Secure Xen on ARM
User's Guide

Revision History

Version	Date	Revised contents
1.0	2008-06-11	Initial revision
1.1	2008-12-10	Adding guide about XenLinux, Xen-tool, and another platform support

Document Information

- Pages: 28 pages

Contact Information & Copyright

Samsung Electronics Co., Ltd.
14-1, Nongseo-dong, Giheung-gu, Yongin-si, Gyeonggi-do
Korea 446-712

Contact us: *sbuk.suh@samsung.com*

Table of Contents

Overview

Purpose

This document describes the procedure of setting up a development environment for Secure Xen on ARM solution, building the solution, deploying it to a real target, and booting it for creating VMs.

Terminology & Acronyms

Term	Description
VMM	Virtual Machine Monitor
VM	Virtual Machine
XenLinux	Linux kernel with patches applied so that it will run on the virtual architecture presented by the Secure Xen on ARM rather than on real hardware
Dom0	Privileged domain constructed by Secure Xen on ARM at initial start-up time.
Dom1	Unprivileged domain constructed by Dom0.
HID	Human Interface Device
Xenstore	Information storage space shared between domains
Foreground domain	Among running domains, the domain which currently interact with HID. (e.g. the GUI of foreground domain is shown in LCD.)

References

1. ARM Ltd., ARM926EJ-S Technical Reference Manual, r0p4/r0p5
2. Freescale Semiconduct, “i.MX21 Application Processor Reference Manual,” Rev. 2, 2005
3. Xen Interface Manual
   http://www.cl.cam.ac.uk/Research/SRG/netos/xen/readmes/interface/interface.html
4. “Secure Architecture and Implemation of Xen on ARM for Mobile Devices”, Sangbum Suh, presented at Xen summit Spring 2007, IBM TJ Watson
   http://www.xen.org/xensummit_4/Secure_Xen_ARM_xen-summit-04_07_Suh.pdf
5. “Xen on ARM: System Virtualization Using Xen Hypervisor for ARM-Based Secure Mobile Phones”, Joo-Young Hwang et. al., In Proceedings of the 5th Annual IEEE Consumer Communications & Networking Conference, USA, January 2008.
6. “A Multi-Layer Mandatory Access Control Mechanism for Mobile Devices Based on Virtualization,” Sung-Min Lee, Sang-bum Suh, Bokdeuk Jeong, Sangdok Mo, In Proceedings of the 5th Annual IEEE Consumer Communications & Networking Conference, USA, January 2008.

1. Introduction

1.1. Overview

Xen is an open source VMM originated as a research project at the University of Cambridge. Its first version, 1.0, came out in 2003 and now the version has reached to 3.3. The virtualization technique adopted by Xen is para-virtualization which requires operating system modification.

Secure Xen on ARM is an ARM port of the x86 version of Xen-3.0.2-2 plus security features in Xen. It allows the simultaneous execution of multiple operating systems and their legacy application software stacks on a single ARM core-based system-on-chip. Each guest OS instance runs in their own OS partition called “domain” and the OS partitions are securely isolated from each other.

The original Xen solution comes with many user-land utilities. We have ported most core component such as xend, xenstore, xm and xenconsole.

Notes: The current version of Secure Xen on ARM only supports “static partitioning” of system memory, which means that the number of guest domains and the amount of memory allocated to the guest domains is fixed at compile time. You have to configure system memory partitioning properly before building Secure Xen on ARM.
The shadow page table and the writable page table are not included in this release.

1.2. Environment

The development environment of Secure Xen on ARM is as follows:

- Host
  - OS: Fedora Core 6 is recommended. Other Linux distributions are not tested.
  - Compiler: GCC-3.4 or higher
- Target
  - HW: Freescale M9328MX21ADS board

2. Deployment Procedure

2.1. Preparation of Deployment

2.1.1. Cross Toolchain

The gcc 3.4.4 and glibc 2.3.5 is used for cross-compilation. You can download it at the following links:

http://www.ertos.nicta.com.au/downloads/tools/arm-linux-3.4.4.tar.gz
http://www.ertos.nicta.com.au/downloads/tools/arm-linux-3.4.4.tar.bz2

2.1.2. Tftp Service

You can download binary files from host PC to target by using tftp. If tftp service is not configured in host PC, install and setup a tftp service first.

```
# yum install tftp
# yum install tftp-server
# vi /etc/xinetd.d/tftp
Service tftp
{
        socket_type=dgram
        protocol            = udp
        wait                = yes
        user                = root
        server              = /usr/sbin/in.tftpd
        server_args         = -s /tftpboot
        disable             = no
        per_source          = 11
        cps                 = 100 2
        flags               = IPv4
}
# mkdir /tftpboot
# service xinetd restart
```

Figure 2-1. Configuring tftp service

Note: Installing tftp server and enabling tftp service can be different depending on your host PC environment.

2.1.3. Terminal Emulator

You can access to the target by using a terminal program. '**Minicom**' is one of the popular terminal programs running in Linux PC. Here we'll explain about serial port set-up in Minicom. You can also use other terminal programs such as '**HyperTerminal**' in Windows PC.

1. Execute a minicom on setting mode.

```
# minicom -s
```

2. Select '**Serial port setup**' menu and set up the parameters as in Figure 2-2.

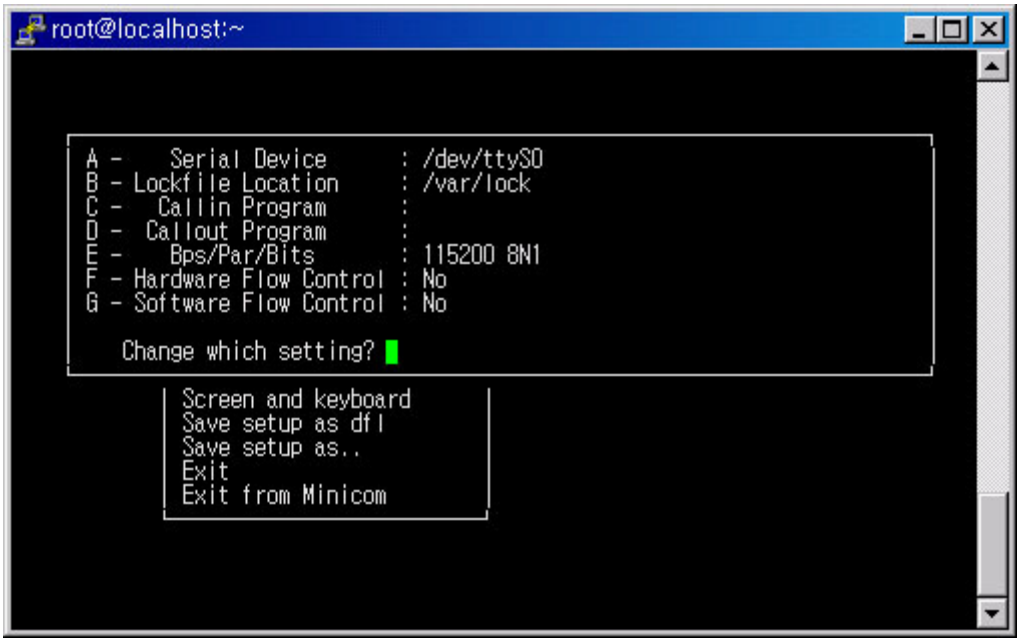

Figure 2-2. Minicom Configuration

2.2. Secure Xen on ARM and XenLinux Build

2.2.1. Secure Xen on ARM Build Procedure

1. Go to the source directory of Secure Xen on ARM. (Assumption: $(XEN_ROOT) is root directory of Secure Xen on ARM source.)

```
# cd $(XEN_ROOT)
```

2. Get on with the task of configuring the Secure Xen on ARM. When you do the make menuconfig to configure the Secure Xen on ARM, don't forget to check the system type that is suitable for your target board.

```
# make menuconfig
```

```
General Setup  --->
        [ ] optimize for size
        [ ] Use AEM EABI to compile the Secure Xen on ARM

System Type  --->
        Select target platform (Freescale i.MX21ADS board)  --->
                (X) Freescale i.MX21ADS board
                ( ) Android emulator board(Goldfish)
                ( ) ARM versatile_pb

Customize Memory Map  --->
        (0xFF000000) Hypervisor virtual address

Security Support  --->
        [ ] Security support
```

```
Debugging and profiling Support  --->
      [ ] Debugging and Profiling Support
```

Figure 2-3. Secure Xen on ARM Configuration

Note: If you intend to use a compiler with support of EABI, enable the corresponding option, **"General Setup -> Use EABI to compile the Secure Xen on ARM"**, in menuconfig.

3. Regarding each guest domain, configure the size of system memory and ramdisk, and the maximum size of kernel image file in menuconfig. Secure Xen on ARM decides the number of runnable domains and their memory sizes at compile time.

```
General Setup  --->
System Type  --->
Customize Memory Map  --->
      (0xFF000000) Hypervisor virtual address
            Domain Memory Size  --->
                  (0x02000000) domain0 memory size (including xen memory
size : 2MB)
                  (0x01000000) domain1 memory size (NEW)
                  (0x01000000) domain2 memory size (NEW)
                  (0x00000000) domain3 memory size (NEW)
            Image Max Size  --->
                  (0x00400000) domain0 image max size (NEW)
                  (0x00400000) domain1 image max size (NEW)
                  (0x00400000) domain2 image max size (NEW)
                  (0x00400000) domain3 image max size (NEW)
            Ram Disk Size  --->
                  (0x00400000) domain0 ramdisk size (NEW)
                  (0x00400000) domain1 ramdisk size (NEW)
                  (0x00400000) domain2 ramdisk size (NEW)
                  (0x00400000) domain3 ramdisk size (NEW)
Security Support  --->
Debugging and profiling Support  --->
```

Figure 2-4. Memory Partitioning

- Domain Memory Size: the size of memory allocated to guest domain
- Image Max Size: the upper limit of kernel image size of guest domain. (The image size should be smaller than the Image Max Size.)
- Ram Disk Size: the size of ramdisk. (It is ignored unless ramdisk is used.)

4. Set compiler prefix in Makefile.

```
CROSS_COMPILE = arm-linux-
```

5. Compile the Secure Xen on ARM by executing '**make xen**' command.

```
# make xen
```

6. Then **xen-bin** file is created in $(XEN_ROOT)/xen. Copy it to the root directory of tftp server, $(TFTP_ROOT).

```
# cp $(XEN_ROOT)/xen/xen-bin $(TFTP_ROOT)
```

2.2.2. XenLinux Build Procedure

1. Download Linux kernel 2.6.21.1 tarball and extract it. (Assumption: $(LINUX_ROOT) is root directory of the kernel.)

2. Patch the kernel and create symbolic link to refer to Secure Xen on ARM's header files.

```
# cd $(LINUX_ROOT)
# cp $(XEN_ROOT)/linux-spase/*  ./
# cd $(LINUX_ROOT)/include/xen
# ln -s $(XEN_ROOT)/xen/include/public  ./interface
```

3. Compile XenLinux. Then the kernel image files of Dom0 and Dom1 (vmlinux.out0 and vmlinux.out1) are created in $(LINUX_ROOT).

```
# ./do_compile.sh
```

4. Copy both vmlinux.out0 and vmlinux.out1 files to $(TFTP_ROOT).

```
# cp $( LINUX_ROOT)/vmlinux.out* $(TFTP_ROOT)
```

2.2.3. Xen-tool Build Procedure

1. First, build ARM-Linux python interpreter. (Python2.4.3_xcompile.patch file is in $(XEN_ROOT)/tools/arm_python directory.)

```
< Download Python-2.4.3 and extract it >
# patch -p1 < Python2.4.3_xcompile.patch
# cp $(XEN_ROOT)/tools/arm_python/cross_build.sh $(PYTHON_ROOT)/
# cd $(PYTHON_ROOT)
# ./cross_build -root ($XEN_ROOT)/tools/arm_python/cross_compiled \
-compiler $(CROSS_COMPILER)
```

2. Compile Xen-tool.

```
# cd $(XEN_ROOT)/tools
# make
# make install
```

Note: "make install" installs the tools necessary for booting Dom1 in $(XEN_ROOT)/tools/target directory. When making a root file system of Dom0, you should include them in the file system.

2.3. Deploying Secure Xen on ARM to Target Platform

In Freescale M9328MX21ADS board, part of NOR flash memory is used as Dom0's root file system. Figure 2-5 shows MTD partition layout of NOR flash memory, which has three partitions (P0, P1, and P2) and P2 is the partition for Dom0's root file system.

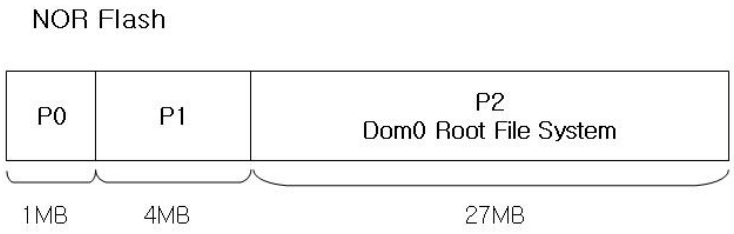

Figure 2-5. NOR Flash Memory Partition Layout

2.3.1. Writing Dom0's Root File System to NOR Flash Memory

1. Reboot target and switch to the prompt mode by pressing space-bar in keyboard.

```
……
# NAND device: Manufacturer ID: 0xec, Chip ID: 0x36 (Samsung mx2nand)
64 MiB
In:    serial
Out:   serial
Err:   serial
To interrupt autoboot, press space-bar...(3 sec delay)
Samsung:mx21ads>
```

2. Load Dom0's root file system image file to target memory.

```
Samsung:mx21ads> set serverip $(IP address of the tftp server)
Samsung:mx21ads> set ipaddr $(IP address of the target)
Samsung:mx21ads> tftp 0xc1000000 $(name of  Dom0 root file system file)
```

3. Write the image file to P2 partition of NOR.

```
Samsung:mx21ads> protect off all
Samsung:mx21ads> erase 0xc9000000 0xc9ffffff
Samsung:mx21ads> cp.b 0xc1000000 0xc9000000 $filesize
Samsung:mx21ads> protect on all
```

2.4. Running Xen and XenLinux

2.4.1. Booting Xen and Dom0

1. Turn on target board and switch to the prompt mode.

2. Load the Secure Xen on ARM binary and Dom0's kernel image file to target memory.
   A. **tftp 0xc0008000 xen-bin**
   B. **tftp 0xc1c00000 vmlinux.out0**

Note: Depending on the memory size of Dom0, the address where vmlinux.out0 is loaded varies.

3. Execute the Secure Xen on ARM.l

A. go 0xc0008000

Note: In current configuration, Xen-bin should be loaded at 0xc0008000 and vmlinux.out0 should be at 0xc1c00000.

U-boot	Flash part: manuf=0x1 id1=0x227e id2=0x2218 id3=0x2200 Flash: 32 MB NAND: MX2 NAND: 8-bit i/o mode NAND device: Manufacturer ID: 0xec, Chip ID: 0x36 (Samsung mx2nand) 64 MiB In: serial Out: serial Err: serial To interrupt autoboot, press space-bar...(3 sec delay) TFTP from server 169.254.100.1; our IP address is 169.254.100.2 Filename 'xen-bin'. Load address: 0xc0008000 Loading: ## done Bytes transferred = 226584 (37518 hex) TFTP from server 169.254.100.1; our IP address is 169.254.100.2 Filename 'vmlinux.out0'. Load address: 0xc1c00000 Loading: ## ## ## ## ## ## ## ############################## done Bytes transferred = 2475584 (25c640 hex) ## Starting application at 0xC0008000 ...
Secure Xen On ARM	[XEN] [XEN]Xen/ARM virtual machine monitor for ARM architecture [XEN] Copyright (C) 2007 Samsung Electronics Co, Ltd. All Rights Reserved. [XEN] Using scheduler: Borrowed Virtual Time (bvt) [XEN] *** LOADING DOMAIN : 0 *** [XEN] Physical Memory Arrangement: c0200000->c2000000 [XEN] VIRTUAL MEMORY ARRANGEMENT: [XEN] Loaded kernel: c0008000->c032d444 [XEN] Init. ramdisk: c032e000->c032e000 [XEN] Phys-Mach map: c032e000->c0335800 [XEN] Start info: c0336000->c0337000 [XEN] Page tables: c0338000->c034e000 [XEN] Boot stack: c034e000->c034f000 [XEN] TOTAL: c0000000->c1e00000 [XEN] ENTRY ADDRESS: c0008000 [XEN] [TODO] dma channel access permission, in construct_dom0() [XEN] [dom0] Xen Start info :
Dom0	[dom0] Magic : xen-3.0-arm_32 [dom0] Total Pages allocated to this domain : 7680 [dom0] MACHINE address of shared info struct : 0x3222228992x [dom0] VIRTUAL address of page directory : 0xc0338000

```
[dom0] Number of bootstrap p.t. frames : 22
[dom0] VIRTUAL address of page-frame list : 0xc032e000
[dom0] VIRTUAL address of pre-loaded module : 0x0
[dom0] Size (bytes) of pre-loaded modules : 0
[dom0] min mfn(min_page in xen) : 786432
[dom0] Command-Linux Address : 0xc0336054
[dom0] Command-Line String :
[dom0] Guest PHYS_OFFSET : 0xc0200000
[dom0] set hypervisor set callback
[dom0] no problem
[dom0] no problem, shared info address is : c0009000
Linux version 2.6.21.1 (root@dh.vmm) (gcc version 3.4.4) #6 Thu Nov 13 16:38:52 KST
2008
CPU: ARM926EJ-S [41069264] revision 4 (ARMv5TEJ), cr=00000000
Machine: Freescale IMX21ADS
Memory policy: ECC disabled, Data cache writeback
Built 1 zonelists.  Total pages: 7620
Kernel command line: console=ttyS0,115200 root=/dev/mtdblock2 rootfstype=jffs2
PID hash table entries: 128 (order: 7, 512 bytes)
Console: colour dummy device 80x30
Dentry cache hash table entries: 4096 (order: 2, 16384 bytes)
Inode-cache hash table entries: 2048 (order: 1, 8192 bytes)
Memory: 30MB = 30MB total
Memory: 27040KB available (2684K code, 421K data, 112K init)
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok
NET: Registered protocol family 16
xenbus_probe_init invoking!
backend XENBUS : Event Channel for Xenstore : 2
xs_init invoking!
Bluetooth: Core ver 2.11
NET: Registered protocol family 31
Bluetooth: HCI device and connection manager initialized
Bluetooth: HCI socket layer initialized
NET: Registered protocol family 2
IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
TCP established hash table entries: 1024 (order: 1, 8192 bytes)
TCP bind hash table entries: 1024 (order: 0, 4096 bytes)
TCP: Hash tables configured (established 1024 bind 1024)
TCP reno registered
NetWinder Floating Point Emulator V0.97 (double precision)
JFFS2 version 2.2. (NAND) (C) 2001-2006 Red Hat, Inc.
fuse init (API version 7.8)
io scheduler noop registered
io scheduler anticipatory registered
io scheduler deadline registered
io scheduler cfq registered (default)
i.MX Framebuffer driver
Frame buffer SSA = c0680000
gw:i.MX Framebuffer driver
[SY]GW SAR = c067e000
Serial: IMX driver
imx-uart.0: ttyS0 at MMIO 0xe000a000 (irq = 20) is a IMX
imx-uart.1: ttyS1 at MMIO 0xe000b000 (irq = 19) is a IMX
loop: loaded (max 8 devices)
cs89x0:cs89x0_probe(0x0)
cs89x0.c: v2.4.3-pre1 Russell Nelson <nelson@crynwr.com>, Andrew Morton
<andrewm@uow.edu.au>
eth0: cs8900 rev K found at 0xec000300 [Cirrus EEPROM]
cs89x0 media RJ-45, IRQ 203, programmed I/O, MAC 00:04:9f:00:64:78
cs89x0_probe1() successful
cs89x0:cs89x0_probe(0x0)
cs89x0: request_region(0xec000300, 0x10) failed
cs89x0: no cs8900 or cs8920 detected.  Be sure to disable PnP with SETUP
```

```
Probing flash00 at physical address 0xc8000000 (32-bit bankwidth)
flash00: Found 2 x16 devices at 0x0 in 32-bit bank
 Amd/Fujitsu Extended Query Table at 0x0040
flash00: CFI does not contain boot bank location. Assuming top.
number of CFI chips: 1
cfi_cmdset_0002: Disabling erase-suspend-program due to code brokenness.
Detected Spansion S29WS128N flash chips.
Erase block size is 0x40000
mx2ads: using command line partition definition
Creating 3 MTD partitions on "flash00":
0x00000000-0x00100000 : "bootloader"
0x00100000-0x00500000 : "Kernel Image Partition"
0x00500000-0x02000000 : "Dom0 Root Filesystem"
mice: PS/2 mouse device common for all mice
input: Keypad for Freescale-Suzhou as /class/input/input0
Kpp Driver 1.0.0, for Freescale-Suzhou
Pen Driver 0.4.0, Motorola SPS-Suzhou
i2c /dev entries driver
[dom0] Setup i2c_imx driver structure
[dom0] init_waitqueue_head(&i2c_imx->queue)
[dom0] platform_set_drvdata(pdev, i2c_imx)
[dom0] ret = request_irq(...) => [dom0] 0
[dom0] i2c_set_adapdata()
[dom0] i2c_imx_set_clk()
[dom0] hclk = imx_get_hclk()
[dom0] desired_div = 1440
[dom0] writeb()
[dom0] disable_delay = 11
[dom0] writeb()
[dom0] imx_gpio_mode()
[dom0] imx_gpio_mode()
[dom0] writeb()
[dom0] writeb()
[dom0] ret = i2c_add_adapter() => [dom0] 0
Bluetooth: HCI UART driver ver 2.2
Bluetooth: HCI BCSP protocol initialized
Event-channel device installed.
nf_conntrack version 0.5.0 (240 buckets, 1920 max)
ip_tables: (C) 2000-2006 Netfilter Core Team
TCP cubic registered
NET: Registered protocol family 1
NET: Registered protocol family 17
Bridge firewalling registered
Bluetooth: L2CAP ver 2.8
Bluetooth: L2CAP socket layer initialized
Bluetooth: RFCOMM socket layer initialized
Bluetooth: RFCOMM ver 1.8
Bluetooth: BNEP (Ethernet Emulation) ver 1.2
VFS: Mounted root (jffs2 filesystem) readonly.
init started:  BusyBox v1.00 (2006.03.28-11:05+0000) multi-call binary
mount: /etc/mtab: Read-only file system
cat: Write Error: No space left on device
*** Running rc.modules
*** Running rc.serial
*** Running rc.xen
*** Attempting to start S05syslog
Starting /sbin/syslogd
Done
Starting /sbin/klogd
Done
*** Attempting to start S20network
Setting up link for loopback
Done
Setting up link for eth0
```

```
eth0: using half-duplex 10Base-T (RJ-45)

Done
*** Attempting to start S23portmap
Starting /sbin/portmap
Done
*** Running rc.local

samsung login: root
Password:


BusyBox v1.00 (2006.03.28-11:05+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

#
```

Figure 2-6. Console message after booting Secure Xen on ARM and Dom0

2.4.2. Booting Dom1

1. Start xenstore and xend.

```
# export PATH=$PATH:/usr/local/xen-tools/bin:/usr/local/xen-tools/sbin:/usr/local/arm-python/bin
# export LD_LIBRARY_PATH=/usr/local/arm-python/lib:/usr/local/xen-tools/lib
# mkdir -p /var/run/xenstored/; rm /var/lib; mkdir -p /var/lib/xenstored/; mkdir -p/var/lib/xend/
# xend start
# xenconsoled
```

Download Dom1's XenLinux kernel image file (vmlinux.out1) from host PC to a directory of Dom0's root file system. (Here we assume the directory is **/images**.)

2. Launch Dom1.

```
# xm create /etc/xen/dom1
```

Note: To see Dom1's booting message, refer to in 2.4.3

3. Check that Dom1 is registed in xenstore with **xenstore-ls** and see Dom1's state with **xentop**.

```
# xenstore-ls
# xentop
```

```
# xm create /etc/xen/dom1
Using config file "/etc/xen/dom1".
privcmd_ioctl, virual address : 40748000, mfn : c2c00 npages : 400
*** LOADING DOMAIN : 1 ***
[XEN] Physical Memory Arrangement: c2000000->c3000000
[XEN] VIRTUAL MEMORY ARRANGEMENT:
[XEN]  Loaded kernel: c0008000->c02c3d78
[XEN]  Init. ramdisk: c02c4000->c02c4000
[XEN]  Phys-Mach map: c02c4000->c02c8000
[XEN]  Store mfn:     c02c8000->c02c9000
[XEN]  Console mfn:   c02c9000->c02ca000
```

```
[XEN]  Start info:    c02ca000->c02cb000
[XEN]  Page tables:   c02cc000->c02db000
[XEN]  Boot stack:    c02db000->c02dc000
[XEN]  TOTAL:         c0000000->c1000000
[XEN]  ENTRY ADDRESS: c0008000
[XEN] store_mfn physical addres c22c8000
[XEN] console_mfn physical addres c22c9000
[XEN]  [TODO] dma channel access permission, in construct_guest_dom()
privcmd_ioctl, virual address : 40018000, mfn : c22c8 npages : 1
fbsetback_probe called
fbsetback_probe: be->fbsetif = 0xc0b728a0
privcmd_ioctl, virual address : 40016000, mfn : c22c9 npages : 1
[XEN] [dom1] Xen Start info :
[dom1] Magic : xen-3.0-arm_32
[dom1] Total Pages allocated to this domain : 4096
[dom1] MACHINE address of shared info struct : 0x3222208512x
[dom1] VIRTUAL address of page directory : 0xc02cc000
[dom1] Number of bootstrap p.t. frames : 15
[dom1] VIRTUAL address of page-frame list : 0xc02c4000
[dom1] VIRTUAL address of pre-loaded module : 0x0
[dom1] Size (bytes) of pre-loaded modules : 0
[dom1] min mfn(min_page in xen) : 786432
[dom1] Command-Linux Address : 0xc02ca054
[dom1] Command-Line String :
[dom1] Guest PHYS_OFFSET : 0xc2000000
[dom1] set hypervisor set callback
[dom1] no problem
[dom1] no problem, shared info address is : c000a000
Started domain dom1
# [dom1] xencons_open: c0ca8000
[dom1] xencons_open: c0ca8000
[dom1] xencons_open: c0ca8000
[dom1] xencons_open: c0ca8000
[dom1] xencons_open: c0ca8000
[dom1] xencons_open: c0ca8000

#
```

Figure 2-7. Console message after booting Dom1

```
# ./xenstore-ls
tool = ""
 xenstored = ""
vm = ""
 00000000-0000-0000-0000-000000000000 = ""
  ssidref = "1074925120"
  uuid = "00000000-0000-0000-0000-000000000000"
  on_reboot = "restart"
  on_poweroff = "destroy"
  name = "Domain-0"
  vcpus = "1"
  vcpu_avail = "1"
  memory = "30"
  on_crash = "restart"
  maxmem = "30"
 921be47f-ce3f-7720-c4e7-e1f9c05e1cb1 = ""
  image = "(linux (kernel /images/vmlinux.out1))"
   ostype = "linux"
   kernel = "/images/vmlinux.out1"
   cmdline = ""
   ramdisk = ""
  ssidref = "0"
  uuid = "921be47f-ce3f-7720-c4e7-e1f9c05e1cb1"
```

```
  on_reboot = "restart"
  start_time = "45.146142"
  on_poweroff = "destroy"
  name = "dom1"
  vcpus = "1"
  vcpu_avail = "1"
  memory = "16"
  on_crash = "restart"
  maxmem = "16"
 local = ""
 domain = ""
 0 = ""
  cpu = ""
   0 = ""
    availability = "online"
  memory = ""
   target = "30720"
  name = "Domain-0"
  console = ""
   limit = "1048576"
  vm = "/vm/00000000-0000-0000-0000-000000000000"
  domid = "0"
  backend = ""
   vkpp = ""
    1 = ""
     1 = ""
      frontend-id = "1"
      domain = "dom1"
      ssa = "0"
      state = "4"
      frontend = "/local/domain/1/device/vkpp/1"
   vfb = ""
    1 = ""
     1 = ""
      frontend-id = "1"
      domain = "dom1"
      frontend = "/local/domain/1/device/vfb/1"
      state = "4"
 1 = ""
  device = ""
   vkpp = ""
    1 = ""
     virtual-device = "1"
     backend-id = "0"
     state = "4"
     backend = "/local/domain/0/backend/vkpp/1/1"
     ring-ref = "9"
     event-channel = "6"
   vfb = ""
    1 = ""
     state = "4"
     backend-id = "0"
     backend = "/local/domain/0/backend/vfb/1/1"
     ring-ref = "8"
     event-channel = "5"
  console = ""
   ring-ref = "795337"
   port = "2"
   limit = "1048576"
   tty = "/dev/pts/0"
  name = "dom1"
  vm = "/vm/921be47f-ce3f-7720-c4e7-e1f9c05e1cb1"
  domid = "1"
  cpu = ""
```

```
        0 = ""
         availability = "online"
        memory = ""
         target = "16384"
        store = ""
         ring-ref = "795336"
         port = "1"
#
```

Figure 2-8. xenstore-ls

```
xentop - 09:43:43   Xen 0.0Da^A
2 domains: 1 running, 0 blocked, 0 paused, 0 crashed, 0 dying, 0 shutdown
Mem: 4301437344k total, 4301373280k used, 64064k free    CPUs: 1 @ 266MHz
      NAME  STATE    CPU(sec) CPU(%)     MEM(k) MEM(%)  MAXMEM(k) MAXMEM(%) VCPUS
 NETS NETTX(k) NETRX(k) SSID
      dom1 ------          2    0.0      16384    0.0   no limit       n/a      1
    0        0        0 4279189256
  Domain-0 -----r        183    0.7      30720    0.0   no limit       n/a      1
    0        0        0 4279189256



  Delay  Networks  VCPUs  Repeat header  Sort order  Quit
```

Figure 2-9. xentop

2.4.3. Swich a Foreground Domain

In order to switch a foreground domain among the guest domains (Dom0 and Dom1), you might just use the magic key in target's keypad. Currently the magic key is assigned to the "**SW26**" button on keypad module. When it is pressed, the foreground domain change occurs. The GUI of foreground domain is displayed in LCD panel.

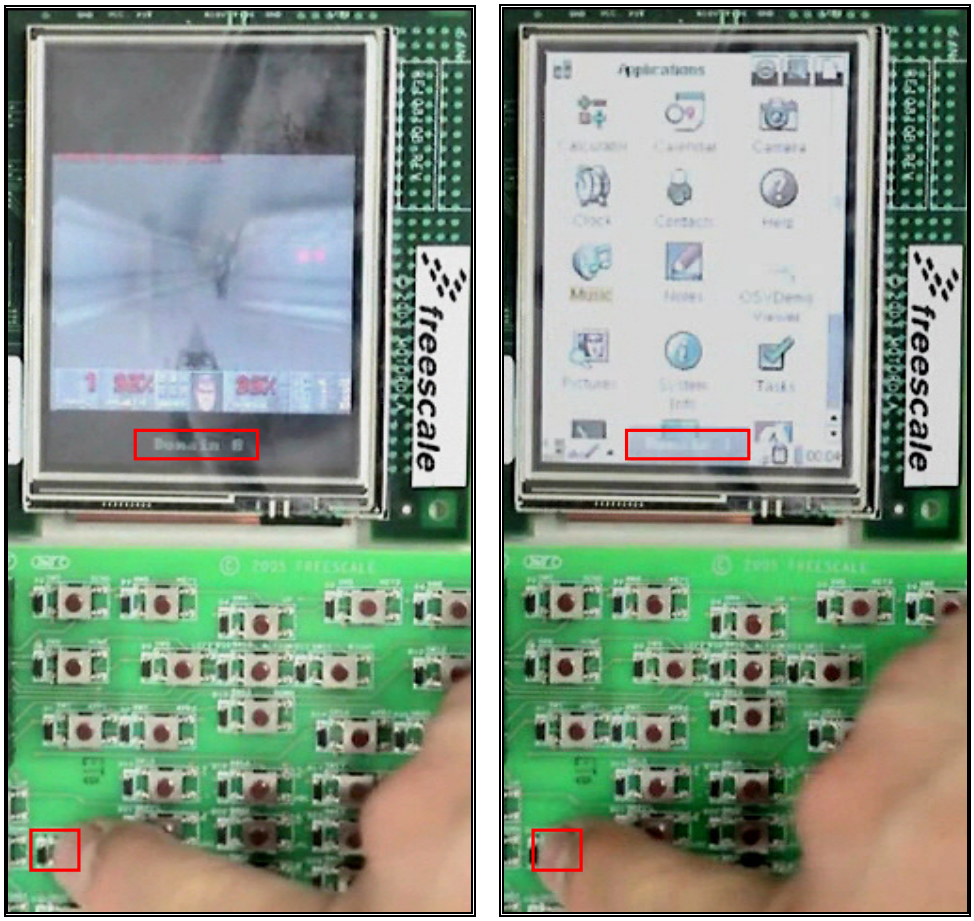

Figure 2-10. Switching foreground domain

2.4.4. xenconsole

When you want to see the console message of Dom1, you can use xenconsole application in Dom0 as follows:

xenconsole 1

Then you can use Dom1's console. By inputting 'ctrl' and ']' keys in keyboard simultanesously (**'ctrl' + ']'**), you can get back to Dom0's console.

```
# ps -x
  PID  Uid     VmSize Stat Command
    1 root       556 S    init
    2 root           SWN [ksoftirqd/0]
    3 root           SW< [events/0]
    4 root           SW< [khelper]
    5 root           SW< [kthread]
    7 root           SW< [xenwatch]
    8 root           SW< [xenbus]
   32 root           SW< [kblockd/0]
   33 root           SW< [kseriod]
   47 root           SW  [pdflush]
   48 root           SW  [pdflush]
   49 root           SW< [kswapd0]
   50 root           SW< [aio/0]
  649 root           SW  [mtdblockd]
  673 root           SW< [krfcommd]
  683 root           SWN [jffs2_gcd_mtd2]
  715 root       448 S    /sbin/syslogd
  722 root       420 S    /sbin/klogd
  752 bin        304 S    /sbin/portmap
  760 root       864 S    -sh
  785 root           SW< [rpciod/0]
  786 root           SW  [lockd]
  806 root       764 S    xenstored --pid-file=/var/run/xenstore.pid
  809 root      4136 S    python /usr/sbin/xend start
  811 root       524 S    xenconsoled
  812 root       524 S    xenconsoled
  813 root       524 S    xenconsoled
  814 root      5076 S    python /usr/sbin/xend start
  815 root      5076 S    python /usr/sbin/xend start
  816 root      5076 S    python /usr/sbin/xend start
  817 root      5076 S    python /usr/sbin/xend start
  818 root      5076 S    python /usr/sbin/xend start
  819 root      5076 S    python /usr/sbin/xend start
  834 root       676 R    ps -x
# xenconsole 1
ready to change dom1
ready to change dom1, fb_ssa=0xc2f40000

/ $ ps -x
  PID  Uid     VmSize Stat Command
    1 root       536 S    init
    2 root           SWN [ksoftirqd/0]
    3 root           SW< [events/0]
    4 root           SW< [khelper]
    5 root           SW< [kthread]
    7 root           SW< [xenwatch]
    8 root           SW< [xenbus]
   25 root           SW< [kblockd/0]
   26 root           SW< [kseriod]
   38 root           SW  [pdflush]
   39 root           SW  [pdflush]
   40 root           SW< [kswapd0]
   41 root           SW< [aio/0]
  146 root       372 S    /sbin/klogd
  157 root       748 S    /bin/sh
  158 root       660 R    ps -x
```

Figure 2-11. Using xenconsole

(Blue box: Dom0's console message, red box: Dom1's console message)

3. How to Enable Security Features

Secure Xen on ARM provides some kinds of security features. This chapter explains how users can use them. Two jobs are required to enable security features. One is about boot loader, and the other is about cryptographic library. Without this job, Secure Xen on ARM may not operate properly.

3.1. Why modify boot loader for Secure Xen on ARM?

In order to use security features, Secure Xen on ARM should be based on security-chain (i.e., security-enabled boot loader → Secure Xen on ARM → security-enabled OS). In this release of Secure Xen on ARM, however, you cannot use full security features we made because this release does not include security-enabled boot loader and security-enabled OS. The core of security-chain is composed of secure-boot and security-information-transfer from boot loader to Secure Xen on ARM.

3.1.1. Secure boot

Security-enabled boot loader makes use of data located on the secure ROM (actually it depends on a target platform feature) and other persistent memory like NOR or NAND flash memory. The data needed for security-enabled boot loader include master key, certificate, access control policies, binary images, their signatures and so on. To make security-enabled boot loader run correctly, these data should be stored on a secure ROM and a persistent memory.

The process of secure boot is as follows:
First of all, boot loader initializes security information for secure boot and checks integrity of Secure Xen on ARM image. If the Secure Xen on ARM image is not changed, boot loader starts it.
In addition, the Secure Xen on ARM receives the security information from boot loader and checks integrity of OS images. If the OS images are not changed, Secure Xen on ARM runs them.

3.1.2. Security information transfer

The security information described above is shared with a boot loader and Secure Xen on ARM. So the security information must be transferred from the boot loader to Secure Xen on ARM. To share the security information, the security-enabled boot loader calls the Secure Xen on ARM image just as a function with a parameter and transfers the security information through the parameter.

3.2. Detailed description about boot loader operation

3.2.1. Initialization

As we described above, the security information, which consist of master key, certificate, access control policies, and binary images (VMM image, kernel images) & their signatures, etc., are used for secure boot and other security mechanisms by boot loader and Secure Xen on ARM. So the boot loader reads these information right after it runs. Generally, the master key is loaded from a secure ROM, and others are loaded from a persistent memory, but it depends on a target board.

3.2.2. To load Secure Xen on ARM and Dom0 binary images to predefined memory locations

Based on the security information, the boot loader knows the memory location to be loaded Secure Xen on ARM and OS image. The boot loader loads Secure Xen on ARM and OS binary image as well as other security information (e.g., access control policy, cryptographic keys, etc) to predefined memory location.

3.2.3. To verify Secure Xen on ARM binary image

It is important that the boot loader checks integrity of the Secure Xen on ARM image with its signature, because the Secure Xen on ARM modified by a malicious entity can be used to attack the OSes on top of it. If the Secure Xen on ARM is altered, the boot loader stops its execution and warns about this modification.

3.2.4. To call the Secure Xen on ARM binary image with a parameter

If the verification is successfully passed, the boot loader calls the Secure Xen on ARM binary image as a function with a parameter, which is a pointer type of 'ssb_transfer_container_t' structure defined in 'secure_storage_struct.h'. Followings are sample codes (see Sample code 3-1) for a security-enabled boot loader to start Secure Xen on ARM.

```
/* XEN_ARM_POS       predefined memory location for Secure Xen on ARM */
/* SEC_INFO_POS      memory location for security information structure */

void (*start_xen_arm)(ssb_image_container_t *trans);
ssb_transfer_container_t *ptc=NULL;

start_xen_arm = (void(*)(ssb_image_container_t *)) XEN_ARM_POS;
ptc = (ssb_transfer_container_t *) SEC_INFO_POS;
/* Secure Xen on ARM start */
(*start_xen_arm)(ptc);
```

Sample code 3-1

3.3. Data structure to be transferred from boot loader to Secure Xen on ARM

In the 'secure_storage_struct.h' header file, you can see the detailed data structure to be transferred from a boot loader to Secure Xen on ARM. As you can see from the sample codes below, transferred structure is the 'ssb_transfer_container_t'. This structure contains 'transfer_struct_t' which has a binary image for each secure partition (see Sample code 3-2).

```
typedef enum {
  PART_MBB, PART_SP1=1, PART_SP2=2, PART_SP3=3, PART_OS_IMAGE=4,
  PART_DRV_RFS, PART_DOM0_RFS, PART_DOM1_RFS, PART_DOM2_RFS,
  TRANSFER_MASTER_KEY, PART_SUB_VMM_IMAGE,
  PART_END
} partition_type_t;

typedef struct {
  partition_type_t type;
  u_int32_t size;
  char *ptr;
} transfer_struct_t;
```

```
typedef struct {
  u_int32_t images_size;
  transfer_struct_t image[MAX_IMAGE_STRUCT_SIZE];
} ssb_transfer_container_t;
```

Sample code 3-2

If you would like to know more about transferred data structure and binary format, see the following functions in the 'sra_func.c' and 'crypto.c' (see Sample code 3-3).

```
int sra_init(void);
int crypto_init(void);
int _load_part(ssb_transfer_container_t *tc, partition_type_t ptype);
int init_master_key(void);
```

Sample code 3-3

3.4. Description about cryptographic library

3.4.1. Overview

The 'crypto.c' file in the 'crypto' directory contains APIs for cryptographic functions and these APIs are used for cryptographic operations such as key/algorithms initialization, hashing, encryption, decryption, and digital signature (They are called by Secure Xen on ARM). If there is cryptographic hardware support, you can connect these APIs to hardware cryptographic engine. Otherwise, you can use open source cryptographic library such as OpenSSL. In this release of Secure Xen on ARM, only cryptographic APIs are provided. So if you want to use cryptographic APIs, you must get cryptographic library first and connect those APIs to cryptographic functions.

3.4.2. APIs for cryptographic library

3.4.2.1. int init_master_key(void);

Description

This function loads the master key to be used by cryptographic library.
This function must be called once before other cryptographic library is called.

Parameters

None

Return Value

0 if succeeds, otherwise error value

3.4.2.2. int crypto_init(void);

Description

This function selects cryptographic algorithm and gets the RSA public key or public key certificate
This function must be called once before other cryptographic library is called.

Parameters

None

Return Value

0 if succeeds, otherwise error value

3.4.2.3. int crypto_sign_data(unsigned char* src, int src_len, unsigned char** sig, int* sig_len);

Description

This function generates a digital signature using RSA algorithm.

Parameters

Name	Type	Description
src	unsigned char *	Pointer to the source data
src_len	int	Byte length of source data
sig	unsigned char**	Pointer to the pointer to the digital signature. If *sig == NULL, newly allocated memory is returned. In the case, memory must be freed by caller
sig_len	int*	Byte length of digital signature

Return Value

0 if succeeds, otherwise error value

3.4.2.4. int crypto_verify_data(unsigned char* src, int src_len, unsigned char* sig, int sig_len);

Description

This function verifies image with digital signature using RSA algorithm.

Parameters

Name	Type	Description
src	unsigned char *	Pointer to the source data
src_len	int	Byte length of source data
sig	unsigned char*	Pointer to the digital signature.
sig_len	int	Byte length of digital signature

Return Value

0 if succeeds, otherwise error value

3.4.2.5. int crypto_verify_data_with_certm(unsigned char* src, int src_len, unsigned char* certm, int certm_len);

Description

This function verifies image with manufacture’s certificate using RSA algorithm.

Parameters

Name	Type	Description
src	unsigned char *	Pointer to the source data
src_len	int	Byte length of source data
certm	unsigned char*	Pointer to manufacture’s certificate
certm_len	int	byte length of manufacture’s certificate

Return Value

0 if succeeds, otherwise error value

3.4.2.6. int crypto_hash_data(unsigned char* src, int src_len, unsigned char** hash, int* hash_len);

Description

This function hashes image using SHA-1 algorithm

Parameters

Name	Type	Description
src	unsigned char *	Pointer to the source data
src_len	int	Byte length of source data
hash	unsigned char**	Pointer to the pointer to the hashed data. If *hash == NULL, newly allocated memory is returned. In the case, memory must be freed by caller
hash_len	Int *	byte length of hashed data

Return Value

0 if succeeds, otherwise error value

3.4.2.7. int crypto_encrypt_data(unsigned char *src, int src_len, unsigned char **enc, int *enc_len);

Description

This function encrypts data using AES algorithm

Parameters

Name	Type	Description
src	unsigned char *	Pointer to the source data
src_len	int	Byte length of source data
enc	unsigned char**	Pointer to the pointer to the encrypted data. If *enc == NULL, newly allocated memory is returned. In the case, memory must be freed by caller
enc_len	int*	Byte length of encrypted data

Return Value

0 if succeeds, otherwise error value

3.4.2.8. int crypto_decrypt_data(unsigned char *src, int src_len, unsigned char **dec, int *dec_len);

Description

This function decrypts data using AES algorithm

Parameters

Name	Type	Description
src	unsigned char *	Pointer to the source data
src_len	int	Byte length of source data
dec	unsigned char**	Pointer to the pointer to the decrypted data. If *dec == NULL, newly allocated memory is returned. In the case, memory must be freed by caller
dec_len	int*	Byte length of decrypted data

Return Value

0 if succeeds, otherwise error value

Appendix

A.1. New hypercalls for Secure Xen on ARM

Table 1 describes newly added hypercalls to Secure Xen on ARM.

Table 1New hypercalls for Secure Xen on ARM

Hypercall Name	Description
__HYPERVISOR_restore_guest_context	Restore CPU context stored in guest kernel stack.
__HYPERVISOR_do_print_profile	Dispatch profiling data
__HYPERVISOR_do_set_foreground_domain	Change foreground domain.
__HYPERVISOR_do_set_HID_irq	Register HID irq. The HID irq is only delivered to foreground domain select by **__HYPERVISOR_do_set_foreground_domain** hypercall.
__HYPERVISOR_dma_op	Request DMA operations
__HYPERVISOR_set_pirq_type	Change IRQ type and attributes
__ HYPERVIOSR_do_acm_op	Override native Xen hypercall. User can choose native or Secure Xen hypercall via menuconfig
__HYPERVISOR_sra_op	Manage secure storage data

A.2. Another platform support

Besides Freescale M9328MX21ADS board, Secure Xen on ARM supports the following platforms:

- Goldfish (QEMU 0.82 based Android emulator)
- ARM Versatile PB

In order to build “xen-bin” binary file for those platforms, you should configure the corresponding system type in menuconfig of Secure Xen on ARM.

```
# make menuconfig
```

```
General Setup  --->
        [ ] optimize for size
        [ ] Use AEM EABI to compile the Secure Xen on ARM

System Type  --->
        Select target platform (Freescale i.MX21ADS board)  --->
                ( ) Freescale i.MX21ADS board
                (X) Android emulator board(Goldfish)
```

```
                ( ) ARM versatile_pb

Customize Memory Map  --->
      (0xFF000000) Hypervisor virtual address

Security Support  --->
      [ ] Security support

Debugging and profiling Support  --->
      [ ] Debugging and Profiling Support
```

And $(XEN_ROOT)/Config.mk file should be modified to configure $(XEN_TARGET_SUBARCH).

```
.....................
#XEN_TARGET_SUBARCH  ?= imx21
XEN_TARGET_SUBARCH  ?= goldfish
#XEN_TARGET_SUBARCH  ?= versatile
.....................
```

In case of Goldfish platform, follow the below steps for target deployment.

1. Build Android emulator. (**$(ANDROID_EMUL)** is the root directory of Android emulator source codes.)

```
> tar xvjf android-emulator-xen_arm-081120.tar.bz2
> cd $(ANDROID_EMUL)
> ./build-emulator.sh
```

2. Build Secure Xen on ARM.

```
> cd $(XEN_ROOT)
> make menuconfig
> make xen
> cp xen/xen-bin $(ANDROID_EMUL)/images/kernel-qemu
```

3. Build mini-OS.

```
> cd $(XEN_ROOT)/extras/mini-os-arm/
> make
> cp mini-os.elf $(ANDROID_EMUL)/
```

4. Launch the Secure Xen on ARM on the Goldfish.

```
> cp $(XEN_ROOT)/xen/xen-bin $(ANDROID_EMUL)
> ./run.sh
```

5. Open debug console and check the console message.

A.3. Credit

Sangbum Suh (sbuk.suh@samsung.com)
Jooyoung Hwang (jooyoung.hwang@samsung.com)
Sungmin Lee (sung.min.lee@samsung.com)
Chanju Park (bestworld@samsung.com)
Sungkwan Heo (sk.heo@samsung.com)
Sangdok Mo (sd.mo@samsung.com)
Jaemin Ryu (jy0922.shim@samsung.com)
Bokdeuk Jung (bd.jeong@samsung.com)
Junghyun Yoo (yjhyun.yoo@samsung.com)
Minsung Jang (minsung.jang@samsung.com)
Joonyoung Shim (jy0922.shim@samsung.com)
Donghyuk Lee (dh5050.lee@samsung.com)
Inki Dae (inki.dae@samsung.com)
Yongho Hwang (yongh.hwang@samsung.com)
Jaechul Lee (galaxyra@empal.com)
Sunghyun Jo (linu@nate.com)
Jin-Mo Sung (feeljuin@gmail.com)
Jeong-Seok Yang (dasomoli@gmail.com)