Recent Advances in VMM Support for Security

David Lie
University of Toronto
Talk Outline

1. What is a VMM?
   - VMM basics
   - What strengths does a VMM have?

2. VMM Security Solutions Overview

3. Improved Monitoring and Detection

4. Secure Isolation of System Components
What is a VMM?

- A Virtual Machine Monitor (VMM) is a thin software layer between OS and hardware
  - Virtualizes the hardware interface so you can run multiple Guest OSs
  - Virtual Machines strongly isolated, can only communicate via network
  - VMM = hypervisor
Impact of VMMs

• VMMs are increasingly becoming common place:
  – Estimated billion dollar market, by 2010, it is estimated that 15% of servers will be running on a VMM
  – Many companies offering technologies (VMware, Microsoft, Virtual Iron, Parallels, XenSource/Citrix, IBM, HP, Sun, SGI…)

• VMMs virtualize the hardware/OS interface:
  – Decoupling the OS from the underlying hardware
  – This allows OS images to be cloned, moved, suspended and restarted
  – Allows multiple OS images to be run on a single hardware instance
Security Strengths of VMMs

VMMs have two characteristics that are beneficial for security:

1. **Simpler and Smaller**:
   - Operating systems have millions of lines of code (LOC). (Windows XP = ~40M LOC, Linux 2.6.18 kernel = ~5M LOC).
   - VMMs can be as small as several 100k lines (Xen 3.1 = ~200K LOC).
   - VMMs also provide less functionality: mimic hardware interface and provide decent performance.
   - Fewer bugs: flaws have been found in VMMs, but they are far fewer in number than in OS code.

2. **Provide Strong Isolation between VMMs**:
   - VMMs isolate software in different VMMs without having to incur costs of extra hardware.
   - VMMs provide *stronger isolation guarantees between VMs than OSs do between processes*. 
Talk Outline

1. What is a VMM?

2. VMM Security Solutions Overview
   - Why might VMMs be good for enhancing security?

3. Improved Monitoring and Detection
   - ISIS: VMM-based IDS
   - Patagonix: Detecting Root-kits and other hidden malware

4. Secure Isolation of System Components
Can VMMs improve security?

• Much concern these days over the security of VMMs:
  – VMM companies like to claim their product is secure
  – While no software is bug-free, some evidence seems to support this:

<table>
<thead>
<tr>
<th>CVE vulns since 2003</th>
<th>VMware ESX</th>
<th>Xen</th>
<th>Windows XP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>13</td>
<td>215</td>
<td>6</td>
</tr>
</tbody>
</table>

• On the other hand, VMMs can also increase security:
  1. Better monitoring:
     • VMMs have complete visibility, but good isolation from Guest OSs.
     • Good for intrusion detect, auditing, etc….
  2. Component isolation:
     • Better protection for sensitive data and programs

• VMMs have the potential to take back the advantage from attackers
Who’s going to win?

- Instead of all-powerful VMM-based malware, how about all-power VMM-based security solutions?
  - Lets use our unfair advantage!
Better Monitoring

• “One-way mirror”:
  – The VMM is at a higher privilege level than the guest OS.
  – Thus the VMM may monitor the guest OS, but unaffected by malware in the guest OS.
  – Can monitor from a secure position.
ISIS is a system that implements a monitor in the Xen VMM
- Allows specific “triggers” to be inserted into the monitored kernel
- When trigger is executed, it triggers a sensor that can inspect the state of the kernel
- Inspecting the state of the VM kernel is called *introspection*
Writing Sensors

Linux Source Code:

```c
798 long sys_open(const char *filename, int flags, int mode)
799 {
... 806  tmp = getname(filename);
```

Sensor Code:

```c
tmp_addr = get_value("tmp", vm_id, regs);
    tmp = read_str(vm_id, tmp_addr);
    flags = get_value("flags", vm_id, regs);
    if (!strcmp(tmp, "/etc/passwd") &&
        ((flags & O_RDWR) || (flags & O_WRONLY))) {
        return COMPROMISED;
    }
```

- Sensors are easy to write. Example:
  - Check if password file opened for writing
- Use symbol names from Linux source code in sensor:
  - Symbol names are resolved by monitor by examining kernel symbol table
  - Kernel symbol table is delivered securely from copy of kernel image in IDS VM
Other IDS systems

• Many other university research projects taking a similar approach:
  – Introvirt (U. of Michigan)
  – Livewire (Stanford University)

• Difficulties with VMM-based IDS:
  – Attacker cannot disable or tamper with the IDS
  – However, attacker may be able to evade or trick the IDS…
Evading Detection

- The in sensor-based VMM-IDS systems:
  - The attacker can trick the VMM into *misinterpreting* the state of the VMM
  - Example, attacker makes another copy of the `sys_open` function and redirects open calls to that instead. The trigger then is never executed.
Semantic Gap

• Attacker can evade detection by IDS because:
  – IDS doesn’t know everything about the state of the OS, so it must make *assumptions* about the internal structure of the OS and the *semantics* of OS operation
    • Of course the attacker is not bound to these assumptions!

• VMM-IDS must make these assumption because there is a *semantic gap*:
  – This gap exists because the VMM operates *below* the abstractions offered by the OS. The VMM does not know how to *interpret* OS state.
  – In our example, the VMM does not know the true location of the *sys_open* handler. It has to assume the handler is where it would be in a pristine kernel.
Patagonix: Avoiding Assumptions

We can avoid making assumptions about OS structure by only relying on what can be observed across the OS/hardware interface:

Example: Patagonix
- Attacker’s frequently use root-kits to hide executing processes from the sys admin.
- Patagonix can detect and identify all executing binaries on a system even if the OS is compromised (i.e. controlled by a rootkit).
- Patagonix does not make assumptions about the OS kernel structure, instead Patagonix only relies on requests between the OS and the hardware that the VMM intercepts.

Patagonix: Detecting Rootkits

- Patagonix uses the NX-bit in hardware to detect code execution:
  - Processor will invoke Patagonix whenever code is executed
  - Patagonix can then inspect the executing code and see if it can match it against either known legitimate code, or a known rootkit
  - Patagonix can also perform anomaly detection by comparing what the OS reports as running with the executing code pages that Patagonix observes
Patagonix

- Problem is that it is difficult to identify code sometimes:
  - Windows has relocations which mean that the code in memory depends on run-time state of the OS

- Solution is that Patagonix uses an OS agent to provide Patagonix with “hints” about the OS state:
  - These hints are then verified against the running code.
  - Example: the agent provides Patagonix with information about the locations of DLLs
  - Patagonix can then verify that these hints are truthful, any deviation indicates that the OS is not functioning properly (possible compromise)

- Enforcement of white lists:
  - VMM can also enforce what code can execute in the kernel and on the system, even if OS is compromised
Talk Outline

1. What is a VMM?

2. VMM Security Solutions Overview

3. Improved Monitoring and Detection

4. Secure Isolation of System Components
   - Terra: VMM-level isolation
   - Proxos: Using VMMs to control application isolation
Poor isolation

- In current OSs, a sensitive application depends on the OS and every other privileged application on the commodity OS.

- Ideally, we would like to protect sensitive applications from unrelated applications on the same OS.
Using Isolation

• Another use of VMMs is to isolate insecure applications from secure ones. For example the Terra architecture:

<table>
<thead>
<tr>
<th>Open VM</th>
<th>Closed VM</th>
<th>Closed VM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email, Web apps</td>
<td>SETI@Home Client</td>
<td>Online Game (Secure Client)</td>
</tr>
<tr>
<td>Commodity OS</td>
<td>Commodity OS</td>
<td>Specialized OS</td>
</tr>
<tr>
<td></td>
<td>Trusted Virtual Machine Monitor (TVMM)</td>
<td></td>
</tr>
</tbody>
</table>

• Secure applications run along side legacy applications:
  – Secure applications are attested, restrictions are placed upon them to maintain their security
  – VMM strongly isolates the VMMs from each other. Vulnerabilities in the Open VM cannot affect the Closed VMs
Isolation and Communication

- This works if the various applications are independent of each other.
  - However, you may have interaction between applications in the same OS that have different security needs:
  - Examples:
    1. A sensitive program like the remote terminal server (i.e. SSH, RD) invokes a shell and other applications:
       - SSH server has access to keys, password files
       - A shell can be used to do arbitrarily risky things
    2. Web browser needs to invoke a helper application to view a file:
       - The web browser has access to sensitive information like cookies, passwords and browsing history
       - The application may not be secure enough to accept remote content.
3. A web server creates and SSL connection:
   - The web server needs access to SSL certificates to authenticate itself
   - Afterwards, it may invoke vulnerable scripts to generate dynamic content

   • Problem:
     – You can’t take these services and put them in different VMs because the applications need to talk to each other on the same OS
     – For example, putting your remote terminal service on a different VM than your applications makes your remote terminal service useless!!
Solution: Proxos

**Proxy Operating System (Proxos)** routes system calls based on their sensitivity to attack

[Ta-Min et al. OSDI 2006]
Specifying System Call Routing Rules

- Proxos routes system calls based on the name of the resource, and the type of resource being accessed:

```plaintext
DISK:("/etc/shadow", priv_fs)
...
priv_fs = {
    .open = priv_open,
    .close = priv_close,
    .read = priv_read,
    .write = priv_write
}
```

- Rules link a method table (priv_fs) with name of resource (/etc/shadow)
- Method table has pointers to system call handlers in the private OS
- Resources not named in the rules are routed to commodity OS by default

Interface is partitioned into accesses to sensitive and non-sensitive resources
System calls routed to commodity OS using RPC’s:
- During startup, a shared memory region between the commodity OS and Proxos is created
• Proxos allows applications to have access to commodity OS, but isolated sensitive resources at the same time. Ex SSHD Server:
  – Sensitive data such as user passwords and the host key stored in private OS
  – All network packets decrypted in private app before sent to command shell. Command shell is in Commodity OS.
Application: Graphical Web Browser

- Graphical Web Browser:
  - Sensitive user data and user interface is isolated from commodity OS
  - Dillo can save downloaded files and invoke file viewers on the commodity OS
- Server Application: Apache & SSL extension
  - Private signing key isolated from commodity OS
  - To maintain performance, minimize private VM startup/shutdown by making host process persistent
Patch Conflicts

• Proxos can also be used to resolve conflicting applications:
  – We all know that applying patches can result in configuration and application conflicts
  – Proxos can separate conflicting applications so that:
    1. Applications the patch does not conflict with can get patched
    2. Applications that conflict with the patch are isolated from the patch.
Conclusions

• VMMs can be used for security:
  – Natural advantages:
    1. Higher privilege level
    2. Small, and simple = Fewer vulnerabilities
    3. Stronger isolation between protection domains

• Many potential uses:
  – VMM-based IDS (ISIS)
  – VMM monitoring for hidden malware (Patagonix)
  – VMM enforcement of white-lists (Patagonix)
  – VMM-based protection of sensitive applications and Data (Proxos)
  – Transparent patch and configuration conflict resolution (Proxos)