

Incident-driven memory snapshot for full-virtualized OS using interruptive debugging techniques

Ruo Ando, Youki Kadobayashi and Youichi Shinoda
National Institute of Information and Communication Technology
4-2-1 Nukui-Kitamachi, Koganei, Tokyo 184-8795 Japan
ruo@nict.go.jp

Abstract

Memory forensics is growing concern. For effective evidence retrieval, it is important to take snapshot timely. With proper modification of guest OS, VMM is powerful tool for timely snapshot. In this paper, we propose an incident-driven memory snapshot for full-virtualized OS using interruptive debugging techniques. We modify debug register handler to invoke snapshot facility of VMM. Software interrupt or signal are generated in register handler. Then, we can take snapshot asynchronously when debug register is changed. On guest OS, we apply three kinds of interruptive debugging techniques: driver supplied callback function, DLL injection. IDT (interrupt descriptor table) is modified by driver supplied callback function, which makes it possible to cope with vulnerability exploitation. DLL injection is applied to insert security check function into a resource access function. Proposed system is implemented XEN virtual machine monitor and KVM (Kernel Virtual machine).

1. Introduction

1.1. Incident-driven memory

Memory forensics is growing concern. Insider attack and information leak have become a serious problem. VMM provides powerful facilities to access all states of guest VM: CPU context, memory and other block devices. Timely snapshot is important for evidence retrieval. If snapshot is taken after incident has happened, usually much of evidence has been lost. In this paper we propose an incident-driven memory snapshot for full-virtualized OS using interruptive debugging techniques. With proper modification of VMM, we can obtain memory snapshot just in time when the incident happens. We modify debug register handler for this purpose. When the incident has been occurred on guest OS, it is notified by changing debug register. An interruption generator is inserted into register handler of hypervisor. Then, host OS receives notification and takes snapshot. In following section, we discuss asynchronous memory forensics. In section 2, modification of VMM is presented. Improving guest Windows(R) OS is discussed in section 3.

1.2. Towards an asynchronous memory forensics

Figure 1 show the concept of our memory snapshot and forensics on VMM. First, a incident is detected in guest OS kernel. Second, the incident is notified to hypervisor as change of debug register state. At this point, proposed system takes snapshot using facilities of hypervisor. Then, Host OS analyzes memory dump and takes some reaction for the incident. With proper modification, VMM can makes these lines (detect, dump, analysis and action) multiplexed. Towards an asynchronous memory forensics, snapshot need to be taken without suspension or need to be taken in short time. In section 2, we present the modification

of VMM (hypervisor and host OS). In section 3, we present how to improve guest OS for incident-driven snapshot.

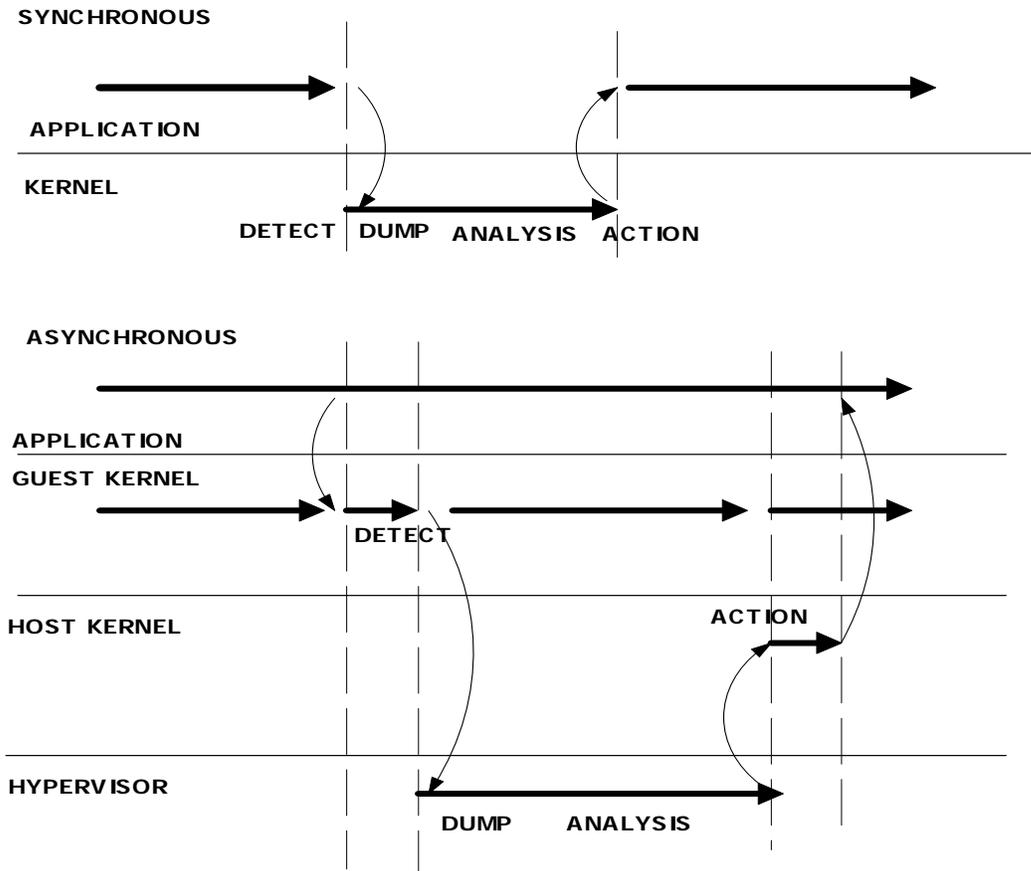


Figure1. Asynchronous memory snapshot and forensics. In VMM, snapshot modules is outside the guest OS. With proper modification of VMM, memory snapshot and analysis is executed in VMM (hypervisor and host OS).These lines can be multiplexed

2. Modification of VMM

In this section discuss show the modification of XEN[3] and KVM[4]. Once the incident is detected on guest OS, the value of special register (DR/MSR) is changed. The context of virtualized CPU is stored in hypervisor stack. VMM can detect the incident of guest OS when domain context is switched because CPU context including the state of DR/MSR register is changed. Then, proposed system sends asynchronous notification to host OS.

Figure 2 shows an implementation of proposed system in XEN. For asynchronous notification, software interruption is applied. Once the incident is detected in guest OS, special registers (DR/MSR) is changed (vector [1]). Then, the register handler caught this change (vector [2]) which is transferred to the host OS by software interruption generator by global pirq (vector [3]). When the host OS caught the pirq, memory snapshot is taken using facilities of QEMU I/O.

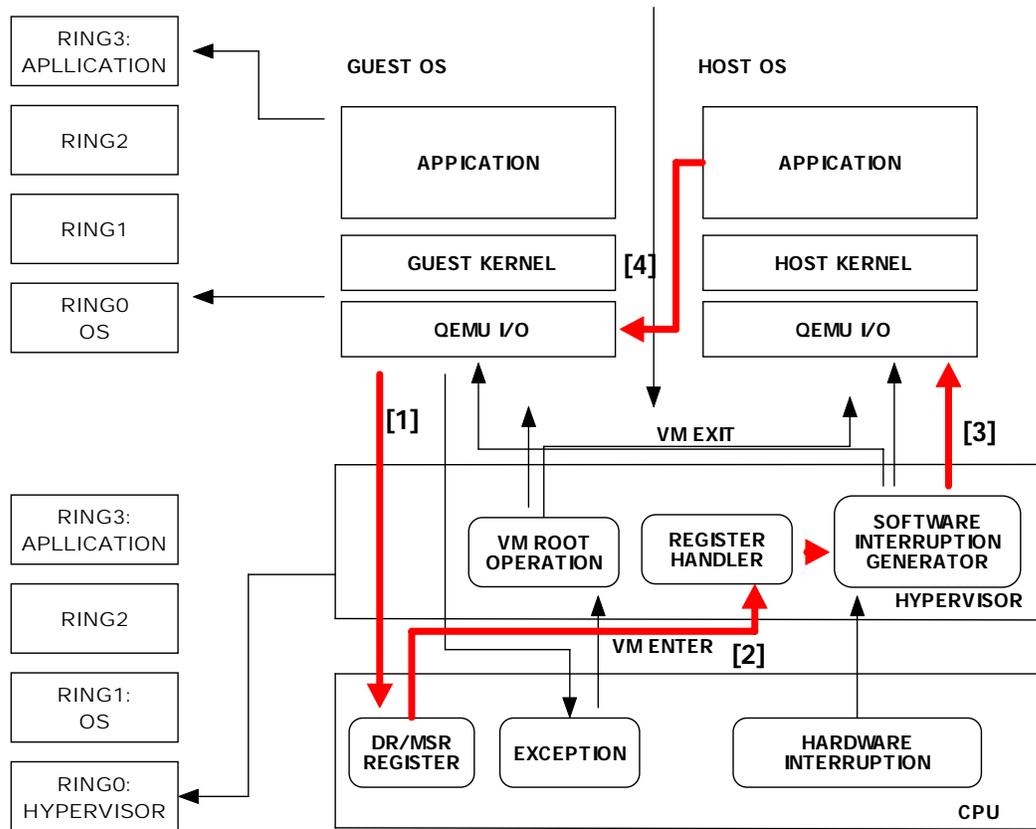


Figure2. Proposed system implemented on XEN. Incident on guest OS is tracked as change of debug register[1]. Register handler caught this change[2]. Then, software interruption as notification is transferred to host OS[3].

Figure 3 shows an implementation of proposed system in KVM (Kernel Virtual Machine). KVM makes Linux as hypervisor. In implementation of KVM, a simple user defined signal is applied for the asynchronous notification. When the incident is detected by guest OS, the value of special registers is changed (vector [1]). When the system control is moved to VM root operation, the change is caught by register handler. Then, user defined signal is sent to QEMU modules of KVM by control application or directly from kernel (vector [3][4][5]). Finally, signal handler invokes memory snapshot facilities using QEMU I/O module.

3. Modification of Windows guest OS

In this section we discuss the modification of host Windows(R) OS for changing special registers (DR/MSR). We apply IDT modification using process structure routine for vulnerability exploitation. For malicious resource access, we apply DLL injection.

3.1. Improving exception handler

Figure 4 is the brief illustration of improved exception handler of proposed system. Improving exception handler is divided into two steps: inserting software break point and modification of IDT(R). At first, PsLoadImageNotiryRoutine is applied to inspect whether target executable is loaded. Then, software breakpoint is inserted below entry point. Second,

exception handler in IDT is modified. Modified routine traces EIP using MSR (model specification register).

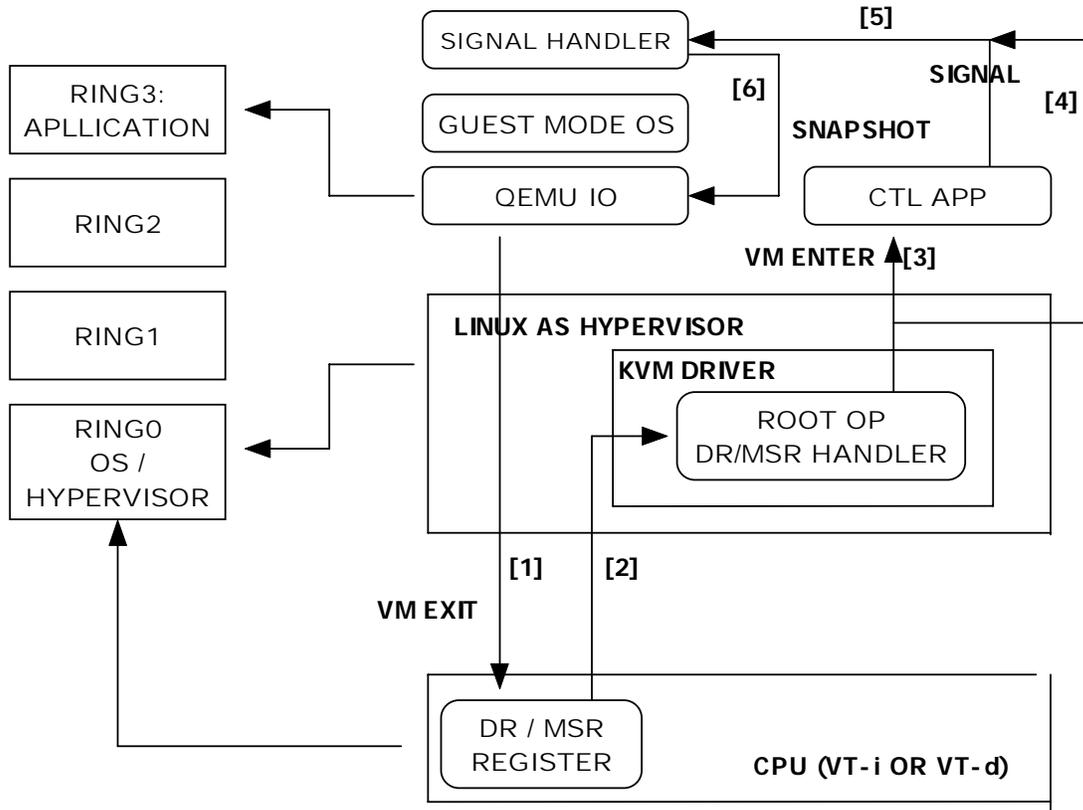


Figure3. Proposed system implemented on KVM. When an incident is detected, guest OS changes debug register. The change is caught in KVM module. Then, signal is generated and sent guest OS to take snapshot.

3.1.1 Driver-supplied callback function

To implement proposed system, we selected driver-based callback function, which is invoked whenever an image is loaded for execution. Driver-based callback function is used for inspect whether target executable is loaded. Highest-level system profiling drivers can call PsSetImageNotifyRoutine to set up their load-image notify routines. This could be declared as follows.

```
void LoadImageNotifyRoutine (
    PUNICODE_STRING FullImageName,
    HANDLE ProcessId,
    PIMAGE_INFO ImageInfo );
```

Once the driver's callback has been registered, operating system calls the callback function whenever an executable image is mapped into virtual memory.

When LoadImageNotifyRoutine is called, the input FullImageName points to a buffered Unicode identifying the executable image file.

3.1.2 Debug register

IA-32 processors family provides MSR(model specific registers) for the purpose of recording taken branches, interrupts and exception. In this paper we focus on last branch interruptions / exceptions flag to save and search the EIP(32 bit instructional pointer). EIP means return address. The most recent taken branches, interrupts and exception are stored in the last branch record stack MSRs. The branch records inform us of branch-FROM and branch-TO instruction address. Concerning F6 family processor, the five kinds of MSR, debugCtlMSR, LastBranchToIP, LastBranchFromIP, LastExceptionToIP and LastExecutionFromIP and available. It is possible to set break points on branches, interrupts and exception and execute single step debugging through these registers. These registers can be used to collect last branch records, to set breakpoints on branches, interrupts, exceptions and to single step from on branch to the next.

3.2 Dll injection

We apply DLL injection for inspecting illegal resource access of malicious process. DLL injection is debugging technology to hook API call of target process. Windows executable applies some functions from DLL such as kernel32.dll. Executable has import table to use the linked DLL. This table is called as import section. Among some techniques of DLL injection, modifying import table is useful because this technique is CPU-architecture independent. Figure 5 show the modification of import table. Address of function A on left side is changed to the address of inserted function on right side. In code table, some original functions are appended to executable. Modified address is pointed to code of inserted function. By doing this, when the function A is invoked, the inserted function is executed. In proposed system, the inserted function changes special registers (DR/MSR) to notify the events to VMM and control domain.

3.2.1 Search and change IAT

After the module to be modified is determined, we need to change the address in IAT (Import Address Table) to our inserted DLL. ReplaceIATEntryInAllMods is available for changing the address of module.

In this ReplaceIATEntryModues, ReplaceIATEntryInOneMod is invoked to get the address modules in import section table. Once the address of modules we try to insert our DLL, WriteProcessMemory is available for change the IAT.

3.2.2 Injecting DLL for all processes

To inject DLL for all running processes, SetWindowsHookEx is useful. For global hook, invoking SetWindowsHookEx maps DLL for all processes.

```
Inject.dll  
call SetWindowsHookEx  
Function to insert  
ReplaceIATEntryInAllMods  
ReplaceIATEntryINOneMod
```

To use this API, avoiding hook for Inject.dll itself is required. In the case that the address of function to insert need to be hidden, LoadLibrary and GetProcAddress is hooked because these APIs can search the address of inserted function.

4. Conclusions

Memory forensics is growing concern. For effective evidence retrieval, it is important to take snapshot timely. With proper modification of guest OS, VMM is powerful tool for timely snapshot. In this paper, we propose an incident-driven memory snapshot for full-virtualized OS using interruptive debugging techniques. We modify debug register handler to invoke snapshot facility of VMM. Software interrupt or signal is generated in register handler. Then, we can take snapshot asynchronously when debug register is changed. On guest OS, we apply three kinds of interruptive debugging techniques: driver supplied callback function, DLL injection. IDT (interruption descriptor table) is modified by driver supplied callback function, which makes it possible to cope with vulnerability exploitation. DLL injection is applied to insert security check function into a resource access function. Proposed system is implemented XEN virtual machine monitor and KVM (Kernel Virtual machine).

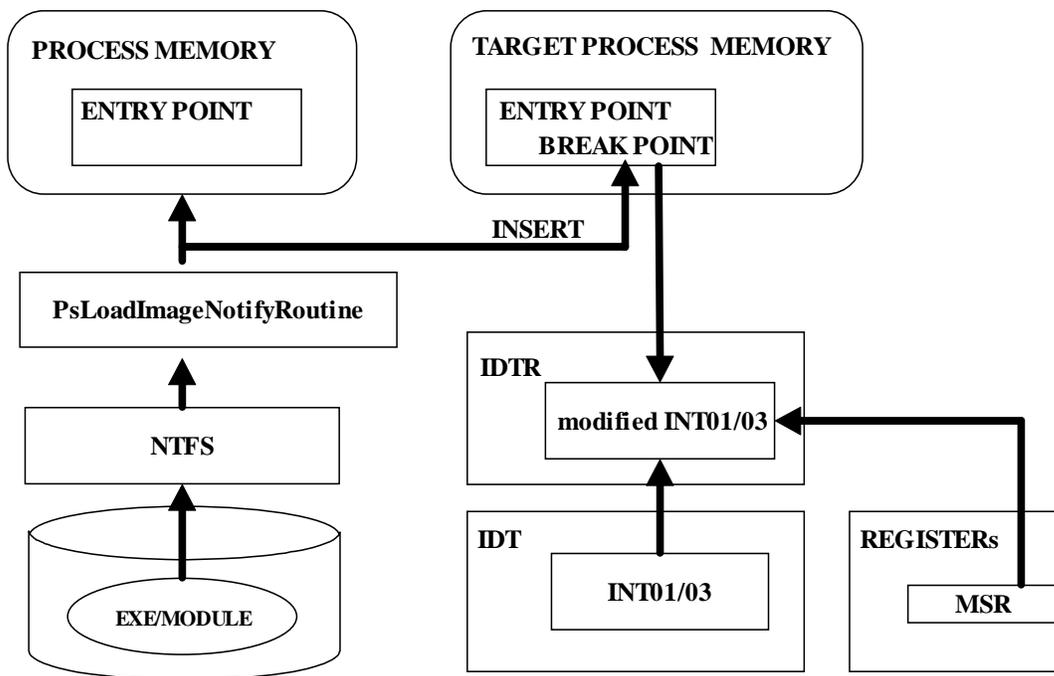


Figure4. Improving exception handler. PsLoadImageNotifyRointe is applied to check if target executable is loaded. Then, software break point is inserted. Also, IDT(R) is modified to trace EIP using MSR.

References

- [1] Paul A. Karger, Mary Ellen Zurko, Douglas W. Bonin, Andrew H. Mason, Clifford E. Kahn, "A Retrospective on the VAX VMM Security Kernel", IEEE Trans. Software Eng. 17(11): 1147-1165, 1991
- [2] XEN virtual machine monitor, <http://www.cl.cam.ac.uk/Research/>.
- [3] Kernal Virtual Machine available at: <http://sourceforge.net/projects/kvm>.
- [4] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In Proceedings of the 19th Symposium on Operating System Principles(SOSP 2003), Bolton Landing, NY, October 2003.

- [5] A Virtual Machine Introspection Based Architecture for Intrusion Detection Tal Garfinkel and Mendel Rosenblum In the Internet Society's 2003 Symposium on Network and Distributed System Security (NDSS), pages 191--206, February 2003.
- [6] George W. Dunlap, Samuel T. King, Sukru Cinar, Murtaza Basrai, and Peter M. Chen. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay. In Proceedings of the 2002 Symposium on Operating Systems Design and Implementation (OSDI 2002), Boston, MA, December 2002.
- [7] Samuel T. King, Peter M. Chen, Yi-Min Wang, Chad Verbowski, Helen J. Wang, Jacob R. Lorch, "SubVirt: Implementing malware with virtual machines", in Proc. IEEE Symp. on Security and Privacy (the Oakland Conference), May 2006.

