Testing System Virtual Machines

Lorenzo Martignoni\textsuperscript{1}  Roberto Paleari\textsuperscript{2}
Giampaolo Fresi Roglia\textsuperscript{2}  Danilo Bruschi\textsuperscript{2}

\textsuperscript{1}Università degli Studi di Udine \textsuperscript{2}Università degli Studi di Milano

International Conference on Software Testing and Analysis (ISSTA 2010)
Virtual machines

"An efficient, isolated duplicate of a real machine"
– Popek & Goldberg
System virtual machines

- VMware
- Parallels
- Microsoft Virtual PC
- Xen

Typically used for:
- Resources consolidation (e.g., VPS)
- System integration
- Development
- Performance

Performance is essential!
A **system virtual machine** provides a complete system platform which supports the execution of a complete operating system.

Typically used for:
- Resources consolidation (e.g., VPS)
- System integration
- Development

*Performance is essential!*
**Emulation**: the environment offered by the physical CPU is simulated by emulating all the instructions

**Native execution**: the environment is simulated by executing as much code as possible directly *as-is* on the host and by using emulation when that is not possible

- The trick used to natively run guest code efficiently is to execute both user and system code of the guest on the host, in user-mode
- Emulation is then used to execute only instructions of the system code that cannot be executed natively on the host
**Privileged instruction**: an instruction that traps when executed in user-mode and does not when executed in system-mode

**Sensitive instruction**: an instruction that changes the configuration of the resources in the system or whose behavior depends on the configuration of the resources

Recall that the trick used by native execution is to execute both user and system code of the guest on the host, in user-mode
Popek & Goldberg requirements for efficient virtualization

- **Privileged instruction**: an instruction that traps when executed in user-mode and does not when executed in system-mode

- **Sensitive instruction**: an instruction that changes the configuration of the resources in the system or whose behavior depends on the configuration of the resources

All sensitive instructions trap and can be easily intercepted and emulated
Popek & Goldberg requirements for efficient virtualization

- **Privileged instruction**: an instruction that traps when executed in user-mode and does not when executed in system-mode.
- **Sensitive instruction**: an instruction that changes the configuration of the resources in the system or whose behavior depends on the configuration of the resources.

The code must be statically analyzed and patched to ensure that all sensitive instructions can be intercepted.
Intel x86 is **not** virtualization friendly

The Intel x86 instruction set of the Pentium processor contains **17 sensitive, but not privileged instructions!**
Intel x86 is **not** virtualization friendly

The Intel x86 instruction set of the Pentium processor contains **17 sensitive, but not privileged instructions**!

An Intel x86 VM based on native execution is as complex as one based on emulation, and prone to the same kind of problems.
Intel x86 is not virtualization friendly

The Intel x86 instruction set of the Pentium processor contains 17 sensitive, but not privileged instructions!

An Intel x86 VM based on native execution is as complex as one based on emulation, and prone to the same kind of problems.

New Intel x86 CPUs provide hardware facilities for virtualization (VT-x), but traditional native execution is still widely in use.
What could go wrong if $\neq$ ?

No kidding, the risk is real!
What could go wrong if $a \neq b$?

**CVE-2009-2267**
VMware Workstation 6.5.x., when Virtual-8086 mode is used, do not properly set the exception code upon a page fault (aka #PF) exception, which allows guest OS users to gain privileges on the guest OS by specifying a crafted value for the cs register.

Found by Tavis Ormandy & Julien Tinnes

**CVE-2009-1542**
The Virtual Machine Monitor (VMM) in Microsoft Virtual PC., does not enforce CPU privilege-level requirements for all machine instructions, which allows guest OS users to execute arbitrary kernel-mode code and gain privileges within the guest OS via a crafted application, . . .

Found by Tavis Ormandy & Julien Tinnes
Our contribution

A testing methodology to test system virtual machines: does the virtual machine behave exactly as the physical one?
Our contribution

A testing methodology to test system virtual machines: does the virtual machine behave exactly as the physical one?

Used to test four system virtual machines: BOCHS, QEMU, VMware, VirtualBox
EmuFuzzer vs KEmuFuzzer

EmuFuzzer (ISSTA’09)

- Testing is performed by comparing the behavior of the CPU of the VM with the behavior of the physical CPU

KEmuFuzzer

- Testing is performed using the same principle, but the methodology is completely different
EmuFuzzer vs KEmuFuzzer

EmuFuzzer (ISSTA’09)

- Testing is performed by comparing the behavior of the CPU of the VM with the behavior of the physical CPU
- Can test only user-mode code

KEmuFuzzer

- Testing is performed using the same principle, but the methodology is completely different
- Can test both user-mode and system-mode code
EmuFuzzer vs KEmuFuzzer

EmuFuzzer (ISSTA’09)

🔹 Testing is performed by comparing the behavior of the CPU of the VM with the behavior of the physical CPU
🔹 Can test only user-mode code
🔹 Can test only process virtual machines

KEmuFuzzer

🔹 Testing is performed using the same principle, but the methodology is completely different
🔹 Can test both user-mode and system-mode code
🔹 Can test system virtual machines
EmuFuzzer vs KEmuFuzzer

EmuFuzzer (ISSTA’09)

- Testing is performed by comparing the behavior of the CPU of the VM with the behavior of the physical CPU
- Can test only user-mode code
- Can test only process virtual machines

KEmuFuzzer

- Testing is performed using the same principle, but the methodology is completely different
- Can test both user-mode and system-mode code
- Can test system virtual machines

The problem is much more challenging!
We can imagine the CPU as an abstract machine

A state $s = (pc, R, M, E)$ of the abstract machine consists of:

- $pc$ – state of the program counter
- $R$ – state of the CPU/FPU registers
- $M$ – state of the (physical) memory
- $E$ – exception state (e.g., ⊥, PF, GPF, UD)
Modelling the behavior of the CPU

To execute an instruction means to transition into a new state

\[ s = (pc, R, M, E) \]

If the instruction is successfully executed,

\[ s' = (pc', R', M', E') \]

If an exception occurred,

\[ s' = (pc', R', M', \perp) \]
Modelling the behavior of the CPU

To execute an instruction means to transition into a new state

\[ s' = (pc', R', M', \perp) \]

the instruction is successfully executed

\[ s = (pc, R, M, E) \]

\( pc \) points to the next instruction, \( R \) and \( M \) are updated according to the semantics of the instruction executed
Modelling the behavior of the CPU

To execute an instruction means to transition into a new state

\[ s = (pc, R, M, E) \]

\[ s' = (pc', R', M', \perp) \]

the instruction is successfully executed

\[ s' = (pc, R, M, E') \]

an exception occurred

\( E \) denotes the exception occurred, \( pc, R, \) and \( M \) are not updated since the execution of the instruction is atomic
The guest (OS and applications) must not be able to distinguish between a physical and a virtual machine.
The guest (OS and applications) must not be able to distinguish between a physical and a virtual machine.

Lack of transparency is highly related to the existence of bugs.
Transparency to guests

\[ s = (p_c, R, M, \perp) \]
Transparency to guests

\[ s = (pc, R, M, \bot) \]

\[ s = (pc', R', M', E') \]

\[ s = (pc', R', M', E') \]

L. Martignoni, R. Paleari, G. Fresi Roglia, D. Bruschi  
Testing System Virtual Machines  
13
Transparency to guests

\[ s = (pc, R, M, \bot) \]

\[ s = (pc', R', M', E') \]

\[ s = (pc', R', M', E') \]

The VM is **transparent to the guest**, for the state \( s \), if the state resulting from the execution in the two environments match.
Transparency to guests

\[ s = (pc, R, M, \bot) \]

\[ s' = (pc', R', M', E') \]

\[ s = (pc', R', M', E') \]

The VM is **not transparent to the guest**, for the state \( s \), if the state resulting from the execution in the two environments differ.
Challenges in testing system virtual machines

- Our definition of transparency assumes that the execution is deterministic:
  - Render the execution as such
- The state space is enormous:
  - Focus efforts on states that requires emulation
- Testing of privileged instruction require to execute code in system-mode:
  - Guarantee that we can always regain the control of the execution
Architecture of KEmuFuzzer

Test-case → Bootable floppy → Virtual machine → Oracle

Kernel

$s \rightarrow s'$

Architecture of KEmuFuzzer

High-level test-case template
A custom compiler mutates and compiles the template
Custom kernel used to bootstrap the virtual machine and to executed the test-case
Architecture of KEmuFuzzer

Bootable floppy image to boot the kernel in a virtual or physical machine

Test-case

Kernel

Bootable floppy

Virtual machine

Oracle

$s \quad s'$

=?

L. Martignoni, R. Paleari, G. Fresi Roglia, D. Bruschi
Testing System Virtual Machines
Architecture of KEmuFuzzer

Off-the-shelf VM extended to dump to disk the state of the CPU when a command is received on a specific I/O port.
Architecture of KEmuFuzzer

Oracle based on a hardware assisted VM
(initial state is forced manually)
Architecture of KEmuFuzzer

Off-line diffing utility used to compare output states
Mutating test-cases (protocol specific fuzzing)

- Test-cases contain operations that are believed to be significant for the testing (e.g., that trigger corner-case situations)
- Symbolic operators (KEF) allows to generate multiple significant test-cases from the same input
Mutating test-cases (protocol specific fuzzing)

- Test-cases contain operations that are believed to be significant for the testing (e.g., that trigger corner-case situations)
- Symbolic operators (KEF_) allows to generate multiple significant test-cases from the same input

```xml
<testcase start_ring="0">
  <ring0>
    // Register syscall handler
    ...
    // Jump to ring3 code
    KEF_JUMP_RING(3);
  </ring0>

  <ring3>
    // Invoke syscall
    mov $0x25, %eax;
    KEF_PREFIX sysenter;
  </ring3>
</testcase>
```

```xml
<testcase start_ring="0">
  <ring0>
    // Load flat data segment
    ...
    // Calculate PTE address
    movl KEF_RING_BASE(3), %eax;
    ...do some math on %eax...
    addl KEF_PT_BASE, %eax;
    // Flip some bits in the PTE
    movl (%eax), %ebx;
    btc KEF_INTEGER(5), %ebx;
    movl %ebx, (%eax);
    ...
    // Jump to ring3 code
    KEF_JUMP_RING(3);
  </ring0>

  <ring3>
    movb $0x0, 0x0;
  </ring3>
</testcase>
```
Modern x86 CPUs (with VT-x extensions) have built-in support for virtualization
Modern x86 CPUs (with VT-x extensions) have built-in support for virtualization

Specific assumptions about our peculiar guest allow to develop a minimalistic VT-x based VM that can be used as a oracle
VT-x based oracle

VM controller

User-mode
/dev/kvm

System-mode

GNU/Linux

Oracle VM Test-case

Ring 3
Ring 2
Ring 1
Ring 0

Kernel

Non-root mode

Root mode

Minimalistic KVM-based VM

L. Martignoni, R. Paleari, G. Fresi Roglia, D. Bruschi
Testing System Virtual Machines
VT-x based oracle

Test-case executed as-in in a VM
VT-x based oracle

Hardware assisted VM monitor executes in root-mode
The user-space controller interacts with the VM monitor to launch the VM and to dump the final state.
Experimental evaluation

**Tested VMs**
- VirtualBox OSE (3.0.8)
- VMware Workstation (7.0)
- QEMU (0.11.0)
- BOCHS (7 nov 2009)

**Oracle**
- Intel Core2 Duo (3.00GHz)

**Operating mode**
- 32-bit protected mode
Experimental evaluation

Tested VMs

- VirtualBox OSE (3.0.8)
- VMware Workstation (7.0)
- QEMU (0.11.0)
- BOCHS (7 nov 2009)

Oracle

- Intel Core2 Duo (3.00GHz)

Operating mode

- 32-bit protected mode

Test-cases

Generated 1915 test-cases from 67 manually written templates

- Correct instruction decoding and privileges enforcement
- Behavior of the CPU with “abnormal” MMU configurations
- Attempt to execute random code
- Some of the test-cases used with EmuFuzzer
Templates and test-cases producing differences

- VMware
- VirtualBox
- QEMU
- BOCHS

Templates and Test-cases comparison: VMware has the least differences, followed by VirtualBox, QEMU, and BOCHS.
sysenter corrupts the state of a segment, GDT entries are not properly updated
Templates and test-cases producing differences

Crashes when certain hw breakpoints, raises wrong exceptions, accepts illegal instructions, ...
Unable to run in native execution mode with KEmuFuzzer’s kernel, manifests most of the issues found in QEMU
Templates and test-cases producing differences

- **BOCHS**
- **QEMU**
- **VirtualBox**
- **VMware**

The `retn` instruction might corrupt the stack, GDT entries are not properly updated.
CVE-2009-2267

- Latest version of the kernel supports VM86 tasks/segments
CVE-2009-2267

- Latest version of the kernel supports VM86 tasks/segments
- KEmuFuzzer identified the problem in VMware
 CVE-2009-2267

- Latest version of the kernel supports VM86 tasks/segments
- KEmuFuzzzer identified the problem in VMware
- KEmuFuzzzer identified the problem in QEMU as well
CVE-2009-2267

- Latest version of the kernel supports VM86 tasks/segments
- KEmuFuzzer identified the problem in VMware
- KEmuFuzzer identified the problem in QEMU as well

We are getting ready for large scale testing of VM86 mode :-}
In summary

\[ s = (pc, R, M, \bot) \]

\[ s_1 = (pc', R', M', E') \]

\[ s_2 = (pc', R', M', E') \]
In summary

\[ s = (pc, R, M, \bot) \]

\[ s_{\text{new}} = (pc', R', M', E') \]

\[ s_{\text{new}} = (pc', R', M', E') \]

Test-case

Oracle

Bootable floppy

Virtual machine

Kernel
In summary
In summary

\[ s = (pc, R, M, \bot) \]

\[ s^* = (pc^*, R', M', E') \]

\[ s = (pc', R', M', E') \]
In summary

\[ s = (pc', R, M, \bot) \]

\[ s_{\text{test}} = (pc', R', M', E') \]

Test-case

Bootable floppy

Virtual machine

Oracle

s

s'

Kernel

VM controller

Oracle VM

Test-case

Ring 1

Ring 1

Ring 0

Kernel

GNU/Linux

Minimalistic KVM-based VM

User mode

System mode

Non-root mode

Root mode

\[ \text{http://code.google.com/p/kemufuzzzer} \]
Testing system virtual machines

Test-case
Kernel

Bootable floppy
Virtual machine

Oracle

http://code.google.com/p/kemufuzzer

Thank you!
Any questions?

Lorenzo Martignoni
lorenzo.martignoni@uniud.it