WinKVM: Windows Kernel-based Virtual Machine

Kazushi Takahashi, Koichi Sasada
University of Tokyo
About me

• Name:
  - 一志 高橋
  - Kazushi Takahashi

• My research area:
  - System software, operating system and virtual machine technology
  - Interested in Linux kernel hacking, distributed system and parallel programming

• Twitter: ddk50
• Blog: http://d.hatena.ne.jp/ddk50/
Agenda

• We have implemented WinKVM
  – WinKVM is a port of KVM(-17) to Microsoft Windows.
• Main point of today’s talk: “How we developed WinKVM”
  – KVM is implemented as Linux device driver
  – Porting “kvm.ko and intel-kvm.ko” to Windows drivers
  – Developing an emulation layer to run Linux drivers on Windows
  – This emulation layer translates Linux kernel functions into Windows kernel APIs

• Why we develop WinKVM
  – To provide a VMM that supports both Windows and Linux
  – To search for the new way of KVM usage
Overview of Our Method

We implemented a Linux emulation layer
- To reduce implementation costs
- To enable any version of KVM to run

WinKVM

Windows Kernel

Kernel-mode-layer

System call

Signal

VMM

Users

QEMU

libkvm

Add modifications to connect QEMU and WinKVM

User-mode-layer

Link the emulation layer and KVM
Now, We can get WinKVM!
Examples of Translated APIs

Most of Linux functions have corresponding windows kernel functions

<table>
<thead>
<tr>
<th>Linux Kernel Functions</th>
<th>Windows Kernel Functions used to emulate the function</th>
</tr>
</thead>
<tbody>
<tr>
<td>kmalloc()</td>
<td>ExAllocatePoolWithTag()</td>
</tr>
<tr>
<td>kfree()</td>
<td>ExFreePoolWithTag()</td>
</tr>
<tr>
<td>mutex_init()</td>
<td>ExInitializeFastMutex()</td>
</tr>
<tr>
<td>mutex_lock()</td>
<td>ExAcquireFastMutex()</td>
</tr>
<tr>
<td>mutex_unlock()</td>
<td>ExReleaseFastMutex()</td>
</tr>
<tr>
<td>mutex_trylock()</td>
<td>ExTryToAcquireFastMutex()</td>
</tr>
<tr>
<td>alloc_page()</td>
<td>ExAllocatePoolWithTag()</td>
</tr>
<tr>
<td>free_page()</td>
<td>ExFreePoolWithTag()</td>
</tr>
<tr>
<td>__va()</td>
<td>MmGetVirtualForPhysical()</td>
</tr>
<tr>
<td>__pa()</td>
<td>MmGetPhysicalAddress()</td>
</tr>
</tbody>
</table>
Two Technical Problems

1. Difference in binary formats (compilers):
   - How to link our emulation layer and KVM drivers
   - KVM source code (inline-asm) depends on GCC
   - Windows driver developers have to use Visual C++
   - Visual C++ CANNOT compile KVM source code

1. Difference in memory architectures:
   - KVM driver and QEMU share guest OS memory region
   - Both OSs support memory sharing between kernel and user memory space
   - Difficulties in emulating Linux memory interfaces by Windows kernel
   - The fault (nopage) handler, which is used by KVM is not supported by Windows.
How to handle the difference in binary formats

1st step
Linux kernel source (version 2.6.24-24)

2nd step
KVM source code (use of KVM-17)

3rd step
Our Emulation layer source code

Pick up some source code. i.e. linked list implementation
Leave the “portable”-potion of the Linux kernel code unmodified to reuse

Compile KVM using Cygwin gcc
We get compiled KVM as COFF binary

*.obj
Link the KVM objects and our emulation layer

Visual C++
With WDK

Finally, We get winkvm.sys!!
How to handle the differences in memory architectures (1/2)

• Problem:
  - The fault handler in Linux is difficult to emulate by Windows kernel functions

• Solution:
  - Modify KVM source code to avoid use of the fault handler
  - Only 1-line modification
How to handle the differences in memory architectures (2/2)

- The mechanism of this modification
  1. Before starting KVM emulation, our emulation layer construct memory mapping regions between kernel and user-space. The layer has already mapped GPA to HVA
  2. When KVM itself also tries to map GPA to HVA, our patch overwrites the mapping with our emulation layer mapping
  3. Never call the fault (nopage) handler in KVM

Latest KVM may not need this modification
DEMO

Does our method work well?

In this demo, we execute on Linux kernel 2.6.20 attached to QEMU, and execute some applications
Future Work

• Overcome guest memory limitation
  - Max 300MB guest memory from 2Gbyte physical memory
  - We are able to solve this problem
    • Modify QEMU to gather scattered memory chunks as a single guest memory space

• Add new functions to the emulation layer
  - Implement SMP functions such as smp_call_function()
  - Catch up the latest version of KVM
    • Nested paged KVM
    • PCI Pass-through

• Debug :-(

12
Summary

• We have implemented WinKVM
  – A port of KVM(-17) to Microsoft Windows

• Main point of today’s talk: “How we developed WinKVM”
  – We implemented an emulation layer to run Linux drivers on Windows
  – We developed WinKVM using this emulation layer
Thank you for your attention!

Have a look at WinKVM repository in GitHub

http://github.com/ddk50/