DEOS-FY2013-QK-01E

QEMU-KVM + D-System Monitor Setup Manual

Version E1.0

2013/07/15

Edited by DEOS R&D Center

DEOS Project

JST-CREST

Research Area
"Dependable Operating Systems for Embedded Systems Aiming at Practical Applications"

Japan Science and Technology Agency

Contents

1. D-Visor and D-System Monitor

D-System Monitor watches OSs from the outside and monitors OSs for attacks or tampering. One way to monitor OSs from the outside is constructing special hardware to support the monitoring of OSs running on that hardware. But, in the DEOS Project, we integrate some mechanisms for monitoring with a Virtual Machine Monitor (VMM) and with an OS in the VMM. This VMM is called D-Visor.

The DEOS Project developed D-Visor86 to run in an x86 multi core CPU and D-System Monitor to run in D-Visor86. For details, please refer to "D-Visro86 + D-System Monitor Setup Manual" (in Japanese). However, the D-Visor86 only runs in a limited range of hardware. Therefore, in order to extend the range of hardware supporting D-System Monitor, we tried to modify the QEMU-KVM to incorporate the D-System Monitor and to use the QEMU-KVM as D-Visor. This is the QEMU-KVM integrated with D-System Monitor (referred to as QEMU-KVM+D-System Monitor) described in this document.

Attack and tampering on OSs is frequent, and monitoring mechanisms dealing with each threat need to be implemented in the D-System Monitor. At this time, the following three monitoring mechanisms are available for QEMU-KVM+D-System Monitor:

1. FoxyKBD
   Generates a large amount of simulated key inputs and monitors OS behavior. Searches for some relation between keyboard inputs and network I/O requests, because this is suspicious behavior.
2. RootkitLibra
   Detects invalid file meta data returned from a guest OS, by comparing file meta data acquired from the guest OS with the data in NFS packets on files under NFS-mounted directories.
3. Waseda LMS (Lightweight Monitoring Service)
   Detects inconsistencies in process data by comparing data in task list with data in run queue in the guest OS.

All of these have the same monitoring mechanisms as those in the D-System Monitor of D-Visor86. Some parts of their source codes are modified and integrated with QEMU-KVM+D-System Monitor.

2. System Requirements

QEMU-KVM+D-System Monitor runs in the following environments:

- hardware
  PC with x86_64 CPU supporting Intel VT. At this time, AMD-V is not supported.
- software
  Host OS: Ubuntu 12.04 LTS x86_64desktop
  Guest OS: Ubuntu 12.04 LTS x86_64 (Uni-Processor kernel)

3. Procedures to Setup Environment

In order to setup an environment to run QEMU-KVM+D-System Monitor, the following procedures are required:

1. Replace the kernel of a guest OS with a uni-processor kernel.
2. Modify the QEMU-KVM in the host OS.
3. Modify the KVM kernel module in the host OS.

Furthermore, the following procedures are required to demonstrate how well these three mechanisms detect abnormal behavior.

4. Apply patches to the kernel of the guest OS to make abnormal behavior for demonstrations.

5. Prepare kernel modules.
6. Prepare a program for a user monitoring mechanism.
7. Prepare a GUI program.

From the next chapter, details of the procedures are described. In this document, it is assumed that the host OS is Ubuntu 12.04 LTS x86_64 with KVM installed and that all operations such as "build" are executed in the host OS. In addition, X Window environment is required in the host OS to make a GUI program for demonstration.

The qemu-kvm-dsysmon-0.1.0.tar.gz file includes the following files. Programs are either source codes or patches.

name	contents
COPYING	GPL version 2 license
COPYRIGHT	copyright
GUI/	GUI program
README	readme file
file-rootkit/	kernel module to generate invalid file meta data for demonstration
guest-os/	patches for guest OS
host-os/	patches for KVM kernel module of host OS
process-rootkit/	kernel module to generate invalid process information for demonstration
qemu-kvm/	patches to integrate D-System Monitor into QEMU-KVM
rpld-receiver/	program to receive keyboard input data
rpld.conf	configuration file for the rpld program
rtkl-collec.kvm/	user program for RootkitLibra. It runs in the guest OS
start-qemu.sh	shell script program to start QEMU-KVM+D-System Monitor
ttyrpld-2.60/	patches for an open source program recording keyboard input

Before setting up, please create a KVM image for the guest OS by using the usual kvm-img command and installing Ubuntu 12.04 LTS x86_64. From the next chapter onward, we replace kernels, installing kernel modules in the OS.

4. Uni-Processor Kernel for the Guest OS

At this time, the only supported guest OS is a uni-processor kernel of Linux x86_64. Please get the Linux kernel source code of Ubuntu 12.04 LTS x86_64 and build it for a uni-processor, not for SMP. The built OS must be used as a kernel of the guest OS. Please start the guest OS in the KVM and install the built OS after transferring it from the host OS to the guest OS. As a reference, the .config file used at the DEOS R&D Center is included as guest-os/.config in the published tar.gz file.

As described in Chapter 8, in order to demonstrate how abnormal behavior is detected, the kernel of the guest OS must be further modified. If the kernel build is done with the modifications described in Chapter 8 in addition to the modifications in this chapter, the total time for build and installation may be reduced.

Virtual addresses of the following three symbols in the kernel are required later. Please look at the Sysetm.map file of the guest OS.

- init_task
- runqueues

init_level4_pgt

5. Modification of QEMU-KVM

In this chapter, QEMU-KVM is modified to integrate D-System Monitor.

5-1. Installation of Necessary Packages

Please install necessary packages in the host OS.

```
$ sudo apt-get install zlib1g-dev
$ sudo apt-get install libglib2.0-dev
$ sudo apt-get install libsdl1.2-dev
$ sudo apt-get install uml-utilities
```

5-2. Applying Patch

Please get the source code of qemu-kvm-1.0+noroms and apply the patch file (qemu-kvm/qemu-kvm-1.0+noroms.patch).

```
$ mkdir work
$ cd work
$ apt-get source qemu-kvm
$ patch –p0 < /path/to/patch/file
```

5-3. Modification of Addresses

Three constants in the source code must be modified using the virtual address values in the System.map of the uni-processor kernel of the guest OS built in Chapter 4.

Please modify the following two lines in the qemu-kvm-1.0+noroms/dsysmon/lms.c file.

```
#define INIT_TASK_DEFAULT       0xffffffff81a16300UL    (address of init_task)
#define PER_CPU_RQ_DEFAULT      0xffffffff81a2d420UL    (address of runqueues)
```

Please modify the following line in the qemu-kvm-1.0+noroms/dsysmon/dsm_lib.c file.

```
#define INIT_LEVEL4_PGT     0xffffffff81a0c000UL      (address of init_level4_pgt)
```

Instead of modifying constants in the source file, the values can be specified when starting D-System Monitor in QEMU-KVM.

5-4. Building QEMU-KVM

Before building QEMU-KVM, please modify two lines of QEMU_CFGAGS of lms_kern.o in the qemu-kvm-1.0+noroms/Makefile.objs file, so that the QEMU_CFGAGS refers to the directories included in the source code of uni-processor kernel built in Chapter 4.

Please create the obj directory and build QEMU-KVM as follows:

```
$ mkdir obj
$ cd obj
$ ../qemu-kvm-1.0+noroms/configure --target-list=x86_64-softmmu
$ make
```

When the make command finishes, the executable file qemu-system-x86_64 is created under the obj/x86_64-softmmu directory.

5-5. Creating Symbolic Links

In order to run the qemu-system-x86_64 built in Section 5-4, several symbolic links must be created under the obj/pc-bios directory.

```
$ cd pc-bios
$ ln -s /usr/share/qemu/*.bin .
```

6. Modification of KVM Kernel Module on the Host OS

Modification of KVM kernel module in the host OS is required. Please apply the patch file (host-os/kvm.patch) and build the modified KVM kernel module.

After setting up the build environment for a kernel of Ubuntu 12.04 LTS x86_64, please update only the KVM kernel module.

```
    $ cd /path/to/linux-source-3.2.0
    $ cd arch/x86/kvm
    $ patch -p1 < /path/to/kvm.patch
    $ cd ../../..
    $ make M=arch/x86/kvm modules
```

When the make command finishes, kvm.ko and kvm-intel.ko are built under arch/x86/kvm. Please replace files under the /lib/modules/<os-version>/kernel/arch/x86/kvm directory of the host OS and reboot the host OS.

7. Execution of QEMU-KVM

After carrying out the procedures in Chapters 4 through 6, QEMU-KVM can be executed. Please appropriately update the path of qemu-system-x86_64 and the path of the KVM image file for the guest OS in the start-qemu.sh shell script file. The following operation starts the QEMU-KVM.

```
    $ sudo /path/to/start-qemu.sh
```

A login prompt is displayed at the terminal where the start-qemu.sh script is executed. Also, the guest OS can be connected using vnc with localhost:5901. This is as same as the usual QEMU-KVM.

After the QEMU-KVM is started, users can connect to the monitor of QEMU-KVM using the following command.

```
    $ sudo nc -U /tmp/qemu-monitor
```

From the monitor of QEMU-KVM, please enter the "dsysmon" command to start the D-System Monitor integrated with the QEMU-KVM.

```
    (qemu) dsysmon       <<< input
    dsysmon
    start dsysmon init_level4_pgt = 0x1a0c000 init_task = 0xffffffff81a16300 runqueues =
    0xffffffff81a2d420
```

Displayed values, such as init_level4_pgt, are virtual addresses specified in Section 5-3. High order position values (0xffffffff8) of the value of init_level4_pgt are omitted.

It is also possible to specify address values of init_levet4_pgt, init_task, and runqueues when starting dsysmon, as follows.

```
    (qemu) dsysmon 0xffffffff81a0d000 0xffffffff81a16400 0xffffffff81a2d520        <<< input
    dsysmon
    start dsysmon init_level4_pgt = 0x1a0d000 init_task = 0xffffffff81a16400 runqueues =
    0xffffffff81a2d520
```

Address values must be specified with hexadecimal numbers in the order of init_levet4_pgt, init_task, and runqueues. Three address values must be specified, including values that do not need to be changed.

Though the dsysmon command of the QEMU-KVM monitor starts the D-System Monitor and then several threads of D-System Monitor start running in the QEMU-KVM, observation of the guest OS is not started yet. To start the observation, some programs including GUI must be set up.

8. Applying Patches to the Guest OS

Note: Modification of the guest OS described in this chapter is to carry out abnormal behavior for demonstration purposes. Please do not use this modification except for demonstration purposes.

In order to demonstrate monitoring mechanisms, the guest OS needs to behave abnormally. Patches (guest-os/linux-source.patch) need to be applied to the uni-processor kernel built in Chapter 4. After applying the patches, kernel modules described in Chapter 9 can run on the kernel. As a reference, the .config file used at the DEOS R&D Center to build a kernel is included (guest-os/.config) in the published tar.gz file.

Please apply the linux-source.patch file to the source code of the uni-processor kernel of the guest OS.

```
$ cd linux-source-3.2.0
$ patch -p1 < /path/to/linux-source.patch
```

After applying the patch file, please build the kernel and replace the uni-processor kernel of the guest OS in the way shown in Chapter 4. If the address values of the symbols init_levet4_pgt, init_task, and runqueues become different from those in the uni-processor kernel built in Chapter 4, then, please re-build the QEMU-KVM as described in Sections 5-3 and 5-4. As described in Chapter 7, it is possible to specify the values when executing the dsysmon command from the monitor of QEMU-KVM.

9. Kernel Modules and Other Programs

Note: Modification of the guest OS described in this chapter is to carry out abnormal behavior for demonstration purposes. Please do not use this modification except for demonstration purposes.

In order to demonstrate monitoring mechanisms, the guest OS needs to behave abnormally. Kernel modules are prepared for carrying out abnormal behavior. In order to use kernel modules described in this chapter, modification of the kernel of the guest OS must be done as described in Chapter 8.

9-1. rpldev.ko and rpld

rpldev.ko is a kernel module to get keyboard input data, and rpld is a program to send the keyboard input data to the other host. Using these two programs, keyboard input data can be sent to the other host.

The libhx-dev package is required for build and execution. Please install it in both the guest OS and the host OS, using the following command.

```
$ sudo apt-get install libhx-dev
```

First, please find the source code of ttyrpld-2.60 and download it. For example, you may be able to use the following Web page.

```
http://sourceforge.net/projects/ttyrpld/
```

Please expand the downloaded source code ttyrpld-2.60.tar.bz2 at an appropriate directory. After applying a patch (ttyrpld-2.60/ttyrpld-2.60.patch), please build rpldev.ko and rpld.

```
$ cd ttyrpld-2.60
$ patch –p1 < /path/to/ttyrpld-2.60.patch
```

Before starting to build the rpldev.ko, please modify the MODULES_DIR definition in the ttyrpld-2.60/k_linux-2.6/Makefile, so that it can refer to the source code of the kernel for the guest OS.

```
$ cd ttyrpld-2.60
$ ./configure
$ make
$ cd k_linux-2.6
$ make
```

When the make command finishes, the rpldev.ko is built under the ttyrpld-2.60/k_linux-2.6 directory and the rpld is built under the ttyrpld-2.60/user directory. Adding the rpld.conf file, please send the three files rpldev.ko, rpld, and rpld.conf to the guest OS. Please put the rpld.conf file under the same directory as the rpld file in the guest OS. In the rpld.conf file, the IP address of the host OS must be set to NET_ADDR. From the next section, it is assumed that /rootkits directory is in the guest OS and that the three files, rpldev.ko, rpld, and rpld.conf are located under the /rootlkits directory.

9-2. rpld_receiver

Please build the rpld_receiver program, using files under the rpld-receiver directory in the published tar.gz file. This program runs in the host OS and receives data from the rpld program built in Section 9-1.

```
$ cd rpld-receiver
$ make
```

9-3. file_rootkit.ko

The file_rootkit.ko is a kernel module to make invalid file meta data, running in the guest OS. Please build the file_rootkit.ko, using files under the file-rootkit directory in the published tar.gz file. Before building it, please modify the MODULES_DIR definition in the file-rootkit/Makefile, so that it refers to the source code of the kernel for the guest OS.

```
$ cd file_rootkit
$ make
```

Please send the process_rootkit.ko under the /rootkits directory in the guest OS, after the make command finishes.

9-4. process_rootkit.ko

The process_rootkit.ko is a kernel module to make invalid process information, running in the guest OS. Please build the process_rootkit.ko, using files under the process-rootkit directory in the published tar.gz file. Before building it, please modify the MODULES_DIR definition in the process-rootkit/Makefile, so that it can refer to the source code of the kernel for the guest OS.

```
$ cd process_rootkit
$ make
```

After the make command finishes, please send the process_rootkit.ko under the /rootkits directory in the guest OS .

9-5. User Program for Monitoring

RootkitLibra is the only one of the three monitoring mechanisms that requires that a user program for monitoring the guest OS be run. The source code of the program rtkl-collect is located under the rtkl-collect.kvm directory. Please build the rtkl-collect as follows:

```
$ cd rtkl-collect
$ make
```

Please send the rtkl-collect to an appropriate directory in the guest OS, after the make command finishes.

10. Preparation for GUI

Because GUI is a program written in Python, Python must be installed in the host OS. In addition, the packages python-wxgtk2.8 and python-matplotlib are required.

```
$ sudo apt-get install python-wxgtk2.8 python-matplotlib
```

A shell script (DEMO-B-gui.sh) under the GUI directory in the published tar.gz file is used to start the GUI of the D-System Monitor. TOOL_DIR and RTKL_DIR in the DEMO-B-gui.sh must be changed appropriately, so that ${TOOL_DIR}/dsm-gui.py and ${RTKL_DIR}/rpld_receiver can refer to the correct executable files.

11. Execution of Demonstration

After carrying out the steps in Chapters 4 through 10, demonstration using GUI can be done.

11-1. Starting GUI

Please start GUI after starting dsysmon from the monitor of the QEMU-KVM. If the QEMU-KVM terminates abnormally, the /tmp/dsysmonNNNNN (NNNNN is a number) directory remains and the GUI program doesn't start. In that case, please remove the /tmp/dsysmonNNNNN directory and all files under it.

First, please start the QEMU-KVM from a terminal.

```
$ sudo /path/to/start-qemu.sh
```

Next, please connect to the monitor of the QEMU-KVM from another terminal and start dsysmon. As described in Chapter 7, it is possible to specify addresses, such as init_vevel4_ptg, when starting dsysmon.

```
$ sudo nc -U /tmp/qemu-monitor
(qemu) dsysmon        <<< input
dsysmon
start dsysmon init_level4_pgt = 0x1a0c000 init_task = 0xffffffff81a16300 runqueues =
0xffffffff81a2d420
```

Furthermore, please start GUI from the other terminal.

```
$ sudo /path/to/DEMO-B-gui.sh
```

Then, GUI windows are displayed.

11-2. Execution of FoxyKBD

First, please start rpld_receiver in the host OS without command options.

```
$ cd /path/to/directory/of/rpld_receiver
$ ./rpld_receiver &
```

Next, please execute /rootkits/rpld in the guest OS after loading the rpldev.ko kernel module. These operations must be done with root privilege. Before starting /rootkits/rpld, please verify that NET_ADDR in the rootkits/rpld.conf file is the IP address of the host OS.

```
$ sudo insmod /rootkits/rpldev.ko
$ sudo /rootkits/rpld &
```

Then, the /rootkits/rpld program in the guest OS obtains keyboard input data and sends the data to the rpld_receiver program in the host OS.

When the "start" button in the "control fkbd injection" area of the GUI is clicked, a large amount of simulated key inputs are generated. The "stop" button terminates the simulated key inputs. Because the left graph on the GUI shows network data transfer between the guest OS and the host OS, the user can determine whether the key input and network data transfer have a relation to each other, indicating suspicious behavior.

11-3. Execution of RootkitLibra

Please load the file_rootkit.ko kernel module in the guest OS.

```
$ sudo insmod /rootkits/file_rootkit.ko
```

Then, virtual files /proc/fhide and /proc/fmodify are created. Invalid file meta data for a file with an i-node number, InodeNumber, is generated by the following operations.

```
$ echo InodeNumber > /proc/fhide
$ echo InodeNumber size > /proc/fmodify
```

Generated invalid values can be verified using commands such as "ls -li".

In the guest OS, please mount a directory exported from the host OS using NFS version 3. At this time, NFS version 4 is not supported. If file meta data of files under the NFS-mounted directory is changed to invalid data and the rtkl-collect program is executed upon the directory, the abnormal data in the host OS is detected.

```
$ sudo mount.nfs -o nfsvers=3 192.168.1.26:/aaa /mnt
$ echo 123456 > /proc/fhide
$ /path/to/rtkl-collect -r /mnt
```

In the example above, the IP address of the host OS is 192.168.1.26. The /aaa directory of the host OS is mounted under the /mnt directory of the guest OS, using NFS version 3. By writing 123456 in the /proc/fhide file using the echo command, the file with i-node number 123456 under the /mnt directory becomes hidden. The rtkl-collect program sends file meta data on the guest OS to the host OS. Invalid file meta data is detected by comparing the data from the rtkl-collect program and data in the NFS packets.

Then, after clicking the "start" button in the "RootkitLibra" area of GUI, detected invalid data is reported every five seconds.

11-4. Execution of Waseda LMS

Please load the process_rootkit.ko kernel module in the guest OS.

```
$ sudo insmod /rootkits/process_rootkit.ko
```

Then, a virtual file, /proc/phide, is created. Invalid process data for the file with process-ID PID is generated by the following operations.

```
$ echo PID > /proc/phide
```

This process is not shown with commands, such as “ps”.

Then, after clicking the "start" button in the "Lightweight Monitoring Service" area of GUI, detected invalid data is reported every second. Because invalid process data is detected only when the process is runnable, processes that are always runnable, such as a program with an infinite loop, must be used to demonstrate this mechanism.

12. Conclusion

Using QEMU-KVM+D-System Monitor, we can watch a guest OS from the outside and detect abnormal behavior of the guest OS. Because the QEMU-KVM can be used in various environments, incorporating the D-System Monitor in the QEMU-KVM is expected to extend the range of hardware supporting the D-System Monitor.