Research on System Virtualization using Xen Hypervisor for ARM based secure mobile phones

Sangwon Seo
(seo.sangwon@tu-berlin.de)

Seminar "'Security in Telecommunications"' ,
Berlin University of Technology,
Korea Advanced Institute of Science and Technology

WS 2009/2010 (Version on January 14, 2010)

Abstract

Mobile phones became the prevailing application in embedded devices. As its market is getting bigger, its security is becoming an important issues. Xen as a virtualization technology could be one of its solutions with domain protection mechanism. Recently, Xen community released the interesting patch that supports mobile devices based on ARM architecture. This seminar paper deals with this patch known as *Xen-On-ARM*, as well as its background, related work, and future work.

1 Introduction

Recently, as mobile phones are connected to the Internet network, these are facing the similar security problems with personal computers and servers. Operating System (OS) level security solutions are not enough to protect from various attack codes.

Virtualization technology of Xen can be applied to the mobile phone security with a secure hypervisor called virtual machine monitor (VMM) as an alternative of security solution. However, Xen only well supports the x86 architecture before *Xen-On-ARM* [Hwang08] is announced. In this seminar paper, Xen-On-ARM denotes our topic paper and its implementation.

For this reason, Xen-On-ARM challenges the trial of porting Xen to ARM as a practical implementation. Recently, their efforts on implementation of Xen-On-ARM was released as a patch in the Xen official community. The reason for choosing ARM architecture was based on the high popularity in the mobile phone industry.

This seminar paper is organized as follows. To make understanding easier, section 2 summarizes background knowledge of Xen Hypervisor principles, and compares two related architectures such as x86 and ARM. Section 3 describes our main topic of Xen-On-ARM with related works done up to now. Section 4 presents possible future works via improvement with two candidate address management schemes. Finally this paper concludes in section 5.

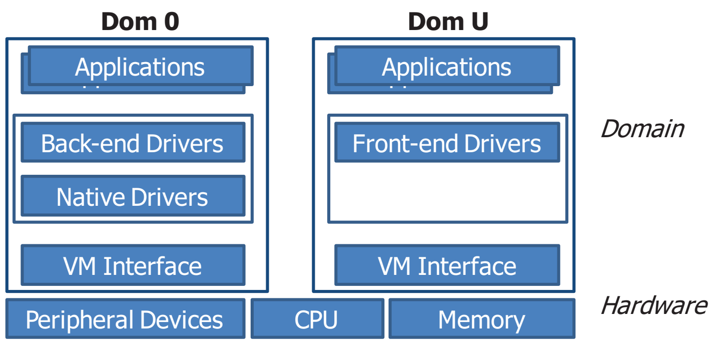

Figure 1: General Xen Architecture

2 Backgrounds

This section shortly describes Xen internals, and two hardware architectures (x86 and ARM) about general concepts, CPU modes, and cache/TLB structures.

2.1 Xen Hypervisor for virtualization

Xen Hypervisor [Barham03] is the *para-virtualization* technology based on the x86 virtual machine monitor. The para-virtualization provides virtualization with virtual machine abstraction for Guest OSes, on behalf of complete emulation of the underlying hardware like full-virtualization. It requires code modification on Guest Oses, but gives relatively high performance than full-virtualization.

The VMM logically provisions the hardware of a single physical machine to multiple virtual machines. That is, it allows multiple operating systems (Guest OSes) to share hardware resources without significant performance penalty [Barham03], as well as it provides interfaces, known as *hypercall*, to isolate Guest OSes in a secure way. Xen makes the privileged VM, called domain0 (Dom 0), for managing other Guest VMs, called domainU (Dom U). Figure 1 shows the overall architecture of Xen. VM Interfaces, such as virtual cpu and virtual network, virtualize underlying physical hardwares, and provide simple API to Guest OSes called domain. Applications can access hardware resources, such as network communication, via *event channel* which is virtually built through the negotiation between back-end driver and front-end driver. The event channel mechanism virtualizes a hardware interrupt. A virtual interrupt is queued in this channel, and then transfered into the destination domain by the domain scheduler that determines the VM scheduling algorithm.

Xen provisions VCPUs per domain, which include the information of scheduling and event channel, for the created domain. Xen utilizes hardware protection mechanisms to isolate Guest OS kernels and its applications. For example, the x86 architecture provides four processor rings (0 to 3), and thus VMM can readily isolate Guest OSes, such that VMM is assigned to ring$_0$, Guest OS kernels are assigned to ring$_1$, and applications are running in ring$_3$.

2.2 Intel x86 architecture overview for Xen

As mentioned in the previous section, Xen is based on the x86 architecture. The x86 architecture supports four distinct privilege modes known as CPU ring mechanism to protect privileged instructions from unprivileged code segments. The OS code typically runs in ring$_0$, while applications execute in ring$_3$. Using this mechanism, inter-domain protection is well provided as in hardware level. For example, sensitive instructions can be executed in privileged mode. Representative sensitive instructions are SMSW for machine status, and SGDT/SLDT for segment descriptor. Therefore, CPU protection is easily virtualized by the hypervisor residing in $ring_0$, while Guest OS kernels and their applications run at other rings. In such a way, Xen simply runs at $ring_0$, Guest OS kernels run at $ring_1$, and applications run at $ring_3$, respectively. If unprivileged processes want to access sensitive instructions, they can use hypercall interfaces provided by the hypervisor.

Regarding the virtualization for memory, it is easier if the architecture supports software managed TLB or a tagged TLB. This is because a processor can discriminate the process mode whether privileged or unprivileged with address identifier, as well as a processor does not need to flush whenever context switch of processes. Unfortunately, x86 does not have software managed TLB or tagged TLB. To minimize overhead of the flush, Xen does not switch the context when the hypervisor enters the dedicated fixed memory region assigned for the hypervisor. This resion is only accessible by $ring_0$.

2.3 ARM architecture overview for Xen

For CPU protection, ARM has typically seven kinds of modes which are *User*, *FIQ*, *IRQ*, *Supervisor*, *Abort*, *Undef*, and *System*. The User mode is uniquely unprivileged mode for user applications. The FIQ and the IRQ modes are commonly for a fast interrupt, but the IRQ is slower than the FIQ, and it normally handles external hardware interrupts. The Supervisor mode is used in OS kernel and its device drivers. The User mode can be jumped into this mode when a software interrupt (SWI) is invoked. For example, if user applications call *systemcall* provided by kernel, a software interrupt will be occurred consecutively. The Abort and the Undef are for handling fault and undefined instructions, respectively. Especially, the System mode shares the same registers with the User mode, but has high privileges. Each mode has exclusive registers, and managed by Current Program Status Register (CPSR) with bit manipulation. For instance, in the User mode, CPSR[4:0] is 10000b, and in the System mode, CPSR[4:0] is 1111b.

ARM architecture has virtually indexed virtually tagged (VIVT) cache, as well as not tagged TLB like x86. Therefore caches and TLBs should be flushed whenever switching address spaces.

2.4 Differences between x86 and ARM

The basic difference of these two architectures is whether RISC or CISC. It means that Intel x86 is based on CISC, and ARM is based on RISC. RISC is designed with a small die size, low power consumption, and compact instruction sets, especially for embedded devices. On the contrary, CISC has opposite features. In point of view for porting to Xen, x86 has four distinct privilege modes called ring, whereas ARM has only two privilege modes which are unprivileged User mode and privileged other six modes. Furthermore, ARM has not tagged TLB as well as not tagged caches, and thus it requires more optimizations to reduce caches/TLBs flush overhead.

To achieve porting to Xen, it requires following four hardware requirements [Ferstay06].

- At least two processor modes of operations.
- Routines for unprivileged applications to use privileged system services.
- Memory protection.
- Asynchronous interrupts for communication between I/O system and CPU.

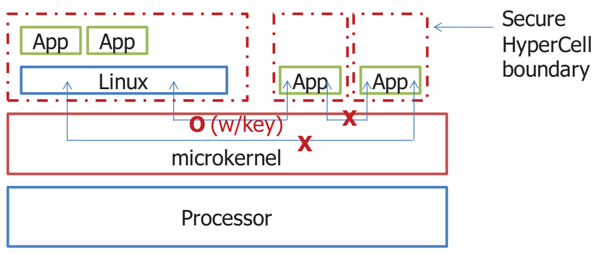

Figure 2: Capabilities used for communication control in OKL4

3 System virtualization on ARM for secure mobile devices

3.1 Related work

This section presents several related works with our topic.

3.1.1 Hypervisors for Consumer Electronics

Heiser [Heiser09] introduced the *OKL4* hypervisor running on Windows CE devices (ARM based), and compared the performance with Xen-On-ARM. OKL4 belongs to the L4 [Liedtke95] microkernel family. It targets for commercial use in embedded systems. Like Xen-On-ARM, Heiser also summarizes what functionalities are needed in hypervisor, which are *real-time capable* for bounded interrupt latency, *isolation* between subsystems, *high-efficient communication channels* with shared memory, *fast inter-VM switches* for scheduling VMs, and finally *freedom from faults* with the very small code base. Main differences with Xen-On-ARM are the real-time support, and the compact code size. Xen-On-ARM does not consider these features, especially for the size of codes which is around twice of OKL4. Typically, the size of code implies how far from faults and errors [Heiser09]. Figure 2 shows capabilities used for communication control with Secure HyperCell (SHC) [Dennis66] which is the same functionality of MAC [Lee08] implemented in Xen. This manages the shared memory and IPC in a fine-grained secure way. Unfortunately, Heiser outlines overall features of OKL4 on behalf of focusing on the porting mechanism, so it is difficult to know how it is ported in. Heiser concludes that the L4 based hypervisor is suitable to meet the requirement of virtualization with real-time support.

3.1.2 Porting the Xen Hypervisor to ARM

LeMay et al. [LeMay09] attempted the project of porting Xen to ARM emulator (QEMU) to protect cellphone viruses. This work is close to practical trial rather than scientific research. The difficulties and solutions that authors have encountered are as follows.
First is the absence of time stamp counter (TSC) access instructions in ARM, and thus they emulated TSC and modified corresponding functions.
Second is the lack of SMP supports in ARM v5 that authors targeted (but ARM v6 is supported). This problem is overcome by modifying the kernel source forcing non-SMP in the naive angle.
Third is about MMU. The hardware architecture is quite different from that of x86, and thus authors replaced the dependable codes under the guidance of ARM Linux Kernel.
Last is the difference of CPU protection rings between ARM and x86. Generally, ARM cores provides only two kinds of modes which are privileged mode and unprivileged mode, but x86 supports 4 rings.

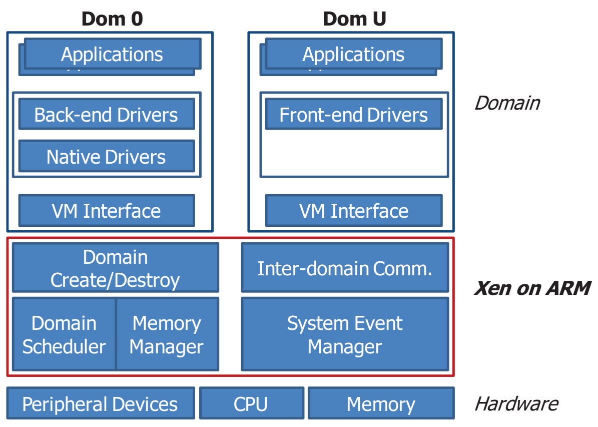

Figure 3: Xen architecture with Xen-On-ARM

3.1.3 Real-time OS Virtualization for Xen-ARM

Miri et al. [Park09] introduced real-time support by extending the Xen-On-ARM implementation. Xen-On-ARM creates a pagetable for each Guest OSes when constructing a Guest domain in order to support virtual memory systems. However, many real-time OSes use only physical memory itself. For this reason, it is possible to modify the memory mapping mechanism to disable virtual memory functionality, since many real-time OSes usually do not use it for high performance. More advanced way is to map the entire size of SDRAM, but it was remained for the future work.

Some real-time algorithms such as EDF (Early Deadline First) need to know the precise physical time. In practice, this clock precision is usually accomplished by a timer interrupt. To properly use real-time scheduling algorithm (to meet the deadline), they modified the tick timer handler of RT Guest to minimize the latency. Finally, they evaluated the latency with two real-time Oses, uC/OS II and mini-os, and showed precise physical time for the guaranteed service of real-time tasks.

3.2 Main Topic: System virtualization on ARM for secure mobile devices (Xen-On-ARM)

Recently, high-speed network connected mobile phones, which are gradually approaching to the personal computer environment, are confronted to security problems. For this reason, Xen-On-ARM proposes an OS level protection solution beyond the address-space separation. This utilizes virtualization for Guest OSes isolation under VMM.

In the paper, authors mainly focused on implementation issues for porting to ARM, since Xen does not support ARM processor. Xen-On-ARM is positioned between underlying physical layer and VM Interface in the Xen architecture as shown in Figure 3.

3.2.1 What are the issues in Virtualization for Embedded Devices?

This section discusses implementation issues and difficulties when porting to ARM.

- *ARM CPU has only one unprivileged mode*
  Typically, ARM provides seven modes, six privileged modes (FIQ, IRQ, Supervisor, Abort, Undef, System) and one unprivileged mode (User). For the hardware level separation between the VMM and Guest OSes, Guest OS kernels and their applications can not help sharing one User mode of ARM, whereas VMM should be running on one of privileged modes. That is, Guest OS kernels and applications

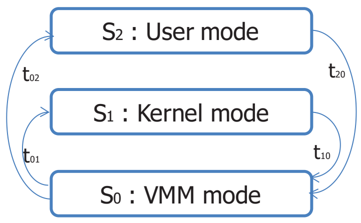

Figure 4: VCPU mode transitions

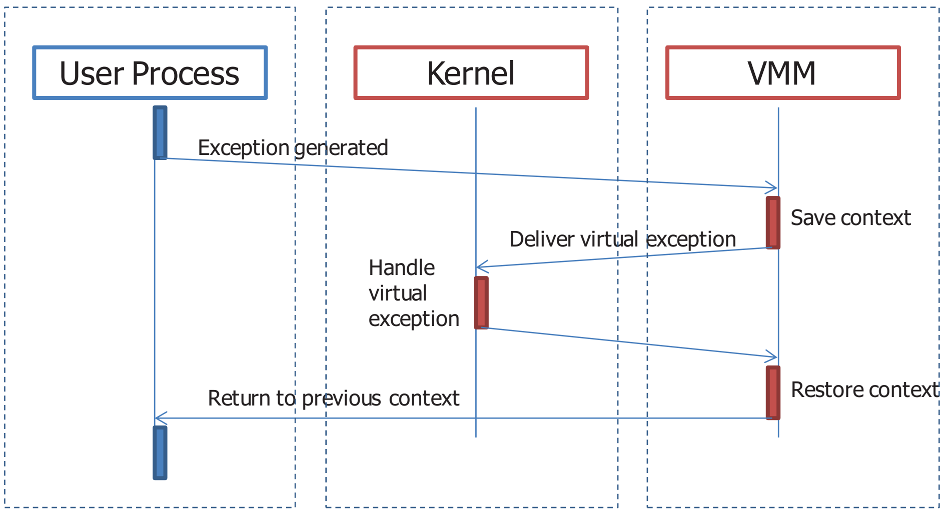

Figure 5: Exception handling

should run together at User mode. For this reason, it requires protection techniques between the Guest OS and applications.
Simply separating the address spaces of applications will increase caches/TLBs flushing overheads, because ARM v4/v5 architecture does not support tagged caches and TLBs. Whenever context switching, all caches and TLBs should be fully flushed providing high address space switching costs.

3.2.2 CPU Virtualization

1. *Virtual Privilege Modes: VCPU mode transition*
   As mentioned in section 3.2.1, the Guest OS and applications run in the same User mode under current implementations of Xen that is optimized only for the x86 architecture. Xen-On-ARM needs two logical modes which are *Kernel* and *User mode*. Xen-On-ARM proposes these two logical modes to protect the kernel from applications in the physically same mode in ARM.
   In brief, there are three modes in Xen-On-ARM, which are the VMM mode and two logical modes (Kernel, User). Xen-On-ARM is responsible for switching between Kernel mode and User mode. The authors call this mode transition to *VCPU mode transition*. Figure 4 simply illustrates the VCPU mode transition diagram. When one of interrupts, faults, aborts and hypercalls occurs, t_{10} transition should be activated. t_{20} transition is on interrupts, faults, aborts and system calls. t_{01} transition happens in upcalls or returns from the exception, and t_{02} transition is valid when the returning from exceptions.
   In User mode, direct access of fourteen all sensitive instructions used in ARM, such

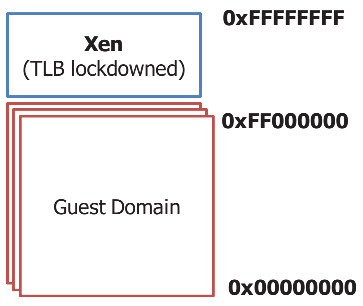

Figure 6: Memory internals with lockdown TLB

as MRS and MSR, is strictly forbidden. It is accessible only through the *hypercalls* privided by Xen-On-ARM.

2. *Exception Handling*
As depicted in Figure 4, if exceptions occur, the VCPU transition is entered to the VMM mode. The status of *SPSR* is saved in the virtual SPSR (VSPSR) for restoring later, and stack pointer is saved in the virtual stack pointer (VSP) register that Xen-On-ARM provides. Consecutively, *upcall* from Xen notifies exceptions to the Kernel mode with virtual exception event and VSPSR. VSPSR includes the information of the last running virtual process mode. Afterwards, the kernel exception handler is invoked for handling the corresponding exception. Finally, the context is restored to the VMM, and the VMM returns the context to the previous context of user process. These procedures are represented in Figure 5.

3.2.3 Memory Virtualization

The goals in memory virtualization are as follows.

- VMM memory region protection from Guest OS kernels/user applications.
- Guest OS kernel memory region protection from user applications.
- User applications memory region protection from each other.

For ensuring the first requirement in above, Xen-On-ARM provides hypercalls for memory update/create pagetable, meaning that only Xen can access memory pagetable. However, it is not enough to guarantee the rest of the requirements, since the typical paging hardware does not know whether user applications or OS kernel running in.

To achieve the remaining requirements as well as the first one, Xen-On-ARM adopts *domain protection mechanism* given by ARM whose architecture provides 16 kinds of domains. The authors only use three (D_0 - D_2) for virtualization purpose, D_0 is for the VMM, D_1 is for Guest OS kernels, and D_2 is for user applications. Each of the domain permissions is controlled by bit adjustment in the Domain Access Control Register (DACR). For example, at the user application mode, DACR of D_0 (for VMM) and D_1 (for Guest OS kernels) are set with *no access*.

For optimization of cache/TLB flushing overhead as mentioned in 3.2.1, Xen-On-ARM proposes following optimization strategy.

- *Motivation*
A flush is not necessary for virtual privilege mode inside the same VM. This means it is only required when the VMM switches domain or Guest OS kernel switches user processes.
- *Proposed strategy*
The authors adopt two lockdown TLB entries (among eight entries) to keep memory area that anyone wants to pin without the flush.

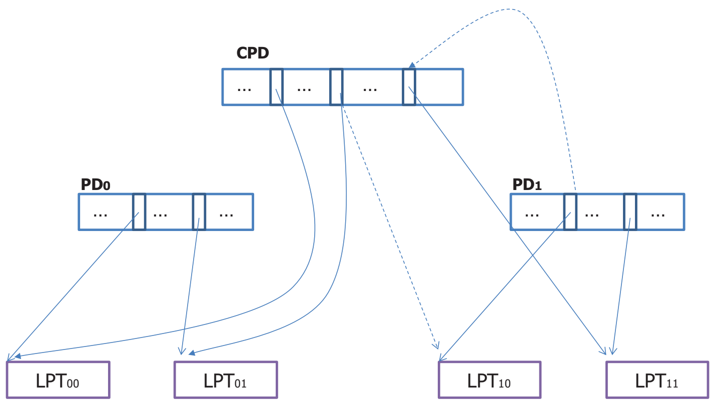

Figure 7: Caching page directory internal

3.2.4 Evaluation

Xen-On-ARM used two kinds of Benchmarks, Micro (LMBENCH) and Macro (Image processing with Qtopia PDA edition), on modified Xen 3.0.2 (23401 lines altered) and Linux 2.6.11 (4518 lines altered). Additionally conventional Xen tools such as XenBus and XenStore are working properly solely with cross compilation.

- *Micro Benchmark*
  LMBENCH is used to compare basic system operation performance and latency between native Linux and Xen-On-ARM. With this evaluation, they gain the result that performance is more and less similar with both of them, and latencies are not higher than double of native Linux due to the additional hypercall overhead.
  The authors also have evaluated context switching latency with four types of configurations which are native Linux, Xen-On-ARM with TLB lockdown, Xen-On-ARM without TLB lockdown, and Xen-On-ARM with legacy pagetable separation scheme. In this evaluation, compared to native Linux, Xen-On-ARM gives a performance penalty only with 50 microseconds additional delay. The TLB lockdown scheme also shows slightly enhanced performance when the context size is larger than 8 Kbytes.
- *Macro Benchmark*
  For real workload, the authors compared performance factors with image processing on Qtopia PDA between Xen-On-ARM and native Linux. This result shows similar performance between Xen-On-ARM and Native Linux with respect to UI loading time, saving time, encoding time, and decoding time.

4 Future works

This section discusses two outstanding future works which can be applied to the Xen-On-ARM. The first is the *Fast Address-Space Switching* (FASS) [Wiggins03]. The FASS significantly minimizes context switching overhead for caches/TLBs without address-space tags. The second is the *VMM with Context Sensitive Page Mappings* (CSPM) [Rosenblum08]. CSPM can consolidate the various security threats.

4.1 Fast Address-Space Switching (FASS)

The ARM architecture has virtually-indexed and virtually-tagged L1 instruction and data caches, and non-tagged TLBs. Therefore, it is required to flush the entry of caches

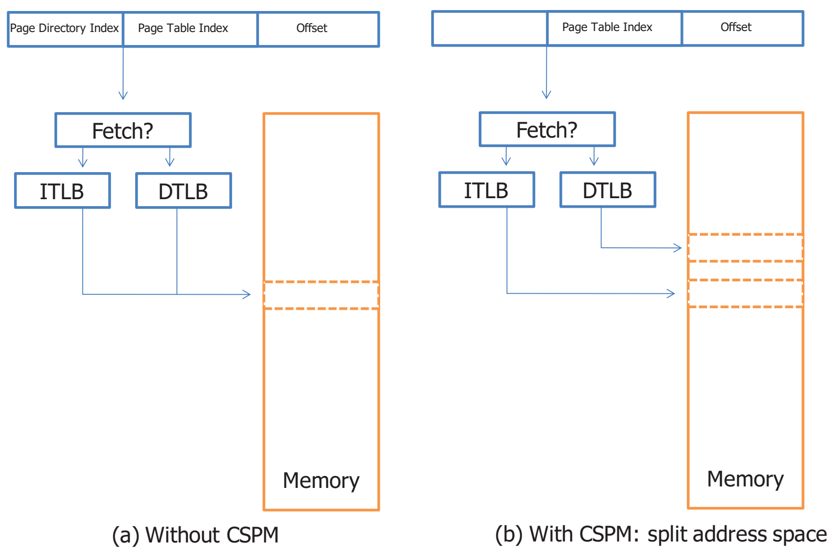

Figure 8: An address space separation with TLBs

and TLBs whenever the context is switched. IPC based microkernels such as L4 are more sensitive to this context switching overhead.

For reducing this penalty, ARM provides several MMU features which are *domains* and *PID relocation*. Domains identify the domain-level mapping, instead of process identifier (It is very limited to small number of domains). Using domains, collision detection is possible [Wiggins00]. PID relocation allows the OS to be able to assign maximum 64 processes without overlapping addresses. To avoid address collision with PID relocated address, typically it is recommended to share the memory region above the upper 2GB of address space.

This approach is working on the two-level pagetable of ARM as shown in Figure 7. With a simple trick, each page entry points a data structure called *caching page directory* (CPD), instead of pointing the new process. The CPD includes information about leaf pagetables (LPT) of several processes. That is, CPD is a cache of Page Directory (PD) entries with domain ID tags. Figure 7 represents this idea. In summary, flushes may happen only following four situations.

- A process attempts to map something into the PID-relocation area (32MB to 2GB).
- A process attempts to map something into the shared area, then it collides with another process.
- Not enough address space in shared area.
- The kernel is running out of domains.

4.2 VMM with Context Sensitive Page Mappings (VMM-CSPM)

VMM-CSPM is implemented with the x86 architecture. The paging hardware on the x86 translates from virtual linear address to physical address via a pagetable. Each process is pointed with page directory by the pagetable base register (i.e., cr3 for x86). To reduce the translation cost, normally recently-used pagetables are cached in a translation lookaside buffer (TLB).

The VMM-CSPM suggests the extension to the Xen VMM for supporting context sensitive page mappings without modifying Guest OSes with minimal overhead when mapping the page. This scheme splits the view of memory architecture as to *Harvard memory* *architecture* via separating the manipulation of instruction and data TLBs. ARM core is hopefully expected that it is often implemented as Harvard architecture, and thus the VMM-CSPM can be readily applied to the ARM than the x86 architecture. Separated ITLB and DTLB entries usually mapped to a specific identical virtual address known as a symmetry state. By manipulating the processor state, this symmetry can be broken, so as to virtual address translation points the actual address of whether ITLB or DTLB indicated according to the context.

Imagine that attacker tries to modify targeting running binary in the VMM-CSPM system, and then binary looks successfully modified without prevention. However, actually attacker is deceived by separated TLBs. That is, the code attacker modified is not on the same location attacker want to modify. This is the key idea behind the VMM-CSPM. Figure 8 illustrates this simple idea. It splits the address space to desynchronized instruction and data TLBs. Figure 8 (b) is after application of CSPM. This mechanism is not directly supported by any architectures including x86, but depends on the ability to be able to manipulate the content of instruction and data TLBs.

5 Conclusion

This seminar paper summarizes Xen-On-ARM which is the first successful porting trial of Xen to ARM, and describes related works, and future works. Through the Xen hypervisor in mobile devices, Guest OS kernels and applications can be readily protected each other under the domain level isolation. As a further consideration, for minimizing performance penalty, Xen-On-ARM suggests FASS scheme. For maximizing security from malicious codes, VMM-SCPM scheme is recommended. Additionally, real-time support can be applied to Xen-On-ARM as described in related works. It may be good choice for realtime mobile applications.

References

[Hwang_08] J-Y Hwang, S-B Suh: *Xen-On-ARM: System Virtualization using Xen Hypervisor for ARM-based Secure Mobile Phones*; IEEE CCNC (2008).

[Barham_03] P. Barham, B. Dragovic: *Xen and the art of virtualization*; Proceedings of the ACM Symposium on Operating Systems Principles (2003).

[Ferstay_06] D. R. Ferstay: *Fast secure virtualization for the arm platform*; Master thesis, Department of Computer Science, University of British Columbia (2006).

[Wiggins_03] A. Wiggins, H. Tuch,: *Implementation of fast address-space switching and tlb sharing on the strong arm processor*; Advances in Computer Systems Architecture, pp. 352.364, (2003).

[Rosenblum_08] N. E. Rosenblum, G. Cooksey: *Virtual Machine-Provided Context Sensitive Page Mappings*; VEE 2008 (2008).

[Oorschot_05] V. Oorschot, P. C., Somayaji: *Hardware-assisted circumvention of self-hashing software tamper resistance*; EEE Trans. Dependable Secur. Comput. 2, 2, 82.92 (2005).

[Heiser_09] Gernot Heiser: *Hypervisors for Consumer Electronics*; IEEE CCNC (2009).

[Liedtke_95] J. Liedtke: *On mu-kernel construction*; in 15th SOSP, Copper Mountain, CO, USA, pp. 237-250 (1995).

[Park_09] Miri Park: *Real-time Operating System Virtualization for Xen-Arm*; The 4th International Symposium on Embedded Technology, Daegu, Korea (2009)

[LeMay_09] M. LeMay, D. Jun: *Porting the Xen Hypervisor to ARM*; Technical Report in UIUC (2009).

[Joshi_05] A. Joshi, S. T. King: *Detecting Past and Present Intrusions through Vulnerability Specific Predicates*; In Proceedings of the 2005 Symposium on Operating Systems Principles (SOSP), October (2005).

[Garfinkel_03] T. Garfinkel, B. Pfaff,: *Terra: A Virtual Machine-Based Platform for Trusted Computing*; In Proceedings of the 2003 Symposium on Operating Systems Principles (SOSP), October (2003).

[Garfinkel_03] T. Garfinkel and M. Rosenblum: *A Virtual Machine Introspection Based Architecture for Intrusion Detection*; In Proceedings of the 2003 Network and Distributed System Security Symposium (NDSS), February (2003).

[Wiggins_00] Wiggins, A., Heiser: *Fast address-space switching on the StrongARM SA-1100 processor*; In Proceedings of the 5th Australasian Computer Architecture Conference (ACAC), Canberra, Australia, IEEE CS Press 97.104 (2000).

[Dennis_96] J. B. Dennis and E. C. Van Horn: *Programming semantics for multiprogrammed computations*; CACM, vol. 9, pp. 143.155 (1966).

[Lee_08] S.-M. Lee, S. bum Suh, B. Jeong, and S. Mo: *A multi-layer mandatory access control mechanism for mobile devices based on virtualization*; in 5th IEEE Consumer Comm. Networking Conf., Las Vegas, NV, USA, Jan, pp. 251.256 (2008).