accessing a physical USB device from the guest. The Dom0 kernel must not have a device driver loaded for the device that the guest wishes to access. This means that the Dom0 kernel must not have that device driver compiled into the kernel or, if using modules, that driver module must not be loaded. Note that this is the device specific USB driver that must not be loaded, either the **UHCI** or **OHCI** USB controller driver must still be loaded.

Going back to the USB mouse as an example, if **lsmod** gives the output:

```
Module                  Size  Used by
usbmouse                4128  0
usbhid                 28996  0
uhci_hcd               35409  0
```

then the USB mouse is being used by the Dom0 kernel and is not available to the guest. Executing the command **rmmod usbhid**[2] will remove the USB mouse driver from the Dom0 kernel and the mouse will now be accessible by the HVM guest.

Be aware the the Linux USB hotplug system will reload the drivers if a USB device is removed and plugged back in. This means that just unloading the driver module might not be sufficient if the USB device is removed and added back. A more reliable technique is to first **rmmod** the driver and then rename

[2]Turns out the **usbhid** driver is the significant one for the USB mouse, the presence or absence of the module **usbmouse** has no effect on whether or not the guest can see a USB mouse.

the driver file in the `/lib/modules` directory, just to make sure it doesn't get reloaded.

A.4.5 Destroy HVM guests

HVM guests can be destroyed in the same way as can paravirtualized guests. We recommend that you shut-down the guest using the guest OS' provided method, for Linux, type the command

`poweroff`

in the HVM guest's console, for Windows use Start -¿ Shutdown first to prevent data loss. Depending on the configuration the guest will be automatically destroyed, otherwise execute the command

`xm destroy` *`vmx_guest_id`*

at the Domain0 console.

A.4.6 HVM window (X or VNC) Hot Key

If you are running in the X environment after creating a HVM guest, an X window is created. There are several hot keys for control of the HVM guest that can be used in the window.

Ctrl+Alt+2 switches from guest VGA window to the control window. Typing `help` shows the control commands help. For example, 'q' is the command to destroy the HVM guest.
Ctrl+Alt+1 switches back to HVM guest's VGA.
Ctrl+Alt+3 switches to serial port output. It captures serial output from the HVM guest. It works only if the HVM guest was configured to use the serial port.

Appendix B

Vnets - Domain Virtual Networking

Xen optionally supports virtual networking for domains using *vnets*. These emulate private LANs that domains can use. Domains on the same vnet can be hosted on the same machine or on separate machines, and the vnets remain connected if domains are migrated. Ethernet traffic on a vnet is tunneled inside IP packets on the physical network. A vnet is a virtual network and addressing within it need have no relation to addressing on the underlying physical network. Separate vnets, or vnets and the physical network, can be connected using domains with more than one network interface and enabling IP forwarding or bridging in the usual way.

Vnet support is included in `xm` and xend:

```
# xm vnet-create <config>
```

creates a vnet using the configuration in the file `<config>`. When a vnet is created its configuration is stored by xend and the vnet persists until it is deleted using

```
# xm vnet-delete <vnetid>
```

The vnets xend knows about are listed by

```
# xm vnet-list
```

More vnet management commands are available using the `vn` tool included in the vnet distribution.

The format of a vnet configuration file is

```
(vnet (id       <vnetid>)
      (bridge   <bridge>)
      (vnetif   <vnet interface>)
      (security <level>))
```

White space is not significant. The parameters are:

- `<vnetid>`: vnet id, the 128-bit vnet identifier. This can be given as 8 4-digit hex numbers separated by colons, or in short form as a single 4-digit hex number. The short form is the same as the long form with the first 7 fields zero. Vnet ids must be non-zero and id 1 is reserved.
- `<bridge>`: the name of a bridge interface to create for the vnet. Domains are connected to the vnet by connecting their virtual interfaces to the bridge. Bridge names are limited to 14 characters by the kernel.
- `<vnetif>`: the name of the virtual interface onto the vnet (optional). The interface encapsulates and decapsulates vnet traffic for the network and is attached to the vnet bridge. Interface names are limited to 14 characters by the kernel.
- `<level>`: security level for the vnet (optional). The level may be one of
  - `none`: no security (default). Vnet traffic is in clear on the network.
  - `auth`: authentication. Vnet traffic is authenticated using IPSEC ESP with hmac96.
  - `conf`: confidentiality. Vnet traffic is authenticated and encrypted using IPSEC ESP with hmac96 and AES-128.

  Authentication and confidentiality are experimental and use hard-wired keys at present.

When a vnet is created its configuration is stored by xend and the vnet persists until it is deleted using `xm vnet-delete <vnetid>`. The interfaces and bridges used by vnets are visible in the output of `ifconfig` and `brctl show`.

B.1 Example

If the file `vnet97.sxp` contains

```
(vnet (id 97) (bridge vnet97) (vnetif vnif97)
      (security none))
```

Then `xm vnet-create vnet97.sxp` will define a vnet with id 97 and no security. The bridge for the vnet is called vnet97 and the virtual interface for it is vnif97. To add an interface on a domain to this vnet set its bridge to vnet97 in its configuration. In Python:

```
vif="bridge=vnet97"
```

In sxp:

```
(dev (vif (mac aa:00:00:01:02:03) (bridge vnet97)))
```

Once the domain is started you should see its interface in the output of `brctl show` under the ports for `vnet97`.

To get best performance it is a good idea to reduce the MTU of a domain's interface onto a vnet to 1400. For example using `ifconfig eth0 mtu 1400` or putting `MTU=1400` in `ifcfg-eth0`. You may also have to change or remove cached config files for eth0 under `/etc/sysconfig/networking`. Vnets work anyway, but performance can be reduced by IP fragmentation caused by the vnet encapsulation exceeding the hardware MTU.

B.2 Installing vnet support

Vnets are implemented using a kernel module, which needs to be loaded before they can be used. You can either do this manually before starting xend, using the command `vn insmod`, or configure xend to use the `network-vnet` script in the xend configuration file `/etc/xend/xend-config.sxp`:

```
(network-script        network-vnet)
```

This script insmods the module and calls the `network-bridge` script.

The vnet code is not compiled and installed by default. To compile the code and install on the current system use `make install` in the root of the vnet source tree, `tools/vnet`. It is also possible to install to an installation directory using `make dist`. See the `Makefile` in the source for details.

The vnet module creates vnet interfaces `vnif0002`, `vnif0003` and `vnif0004` by default. You can test that vnets are working by configuring IP addresses on these interfaces and trying to ping them across the network. For example, using machines hostA and hostB:

```
hostA# ifconfig vnif0004 192.0.2.100 up
hostB# ifconfig vnif0004 192.0.2.101 up
hostB# ping 192.0.2.100
```

The vnet implementation uses IP multicast to discover vnet interfaces, so all machines hosting vnets must be reachable by multicast. Network switches are often configured not to forward multicast packets, so this often means that all machines using a vnet must be on the same LAN segment, unless you configure vnet forwarding.

You can test multicast coverage by pinging the vnet multicast address:

```
# ping -b 224.10.0.1
```

You should see replies from all machines with the vnet module running. You can see if vnet packets are being sent or received by dumping traffic on the vnet UDP port:

```
# tcpdump udp port 1798
```

If multicast is not being forwarded between machines you can configure multicast forwarding using vn. Suppose we have machines hostA on 192.0.2.200 and hostB on

192.0.2.211 and that multicast is not forwarded between them. We use vn to configure each machine to forward to the other:

```
hostA# vn peer-add hostB
hostB# vn peer-add hostA
```

Multicast forwarding needs to be used carefully - you must avoid creating forwarding loops. Typically only one machine on a subnet needs to be configured to forward, as it will forward multicasts received from other machines on the subnet.

Appendix C

Glossary of Terms

Domain A domain is the execution context that contains a running **virtual machine**. The relationship between virtual machines and domains on Xen is similar to that between programs and processes in an operating system: a virtual machine is a persistent entity that resides on disk (somewhat like a program). When it is loaded for execution, it runs in a domain. Each domain has a **domain ID**.

Domain 0 The first domain to be started on a Xen machine. Domain 0 is responsible for managing the system.

Domain ID A unique identifier for a **domain**, analogous to a process ID in an operating system.

Full virtualization An approach to virtualization which requires no modifications to the hosted operating system, providing the illusion of a complete system of real hardware devices.

Hypervisor An alternative term for **VMM**, used because it means 'beyond supervisor', since it is responsible for managing multiple 'supervisor' kernels.

Live migration A technique for moving a running virtual machine to another physical host, without stopping it or the services running on it.

Paravirtualization An approach to virtualization which requires modifications to the operating system in order to run in a virtual machine. Xen uses paravirtualization but preserves binary compatibility for user space applications.

Shadow pagetables A technique for hiding the layout of machine memory from a virtual machine's operating system. Used in some **VMMs** to provide the illusion of contiguous physical memory, in Xen this is used during **live migration**.

Virtual Block Device Persistent storage available to a virtual machine, providing the abstraction of an actual block storage device. **VBD**s may be actual block devices, filesystem images, or remote/network storage.

Virtual Machine The environment in which a hosted operating system runs, providing the abstraction of a dedicated machine. A virtual machine may be identical to the underlying hardware (as in **full virtualization**, or it may differ, as in **paravirtualization**).

VMM Virtual Machine Monitor - the software that allows multiple virtual machines to be multiplexed on a single physical machine.

Xen Xen is a paravirtualizing virtual machine monitor, developed primarily by the Systems Research Group at the University of Cambridge Computer Laboratory.

XenLinux A name for the port of the Linux kernel that runs on Xen.