Taming Hosted Hypervisors with (Mostly) Deprivileged Execution

Chiachih Wu†, Zhi Wang*, Xuxian Jiang†

†North Carolina State University, *Florida State University
Virtualization is Widely Used

- “There are now hundreds of thousands of companies around the world using AWS to run all their business, or at least a portion of it. They are located across 190 countries, which is just about all of them on Earth.”
  
  Werner Vogels, CTO at Amazon
  AWS Summit ‘12

- “Virtualization penetration has surpassed 50% of all server workloads, and continues to grow.”
  
  Gartner
  Magic Quadrant for x86 Server Virtualization Infrastructure
  June ‘12
Threats to Hypervisors

- **Large Code Bases**

<table>
<thead>
<tr>
<th>Hypervisor</th>
<th>SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xen (4.0)</td>
<td>194K</td>
</tr>
<tr>
<td>VMware ESXi¹</td>
<td>200K</td>
</tr>
<tr>
<td>Hyper-V¹</td>
<td>100K</td>
</tr>
<tr>
<td>KVM (2.6.32.28)</td>
<td>33.6K</td>
</tr>
</tbody>
</table>

¹: Data source: NOVA (Steinberg et al., EuroSys '10)

- **Vulnerabilities**

<table>
<thead>
<tr>
<th>Hypervisor</th>
<th>Vulnerabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xen</td>
<td>41</td>
</tr>
<tr>
<td>KVM</td>
<td>24</td>
</tr>
<tr>
<td>VMware ESXi</td>
<td>43</td>
</tr>
<tr>
<td>VMware Workstation</td>
<td>49</td>
</tr>
</tbody>
</table>

Data source: National Vulnerability Database ('09~'12)
Threats to Hosted Hypervisors

Can we prevent the compromised hypervisor from attacking the rest of the system?
DeHype

- Decomposing the KVM hypervisor codebase
  - De-privileged part → user-level (93.2% codebase)
  - Privileged part → small kernel module (2.3 KSLOC)

~4% overhead
Challenges

- Providing the OS services in user mode
- Minimizing performance overhead
- Supporting hardware-assisted memory virtualization at user-level
Challenge I

- Providing the OS services in user mode

- De-privileged Hypervisor

Original Hosted Hypervisor

DeHype’d Hosted Hypervisor

User

Kernel

Hypervisor

Host OS

Physical Hardware

HypeLet
Dependency Decoupling

- Abstracting the host OS interface and providing OS functionalities in user mode

- For example
  - Memory allocator: kmalloc/kfree, alloc_page, etc.
  - Kernel APIs for memory access: virt_to_page, etc.
  - Scheduling, signal handling, invoking system calls
    - Leveraging GLIBC
## Dependency Decoupling

### 10 privileged services provided by HypeLet

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMREAD</td>
<td>Read VMCS fields</td>
</tr>
<tr>
<td>VMWRITE</td>
<td>Write VMCS fields</td>
</tr>
<tr>
<td>GUEST_RUN</td>
<td>Perform host-to-guest world switches</td>
</tr>
<tr>
<td>GUEST_RUN_POST</td>
<td>Perform guest-to-host world switches</td>
</tr>
<tr>
<td>RDMSR</td>
<td>Read MSR registers</td>
</tr>
<tr>
<td>WRMSR</td>
<td>Write MSR registers</td>
</tr>
<tr>
<td>INVVPID</td>
<td>Invalidate TLB mappings based on VPID</td>
</tr>
<tr>
<td>INVEPT</td>
<td>Invalidate EPT mappings</td>
</tr>
<tr>
<td>INIT_VCPU</td>
<td>Initialize vCPU</td>
</tr>
<tr>
<td>MAP_HVA_TO_PFH</td>
<td>Translate host virtual address to physical frame</td>
</tr>
</tbody>
</table>

- **Privileged instructions**
- **Service routines**
Challenge II

- Minimizing performance overhead

1. System call
2. Function call
3. 195187 privileged instructions

DeHyped KVM

HypeLet

- 195187 system calls

- 10% performance overhead
Optimization: Caching VMCS

- VMCS (Virtual Machine Control Structure)
  - ~90% of the privileged instructions issued by the hypervisor are for accessing VMCS
  - Accessed by the hypervisor for monitoring or controlling the behavior of the guest VM
  - Indirectly affected by the guest VM throughout the running period in guest mode
Optimization: Caching VMCS

- Maintaining cached copy of VMCS in user-level
- Caching only the most frequently accessed fields

- Caching 8 VMWRITE’d fields: 98.28% VMWRITE system calls reduced

<table>
<thead>
<tr>
<th>Top 8 Most Frequently VMWRITE’d VMCS Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU_BASED_VM_EXEC_CONTROL</td>
</tr>
<tr>
<td>VM_ENTRY_INTR_INFO_FIELD</td>
</tr>
</tbody>
</table>

- Caching 28 VMREAD’d fields: 99.86% VMREAD system calls reduced
Challenge III

- Supporting hardware-assisted memory virtualization at user-level
  - Maintaining nested page tables which translate guest-physical to host-physical addresses
    - Memory may be paged out
    - Virtual-physical mapping information is unknown
  - Preventing the untrusted hypervisors from accessing memory areas not belonged to them
    - Batch-processing NPT updates with sanity checks in HypeLet
Implementation and Evaluation

- Prototype
  - KVM 2.6.32.28 with qemu-kvm-0.14.0
  - ~93.2% of KVM codebase is de-privileged
  - 2.3K SLOC small kernel module (HypeLet)

- Evaluation
  - Security benefits
  - Non-security benefits
  - Performance
Testing real-world vulnerabilities

- CVE-2010-0435
  - Guest OS causing a NULL pointer dereference (accessing debug registers with MOV) in KVM running in privileged mode
Facilitating hypervisor development

- e.g., debugging the NPT fault handler with GDB
  - set breakpoint
  - NPT fault occurs
  - register dump
  - call trace
  - continue the program
Running multiple hypervisors

- Running each hypervisor in a different security level
  - Suspicious guests: running on VMI-enabled hypervisors
  - Others: running on normal hypervisors

- Live-migrating guests to another hypervisor in the same host computer
  1. New vulnerability reported and fixed
  2. Starting a patched hypervisor
  3. Live-migrating all guests one-by-one
Performance Evaluation

- Test platform
  - Dell OptiPlex 980: Intel Core i7 860 + 3G RAM
  - Host: Ubuntu 11.10 desktop + Linux kernel 2.6.32.28
  - Guests: Ubuntu 10.04.2 LTS server

- Benchmarks

<table>
<thead>
<tr>
<th>Software Package</th>
<th>Version</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEC CPU2006</td>
<td>v1.0.1</td>
<td>Reportable int</td>
</tr>
<tr>
<td>Bonnie++</td>
<td>1.03e</td>
<td>bonnie++ -f -n 256</td>
</tr>
<tr>
<td>Linux kernel</td>
<td>2.6.39.2</td>
<td>untar_kernel: tar zfx &lt;KERNEL-TARBALL&gt; make_kernel: make defconfig vmlinux</td>
</tr>
</tbody>
</table>
Relative Performance

- DeHype
- DeHype+VMCS caching
- DeHype+VMCS caching+securely NPT updates
Discussion

- HypeLet and the host OS are a part of the TCB
  - HypeLet is the main attack surface in the cloud environment
  - HypeLet is highly constrained (2.3 KSLOC, 10 services)

- Prototype limitations
  - Pinning guest memory
    - Could be extended with Linux MMU notifier
  - Not supporting all KVM features
    - SMP
    - Para-virtualized I/O
Related Work

- Improving hypervisor security
  - seL4 (Klein et al., SOSP ’09), NOVA (Steinberg et al., EuroSys ’10), HyperLock (Wang et al., EuroSys ’12) …

- Isolating untrusted device drivers
  - Nooks (Swift et al., SOSP ‘03), Microdrivers (Ganapathy et al., ASPLOS ‘08) …

- Applying virtualization to host security
  - HookSafe (Wang et al., CCS ‘09), Lockdown (Vasudevan et al., TRUST ‘12) …
Conclusion

- DeHype substantially reduces hosted hypervisor’s attack surface and brings additional benefits
  - Better development and debugging
  - Concurrent execution of multiple hypervisors

![Diagram showing DeHyped KVM and HypeLet with 93.2% of original KVM and 2.3 KSLOC]
Thanks, Questions?

Chiachih Wu cwu10@ncsu.edu
Backup Slides
Memory Rebasing

1. Pre-allocating pinned memory in kernel space

2. Remapping the pinned memory to user space

3. $u_{\text{addr}} \rightarrow k_{\text{addr}}$

4. $k_{\text{addr}} \rightarrow p_{\text{addr}}$

1. Pre-allocating pinned memory in kernel space

2. Remapping the pinned memory to user space
Securely Update NPT Entries

- Preventing the untrusted hypervisor from updating the NPT tables directly
  - Recording the update operations into buffer
  - Batch-processing the updates in next host-to-guest switch with sanity check (by HypeLet)
  - Issue: the hypervisor needs the actual NPTs to traverse the layer-based NPTs

Update entry \( l \):
1. Allocate \( A \); \( R[i] = A \)
2. Allocate \( B \); \( A[j] = B \)
3. Allocate \( C \); \( B[k] = C \)
4. Update \( C[l] \)

Update entry \( m \):
1. \( A = R[i] \)
2. \( B = A[j] \)
3. \( C = B[k] \)
4. Update \( C[m] \)

Recording only

Cannot traverse
**Pseudo NPT**

- **Host mode, User-level**
  - Pseudo NPTs (allocated from heap)
  - Allocate A; $R[i]=A$
  - Allocate B; $A[j]=B$
  - Allocate C; $B[k]=C$

- **Privileged Service Request**

- **VM Entry**

- **Real NPTs (allocated from the remapped memory pool)**

- **Time**

- **Guest Mode**

- **Guest**

- **Host mode, Kernel-level**

- **Buffer**
Intel VT-x: World Switches

**VM Entry**
- Transition from VMM to Guest (**VMLAUNCH/VMRESUME**)
- Enters VMX non-root operation (guest mode)
- Saves VMM state in VMCS
- Loads Guest state and exit criteria from VMCS

**VM Exit**
- Transition from Guest to VMM (**VMEXIT**)
- Enters VMX root operation (host mode)
- Saves Guest state in VMCS
- Loads VMM state from VMCS
## Optimization: Caching VMCS

### Top 28 Most Frequently VMREAD’ed VMCS Fields

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Name</th>
<th>Field Name</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUEST_INTERRUPTIBILITY_INFO</td>
<td>EXIT_QUALIFICATION</td>
<td>GUEST_CS_BASE</td>
<td>GUEST_RSP</td>
</tr>
<tr>
<td>IDT_VECTORING_INO_FIELD</td>
<td>GUEST_CS_SELECT</td>
<td>GUEST_DS_BASE</td>
<td>GUEST_RIP</td>
</tr>
<tr>
<td>GUEST_PHYSICAL_ADDRESS_HIGH</td>
<td>GUEST_CS_AR_BYTE</td>
<td>GUEST_ES_BASE</td>
<td>GUEST_CR0</td>
</tr>
<tr>
<td>GUEST_PHYSICAL_ADDRESS_YELLOW</td>
<td>GUEST_PDPTR0_HIGH</td>
<td>GUEST_PDPTR0</td>
<td>GUEST_CR3</td>
</tr>
<tr>
<td>VM_EXIT_INTR_INFO</td>
<td>GUEST_PDPTR1_HIGH</td>
<td>GUEST_PDPTR1</td>
<td>GUEST_CR4</td>
</tr>
<tr>
<td>VM_EXIT_INSTRUCTION_LEN</td>
<td>GUEST_PDPTR2_HIGH</td>
<td>GUEST_PDPTR2</td>
<td>GUEST_RFLAGS</td>
</tr>
<tr>
<td>CPU_BASED_VM_EXEC_CONTROL</td>
<td>GUEST_PDPTR3_HIGH</td>
<td>GUEST_PDPTR3</td>
<td>VM_EXIT_REASON</td>
</tr>
</tbody>
</table>
Combining privileged instructions

- **VMPTRLD**: a privileged instruction to load guest states before switching to guest mode

- CPU intensive workload
  - KVM handles most VM Exits
  - **One** VMPTRLD is followed by **multiple** runs of (VMRESUME, VMEXIT)
  - The latency of VMPTRLD is not significant
Combining privileged instructions

- IO intensive workload
  - QEMU handles most VM exits for issuing IO instructions
  - One VMPTRLD is followed by one run of (VMRESUME, VMEXIT)
  - VMPTRLD introduces significant latency

- Postponing the VMPTRLD instruction until the first VMRESUME instruction
Testing real-world vulnerabilities

- CVE-2009-4031
  - KVM attempting to interpret wrong-size (too long) instructions
  - Being exploited
    - Causing large latencies in non-preempt hosts

- With DeHype
  - Instruction emulation is done in user-level where preemption is natively enabled
Testing real-world vulnerabilities

- CVE-2010-3881
  - KVM copying certain data structures to user program without clearing the padding
  - Being exploited
    - QEMU processes potentially obtaining sensitive information from kernel stack

- With DeHype
  - QEMU process obtaining information from the stack of the hypervisor paired with it, not from the kernel stack