Return-Oriented Programming
Attack on the Xen Hypervisor

2012 Seventh International Conference on Availability, Reliability, and Security (3rd Tier)

Authors: Baozeng Ding, Yanjun Wu, Yeping He, Shuo Tian, Bei Guan, Guowei Wu

by Rodney D. Lykins
Agenda

• Xen Hypervisor
• Return-Oriented Programming (ROP)
• ROP Attack on Xen
• Hardening Hypervisors
• Article Issues
Xen Hypervisor

- ~350K lines of code for v4.1.2
- About same as Space Shuttle
- Windows 7 = ~40M lines
- Runs virtual machines in environments known as domains
- Privileged (dom0) versus unprivileged (domU)
- Privilege controlled by Boolean value in domain structure
Why Return Oriented Programming?

- Compiler, operating system, and anti-virus mitigations are making traditional exploit techniques more difficult

- Current hardware defenses are ineffective against Return-Oriented Programming

- Microsoft 2012 Blue Hat Top 3 Finalists were ROP mitigations

- Real-World Attacks
  - Sequoia AVC Voting Machine
  - Adobe Reader
  - Adobe Flash Player
  - Quicktime Player
  - Apple iPhone Jailbreak

Black Hat 2010: Dino A. Dai Zovi
How Did We Get Here?

- **Buffer Overflows**
  - 30+ years old
  - Subvert usual program execution control flow
  - Stack Smashing, Format String, Heap and Integer Overflows
  - Typically leads to code injection to launch a shell

- **WEX**
  - Mark all writable memory locations as non-executable (i.e. Microsoft’s Data Execution Prevention)
    - Blocks all code injection exploits
  - Hardware Support
    - AMD “NX” bit, Intel “XD” bit
    - Makes memory pages non-executable
How Did We Get Here? (cont’d)

• **Return-to-libc Attack**
  – Instead of injecting code, use existing code, and subvert usual execution flow by redirecting it to function in linked system libraries
  – The C Library: libc
    • Linked to almost every Unix program
    • Defines system calls and other basic functions, e.g., system()
  – Limitations
    • Relies on functions in libc
    • No branching is possible

• **Address-Space Layout Randomization (ASLR)**
  – Randomizes base address of memory segments
  – Limitations
    • Brute force attacks
    • Incompatible libraries not randomized
    • …or at least maintain consistent offsets
Martians invade earth

Incredible as it may seem, it has been confirmed that a large martian invasion fleet has landed on earth tonight.

First vessels were sighted over Great Britain, Denmark and Norway already in the late evening from where, as further reports indicate, the fleet headed towards the North Pole and Santa Claus was taken hostage by the invaders.

Afterwards they split apart in order to approach most of the major cities around the world. The streets filled as thousands fled their homes, many only wearing their pajamas...
ROP Overview

- Disassemble code to find useful code sequences; chain together to make gadgets to perform desired operations
  - Gadgets: short instruction sequences ending in RET (0xC3)

- Turing Complete without:
  - No malicious code injection
  - No original code modification
  - No library function calls

- Architectures affected:
  - Intel x86, SPARC, ARM, PowerPC, etc.

- Limitation: still requires initial control of EIP…
Finding Gadgets

- Any sequence ending in RET (0xC3)
- Recursively look back from each RET:
  - Preceding bytes valid instruction?
- Finding Gadgets: Mona
  - Released by Corelan in 2011
  - PyCommand
    - Immunity Debugger
    - WinDBG
- Other Gadget Finding Tools
  - Msfrop (Metasploit)
  - ROPME (used in article)
  - ROPgadget
  - OptiROP

Example: RM2MP3Converter.exe

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x004025A8</td>
<td>5F</td>
<td>POP EDI</td>
</tr>
<tr>
<td>0x004025A9</td>
<td>5E</td>
<td>POP ESI</td>
</tr>
<tr>
<td>0x004025AA</td>
<td>5D</td>
<td>POP EBP</td>
</tr>
<tr>
<td>0x004025AB</td>
<td>5B</td>
<td>POP EBX</td>
</tr>
<tr>
<td>0x004025AC</td>
<td>83C4</td>
<td>ADD ESP, 0C</td>
</tr>
<tr>
<td>0x004025AF</td>
<td>C3</td>
<td>RETN</td>
</tr>
</tbody>
</table>

Result: 5 potential gadgets
Unintended Instruction Sequences

• Variable-length instructions
  – 1 to 20 bytes long

• Unaligned memory accesses
  – Address not divisible by 32 or 64-bits

• Jump into an arbitrary position of a valid instruction
  – Results in a new instruction sequence not intended by programmer or compiler

Decoding a byte sequence at different offsets yields different instructions
Using Gadgets

• Many semantically equivalent gadgets available to perform same action:
  – Example: put 0x40 (64) into EDX

<table>
<thead>
<tr>
<th>GIVEN</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>If EDX = 0</td>
<td>INC EDX (64X)</td>
</tr>
<tr>
<td>If EDX = 68</td>
<td>DEC EDX (4X)</td>
</tr>
<tr>
<td>If EDX = 3 and EBX = 64</td>
<td>XCHG EDX, EBX</td>
</tr>
<tr>
<td>If EDX = 3 and EBX = 64</td>
<td>MOV EDX, EBX</td>
</tr>
</tbody>
</table>

• Compensate for non-required instructions in middle of useful gadget

<table>
<thead>
<tr>
<th>Desired:</th>
<th># INC EDX # RETN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Found:</td>
<td># INC EDX # STD # POP ESI # POP EDI # POP EBX # RETN</td>
</tr>
<tr>
<td>Solved:</td>
<td>Compensate by pushing dummy value on stack (i.e., 0x41414141)</td>
</tr>
</tbody>
</table>

• Avoid modifying registers that already have desired values

• Avoid NULL bytes
So...back to the article...
ROP Attack on Xen

• Goal: escalate the privilege of domU to dom0 level

• Setup
  • domain_id: the identification of the domain  //offset is 0
  • next_in_list: used to link domain structures together  //offset is 0x44
  • is_privileged: Boolean field for privilege identification  //offset is 0x9E

• Approach
  • Identify address of the domain structure of vulnerable domU
  • Get offset of is_privileged field in the structure

• Implementation
  • x86 architecture running Xen 4.1.2 with dom0/domUs both using Fedora 8, Kernel 2.6.18
  • Develop vulnerable hypercall with stack-based buffer overflow
  • Use ROPME to find useful instruction sequences (i.e., gadgets); found 23,674 gadgets
  • Construct payload
Payload Construction

- dom0 in memory at 0xFF2FF080

  1. Get dom0
     - movl (0xFF255080), %eax

  2. Get domain struct of domU
     - movl 0x44(%eax), %eax

  3. Set is_privileged = 1
     - movb 0x1, 0x9e(%eax)

<table>
<thead>
<tr>
<th>Address</th>
<th>Inst. Seq.</th>
<th>Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFF122DE3</td>
<td>58</td>
<td>pop %eax</td>
</tr>
<tr>
<td>04 8B</td>
<td></td>
<td>addb a1, 0x8B</td>
</tr>
<tr>
<td>08 F6</td>
<td></td>
<td>orb dh,dh</td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td>ret</td>
</tr>
<tr>
<td>0xFF1DEA31</td>
<td>8B 40 18</td>
<td>movl 0x18(%eax), %eax</td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td>ret</td>
</tr>
<tr>
<td>0xFF12DCA7</td>
<td>89 C1</td>
<td>movl %eax, %ecx</td>
</tr>
<tr>
<td>89 C8</td>
<td></td>
<td>movl %ecx, %eax</td>
</tr>
<tr>
<td>5D</td>
<td></td>
<td>pop ebp</td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td>ret</td>
</tr>
<tr>
<td>0xFF188C51</td>
<td>2D 82 FA FF C9</td>
<td>subl 0xC9FFFA82, %eax</td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td>ret</td>
</tr>
<tr>
<td>0xFF124FDF</td>
<td>01 C8</td>
<td>addl %ecx, %eax</td>
</tr>
<tr>
<td>5D</td>
<td></td>
<td>pop %ebp</td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td>ret</td>
</tr>
<tr>
<td>0xFF1E2B46</td>
<td>FE 01</td>
<td>inc (%ecx)</td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td>ret</td>
</tr>
</tbody>
</table>
Hardening Hypervisors

- Trustvisor
- Bitvisor
- Disaggregation
- NOVA
- HyperGuard
- HyperCheck
- HyperSentry
- HyperSafe
Issues With Article

• Xen has 350K lines of code (Libc has only 200K)
  • Of course gadgets would be available!
• Authors developed vulnerable hypercall
  • Vulnerability not organic to Xen
• Misinformation regarding ROP mitigation strategies
Questions ???