Deconstructing Xen

Lei Shi, Yuming Wu, Yubin Xia, Nathan Dautenhahn, Haibo Chen, Binyu Zang, Haibing Guan, Jinming Li

Shanghai Jiao Tong University, University of Pennsylvania, Huawei Inc.
Hypervisors have Bugs

- Xen is used by Amazon EC2
- Xen’s CVE is growing
  - 210 XSA (Xen Security Advisories)
  - Xen’s LoC is growing from 45K (v2.0) to 270K (v4.0)
- KVM also has 100+ CVEs

Data from https://xenbits.xen.org/xsa/
Analyze 201 of Xen’s Vulnerabilities (XSA)

191

144 (75% of 191)

47

10

144 are in the hypervisor
- E.g., Host DoS, privilege escalation, etc.
- Use hypervisor to attack VM

47 are not in hypervisor
- Some are in Domain-0
- Some are in Qemu

10 are ignored
- 7 numbers are not used
- XSA-161 was withdrawn
- XSA-99 is irrelevant
- XSA-166 is too vague

Focus on this part
3 Dimensions to Categorize (144 Hypervisor bugs)

<table>
<thead>
<tr>
<th>Attack targets</th>
<th>Key steps of attack</th>
<th>Results of attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypervisor</td>
<td>Hypervisor</td>
<td>Hypervisor</td>
</tr>
<tr>
<td>Attack targets</td>
<td>Key steps of attack</td>
<td>Results of attack</td>
</tr>
<tr>
<td>Memory management: 25.7%</td>
<td>Memory corruption: 45.1%</td>
<td>Host DoS: 61.8%</td>
</tr>
<tr>
<td>CPU virtualization: 21.5%</td>
<td>Misuse of hardware: 22.2%</td>
<td>Privilege escalation: 15.3%</td>
</tr>
<tr>
<td>Code emulation: 13.2%</td>
<td>Live lock: 8.3%</td>
<td>Info leak: 13.9%</td>
</tr>
</tbody>
</table>

...
1. Xen Components with Bugs

- **Components with bugs**
  - 25.69%: Memory management
  - 21.53%: CPU virtualization
  - 13.19%: Code emulation

- **Observations**:
  - Some components are more attractive to attackers
  - Memory management is critical and hard to get right
2. The Types of Key Step of Attack

- Memory corruption: 45.14%
  - Illegal memory read
    - E.g., out-of-boundary
  - Illegal memory write
    - E.g., write to an invalid pointer

- Misuse of hardware: 22.2%
- Live lock
- Infinite loop
- False BUG_ON
- General fault
- Run out of resource
- Dead lock
3. The Consequences of Attack

- Host DoS: more than 60%
  All DoS: more than 70%

  - Guest to guest attack
    Some guest app leverages hypervisor to DoS its own guest VM
Summary: Observations

• **Hypervisors have bugs**
  – Some previous studies focused on bugs of dom-0 or host OS
  – Some systems (e.g., nested virtualization) can solve the problem but may cause performance overhead due to nested levels

• **Some components have more vulnerabilities (found)**
  – Take consideration on mem management, code emulation, etc.

• **DoS cannot be ignored**
  – Need to tolerant DoS for availability
Deconstruction for Isolation

NEXEN: NESTED XEN

It’s a palindrome!
From Observations to Nexen

• **Hypervisors have bugs**
  – Deconstruct the hypervisor to isolated components
  – “Nesting” within single hardware privilege for performance

• **Some components have more vulnerabilities (found)**
  – Isolate vulnerabilities in the boundary of VM

• **DoS cannot be ignored**
  – Isolate failure in the boundary of VM
Deconstructing Xen

Partition Xen into several internal domains, all the domains run in the same hardware privilege.
Xen Slice

Each Xen slice serves only one VM, containing the VM’s metadata and handling its VMExits.
Only one shared service. It does not interact directly with VM, just serves Xen slices.
Xen Destruction

• Questions
  – Which parts to put in Xen slices?
  – Which parts to put in shared service?

• Principles
  – Least privilege
  – Minimize runtime communication
  – Separate mechanism from policy
<table>
<thead>
<tr>
<th>Component</th>
<th>Original Xen</th>
<th>Vulnerabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory management</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>CPU virtualization</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>Code emulation</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>I/O</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>Exception handling</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>Grant table</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>Global</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>Domain control</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>Domain building</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>Event channel</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>XSM</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>Scheduler</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>Others</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>Component</td>
<td>Xen Slice</td>
<td>Shared Service</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------</td>
<td>----------------</td>
</tr>
<tr>
<td>Memory management</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU virtualization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code emulation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I/O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exception handling</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Grant table</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Global</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Domain control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Domain building</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Event channel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XSM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scheduler</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Others</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The **security monitor** controls guest page tables and EPTs. It offers interfaces & does security checks.
Protecting the Security Monitor

• MMU virtualization
  – Get higher software privilege in the same hardware privilege
  – Similar with the nested-kernel architecture [ASPLOS’16]

• Only the monitor can modify page tables
  – Page tables are mapped as read-only to other components
  – No page table operation instructions out of the monitor
  – Enforce security policies on each operation of page table
  – Bootstrap security: through Intel TXT or TPM
Same Memory, Different Views

- **Self-VM**
  - Read/Write (RW)

- **Other VM**
  - Read/Write (RW)

- **Guest VM**
  - Read (RO)

- **Xen Slice**
  - Read (RO)

- **Shared Service**
  - Not mapped

- **Secure Monitor**
  - Not mapped

- **Object Viewer**
  - Guest VM
  - Xen Slice
  - Shared Service
  - Security Monitor
Call Gate: Intercept Switches between Slices

Intercept switches between Xen slices & shared service, as well as VM & its Xen slice.
## Summary: What Nexen can/cannot Defend?

<table>
<thead>
<tr>
<th>Malicious Component</th>
<th>Steal or tamper with VM’s data</th>
<th>Host DoS</th>
<th>Guest DoS</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM (user)</td>
<td>N.A.</td>
<td>Considered</td>
<td>Considered</td>
</tr>
<tr>
<td>VM (kernel)</td>
<td>Not considered</td>
<td>Considered</td>
<td>N.A.</td>
</tr>
<tr>
<td>Other VM</td>
<td>Considered</td>
<td>Considered</td>
<td>Considered</td>
</tr>
<tr>
<td>Xen Slice</td>
<td>Considered</td>
<td>Considered</td>
<td>Not considered</td>
</tr>
<tr>
<td>Shared Service</td>
<td>Considered</td>
<td>Not considered</td>
<td>Not considered</td>
</tr>
</tbody>
</table>

Nexen cannot defend against attacks through legal interfaces (aka., Iago attack)
Security & Performance

EVALUATION
Security Evaluation on 144 XSAs

107/144 (74%): Defended

10/144 (7%): attack through interface, depends on semantic

27/144 (19%): target the shared service and can cause host failure
Case Study: XSA-108

- **Type:** Out-of-boundary mem access in code emulation causes info leak

- **Description**
  - Xen’s code emulation for APIC erroneously emulates read and write permissions for 1024 MSRs where there are actually 256 MSRs. A read operation can go beyond the page set up and potentially get sensitive data from the hypervisor or other VMs

```c
- case MSR_IA32_APICBASE_MSR ... MSR_IA32_APICBASE_MSR + 0x3ff: + case MSR_IA32_APICBASE_MSR ... MSR_IA32_APICBASE_MSR + 0xff:
```
SPEC CPU2006 (less than 1%)
IOzone (2.4% on average)
Conclusion

- **Methodology of deconstruction**
  - Analyze 201 Xen’s vulnerabilities
  - Derive boundary of isolation from the result
  - Deconstructing system to internal domains and security monitor

- **Nexen implementation**
  - Deconstruct Xen to multiple *Xen slices* and one *shared service*
  - Using nested kernel design to protect the *security monitor*

More info: [http://ipads.se.sjtu.edu.cn/xsa](http://ipads.se.sjtu.edu.cn/xsa)

Thanks!
BACKUP SLIDES
Same Memory, Different Views

- Secure Monitor
- Xen Slice Code
- Shared Service Code
- Xen Slice Data
- ... 
- Xen Slice Data
- Shared Service Data

Xen Slice 1

Xen Slice n

Shared Service

Memory Space
<table>
<thead>
<tr>
<th>Target</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory management</td>
<td>25.69%</td>
</tr>
<tr>
<td>CPU virtualization</td>
<td>21.53%</td>
</tr>
<tr>
<td>Code emulation</td>
<td>13.19%</td>
</tr>
<tr>
<td>I/O</td>
<td>9.03%</td>
</tr>
<tr>
<td>Exception handling</td>
<td>5.56%</td>
</tr>
<tr>
<td>Grant table</td>
<td>4.86%</td>
</tr>
<tr>
<td>Global</td>
<td>4.17%</td>
</tr>
<tr>
<td>Domain control</td>
<td>4.17%</td>
</tr>
<tr>
<td>Domain building</td>
<td>3.47%</td>
</tr>
<tr>
<td>Event channel</td>
<td>2.08%</td>
</tr>
<tr>
<td>XSM</td>
<td>1.39%</td>
</tr>
<tr>
<td>Scheduler</td>
<td>0.69%</td>
</tr>
<tr>
<td>Others</td>
<td>3.47%</td>
</tr>
</tbody>
</table>
The Control Flow

- Gate keeper in the monitor
  - Switch between memory spaces
- Intercept transferring between:
  - Guest VM & Hypervisor
  - Xen slice & shared service
- Complete mediation
  - Cannot be bypassed
Case Study: XSA-191

• Type
  – Misuse of H/W feature in code emulation causes privilege escalation to guest kernel

• Description
  – Intel hardware uses NULL segment selectors to prevent access. Xen code emulator fails to check this condition and may erroneously permit invalid access. An unprivileged guest user program may be able to elevate its privilege to that of the guest operating system
Case Study: XSA-191

• How to trigger?
  1. try to set kernel data segment selector to NULL
  2. trigger an instruction that requires emulation, the side effect of which changes an entry of kernel page table
  3. the instruction emulated, changing the page table entry, giving the user program access to some kernel data
Case Study: XSA-191

• Why cannot defend?
  – Not harming other VMs: the process completely finish in code emulator of one VM
  – lago attack: logic error of code emulator
Performance Evaluation: Negligible Overhead

SPEC CPU2006 (less than 1%)

IOzone (2.4% on average)
Case Study: XSA-83

• Type
  – Memory corruption in shared service causes privilege escalation

• Description
  – Out-of-memory condition yielding memory corruption during IRQ setup. When setting up the IRQ for a passed through physical device, a flaw in the error handling could result in a memory allocation being used after it is freed, and then freed a second time
Case Study: XSA-83

- **Patch**

```c
@@ -1590,8 +1590,7 @@ int pirq_guest_bind(struct vcpu *v, struct
    printk(KERN_WARNING "Cannot bind IRQ%d to dom%d. Out of memory.\n",
            pirq->pirq, v->domain->domain_id);
-
-    rc = -ENOMEM;
-    goto out;
+    return -ENOMEM;

```
Case Study: XSA-83

• Why cannot Nexen defend?
  – Since the shared service is critical in Nexen, exploiting a bug in this part will allow the attacker to do almost anything destructive towards the whole system
  – VM’s data are still protected
## Comparing with Related Work

<table>
<thead>
<tr>
<th></th>
<th>Hypervisor illegally accesses guest’s data</th>
<th>Guest causes host DoS</th>
<th>Guest apps attack its own VM by hypervisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disaggregated Xen</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Xoar</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Turtles KVM</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DeHype</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>HyperLock</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CloudVisor</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Nexen</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Comparing with Related Work

<table>
<thead>
<tr>
<th></th>
<th>Xen</th>
<th>KVM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(a) Xen</td>
<td>(f) KVM</td>
</tr>
<tr>
<td></td>
<td>(b) Disaggregated Xen</td>
<td>(g) Turtles KVM</td>
</tr>
<tr>
<td></td>
<td>(c) Xoar</td>
<td>(h) DeHype</td>
</tr>
<tr>
<td></td>
<td>(d) CloudVisor</td>
<td>(i) HyperLock</td>
</tr>
<tr>
<td></td>
<td>(e) Nexen</td>
<td></td>
</tr>
</tbody>
</table>

- **Xen**
  - VM-1
  - VM-2
  - Dom0
  - Xen
  - (a) Xen
  - (b) Disaggregated Xen
  - (c) Xoar
  - (d) CloudVisor
  - (e) Nexen

- **KVM**
  - VM-1
  - VM-2
  - Qemu
  - Linux+KVM
  - L1: Qemu
  - L2: Linux+KVM
  - (f) KVM
  - (g) Turtles KVM
  - (h) DeHype
  - (i) HyperLock

- Trusted Component

- Dom0
- CloudVisor
- Secure Monitor
- ShdSrv
- Slice-1
- Slice-2

(40 27)
Internal Domain API

• Domains interaction
  – Create a Xen slice
  – Allocate protected memory to a Xen slice
  – Specify policy for a piece of memory
Case Study: XSA-111

• Type
  – False BUG_ON in CPU virtualization causes host DoS

• Description
  – A piece of hypercall parameter translation code assumes that only the lower 32 bits of a 64-bit register variable are used, violation of which will trigger a BUG_ON that kills the hypervisor
Case Study: XSA-111

• How to trigger?
  – This condition can be deliberately violated by an HVM guest by temporarily changing to 64-bit mode and passing an invalid 64-bit parameter

```c
int hypercall_xlat_continuation(unsigned int *id, unsigned int nr, unsigned int mask, ...) {
  ...
  regs = guest_cpu_user_regs();
  ...
  BUG_ON(*reg != (unsigned int)*reg);
}
```
Case Study: XSA-111

• How to defend?
  – In Nexen, the vulnerable code runs in the context of a Xen slice
  – The modified BUG_ON logic will only kill current Xen slice VM when it is triggered