Anti-VM Techniques
Anti-VM Techniques

- Virtual machines initially used only by malware analysts
  - Malware benefited from detecting VM (especially VMware) and shutting down to escape analysis
- Normal users also use VM
  - Rollback recovery easy
  - Portability
- Malware also wants to infect VMs
  - Anti-VM techniques become less popular
VM Artifacts

- Process listing
  - 3 VM processes are running – easy to see

- Filesystem – VM installation directory
  - (e.g. C:\Program Files\VMware\VMware Tools)

- Registry keys – hardware (mouse, adapters)

- Networking
  - MAC addresses assigned for use by IEEE for VMware NICs begin with 00:0C:29 (virtual NIC)
  - Malware checking version of hardware, MAC

- Memory (search string VMware through physical memory)
<table>
<thead>
<tr>
<th>Company and Products</th>
<th>MAC unique identifier(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMware ESX 3, Server, Workstation, Player</td>
<td>00-50-56, 00-0C-29, 00-05-69</td>
</tr>
<tr>
<td>Microsoft Hyper-V, Virtual Server, Virtual PC</td>
<td>00-03-FF</td>
</tr>
<tr>
<td>Parallels Desktop, Workstation, Server, Virtuozzo</td>
<td>00-1C-42</td>
</tr>
<tr>
<td>Virtual Iron 4</td>
<td>00-0F-4B</td>
</tr>
<tr>
<td>Red Hat Xen</td>
<td>00-16-3E</td>
</tr>
<tr>
<td>Oracle VM</td>
<td>00-16-3E</td>
</tr>
<tr>
<td>XenSource</td>
<td>00-16-3E</td>
</tr>
<tr>
<td>Novell Xen</td>
<td>00-16-3E</td>
</tr>
<tr>
<td>Sun xVM VirtualBox</td>
<td>08-00-27</td>
</tr>
</tbody>
</table>
Bypassing Vmware Detection

- Malware scans process list for VMwareTray.exe – if found, jump to exit code
- Counter-measures
  - Patch the binary to avoid the jump
  - Use hex editor to modify the name VMwareTray.exe so it will compare something else
Existing DLLs and Sys

- **VMware**
  - C:\windows\System32\Drivers\Vmmouse.sys
  - C:\windows\System32\Drivers\vm3dgl.dll
  - C:\windows\System32\Drivers\vmdum.dll
  - C:\windows\System32\Drivers\vm3dver.dll
  - C:\windows\System32\Drivers\vmtouch.dll
  - C:\windows\System32\Drivers\vmmousever.dll
  - C:\windows\System32\Drivers\vmmouse.sys
  - C:\windows\System32\Drivers\vmhgfs.dll
  - C:\windows\System32\Drivers\vmGuestLib.dll
  - C:\windows\System32\Drivers\VmGuestLibJava.dll
  - C:\windows\System32\Drivers\vmhgfs.dll

- **VirtualBox**
  - C:\windows\System32\Drivers\VBoxMouse.sys
  - C:\windows\System32\Drivers\VBoxGuest.sys
  - C:\windows\System32\Drivers\VBoxSF.sys
  - C:\windows\System32\Drivers\VBoxVideo.sys
  - C:\windows\System32\Drivers\vboxdisp.dll
  - C:\windows\System32\Drivers\vboxhook.dll
  - C:\windows\System32\Drivers\vboxmonitor.dll
  - C:\windows\System32\Drivers\vboxogl.dll
  - C:\windows\System32\Drivers\vboxoglarrayspu.dll
  - C:\windows\System32\Drivers\vboxoglcrutil.dll
  - C:\windows\System32\Drivers\vboxoglerrorspu.dll
  - C:\windows\System32\Drivers\vboxoglfeedbackspu.dll
  - C:\windows\System32\Drivers\vboxoglpackspu.dll
  - C:\windows\System32\Drivers\vboxoglpasssthroughspu.dll
  - C:\windows\System32\Drivers\vboxservice.exe
  - C:\windows\System32\Drivers\vboxtrak.sys
## Interrupt Descriptor Table

- Data structure – x86 implements an interrupt vector table
- Determine the response to interrupts and exceptions

<table>
<thead>
<tr>
<th>Interrupt ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>divide error</td>
</tr>
<tr>
<td>1</td>
<td>debug exception</td>
</tr>
<tr>
<td>2</td>
<td>NMI interrupt</td>
</tr>
<tr>
<td>3</td>
<td>Breakpoint</td>
</tr>
<tr>
<td>4</td>
<td>INTO-detected Overflow</td>
</tr>
<tr>
<td>5</td>
<td>BOUND range exceeded</td>
</tr>
<tr>
<td>6</td>
<td>Invalid opcode</td>
</tr>
<tr>
<td>7</td>
<td>coprocessor not available</td>
</tr>
<tr>
<td>8</td>
<td>double fault</td>
</tr>
<tr>
<td>9</td>
<td>coprocessor segment overrun</td>
</tr>
<tr>
<td>10</td>
<td>invalid task state segment</td>
</tr>
<tr>
<td>11</td>
<td>segment not present</td>
</tr>
<tr>
<td>12</td>
<td>stack fault</td>
</tr>
<tr>
<td>13</td>
<td>general protection</td>
</tr>
<tr>
<td>14</td>
<td>page fault</td>
</tr>
<tr>
<td>15</td>
<td>reserved</td>
</tr>
<tr>
<td>16</td>
<td>coprocessor error reserved</td>
</tr>
<tr>
<td>17-31</td>
<td>maskable interrupts</td>
</tr>
<tr>
<td>32-255</td>
<td>maskable interrupts</td>
</tr>
</tbody>
</table>
Vulnerable Instructions

- One IDTR for VM, one for host? Conflict
- VM will move the IDTR to a different memory location – check whether it is “virtual” or “physical”.
- Detect VM by Descriptor Table Instructions
  - Guest VM must have a different location for these tables than Host VM
    - VM software creates separate locations
    - But, Guest VM can directly execute x86 instructions that directly access underlying registers to check for inconsistency
    - Use these registers to detect the presence of VM (discussed next)
Movie Matrix: Neo has to choose between a blue and red pill.
Blue Pill: fake reality – live in ignorance. – VM is not detectable.
Red Pill: real reality (sidt – store interrupt descriptor table register in a new destination operand)
x86 instruction sidt loads value of IDTR
Virtual machine monitor must relocate the guest IDTR to avoid conflict with host IDTR
Check if VM runs the sidt instruction
Cmp al, 0FFh -> the VM signature (fifth byte of memory location)
Only works on single-processor machine – multi-core processor each processor has an IDT so Red Pill is not reliable
Querying I/O port

- VMware *virtualizes I/O ports* – weakness
  - Communication between guest OS and host.
  - Check virtual I/O port is working? Use the *in* (0xED) instruction.
  - *in 0xED* - Copies data from I/O port from source operand to a memory location.
  - Capture I/O destined for the communication channel port 0x5668 (VX)
  - **Magic number 0x564D5868 (VMXh) – EAX register is loaded with magic number;** 
    ECX loaded with action you want to perform – 0xA – get VMware version type; 0x14 will get memory size. ECX will contain the VM type

- Common Anti-VM instructions
  - *sidt, sgdt, sldt, smsw, str, in, cupid* – replace by *NOP*
  - Patching the binary to avoid calling these instructions
  - **Use Multiprocessor (multiple IDTR) – Red Pill Test work for single processor**
  - 20 instructions designated by VMware as “not virtualizable”
  - Malware will only use them if checking for VMs