Virtual Machine Based Intrusion Detection

Murat Kantarcioglu

Based on Garfinkel et al. NDSS 2003 work
Intrusion Detection Systems

- Two types
  - Network based Intrusion Detection Systems (NIDS)
    - Resistant against attacks
    - Do not know the individual host states
  - Host based Intrusion Detection System (HIDS)
    - High host visibility
    - Easier to attack.
    - Kernel level HIDS
      - User programs can modify kernel (e.g. sys_call_table) through loadable kernel modules
      - IDS crash could create system vulnerable
Virtual Machine Introspection

• Idea: Use VMM level IDS. Do Virtual Machine introspection to detect attacks.
  – Advantage: Can observe machine states, harder to attack
  – Disadvantage: Potential costs
VMM Capabilities

• VMM will be harder to attack
  – Simpler than traditional OS
    • Does not need to have networking

• Isolation due to VMM
  – IDS and Guest OS will be isolated.

• Inspection
  – VMI IDS can directly inspect the machine state
  – Harder to hide actions

• Interposition
  – VMI IDS can use VMM to be notified when certain events happen
Threat Model for VMI IDS

- **Guest host is not trusted**
  - All info. Gathered from guest host is assumed to be tainted and not to be trusted.

- **VMM is trusted**

- **VMI IDS has some assumptions about the structure of the guest OS in order to implement some IDS policies.**
Design Goals

- Low overhead
  - Only monitor events that are closely related to intrusions (e.g., sensitive memory modifications etc.)
- Minimize change to VMM
  - VMM should be kept simple and bug free.
- Limit VMM exposure
  - IDS and VMM could be kept separate.
  - IDS compromise must not affect the VMM security
VMI IDS Design
VMM Interface

- **Inspection Commands**
  - Inspect state info such as memory, register, I/O devices

- **Monitor Commands**
  - Get notification for certain event occurrence.

- **Administrative Commands**
  - Allows the control of VM
    - Stop the VM if intrusion is detected.
OS Interface Library

- Provides necessary functionality to translate VMM states to OS level semantics
  - E.g., display the content of the task structure for PID 231.
Policy Engine

- Interprets the system state and event from VMM Interface and OS library interface
- Implements various policies such as burglar alarm, misuse detection, integrity checkers etc.
- Provides policy engine for more complex detection.
Example Policies:

- Polling policy modules
  - Check for activities in a certain time intervals
- Lie detectors
  - See whether guest os lies about the OS parameters
  - E.g., check whether what ps returns is consistent with what VMM observes
- User program integrity detector
  - Make sure the images in memory not modified
- Signature detector
- Raw socket detector (burglar alarm)
Event Driven Policy Modules

- Detecting tampering with OS code segment
  - Mark sensitive OS parts read only
  - Use copy-on-write mechanism to detect changes
- NIC access enforcer
  - Detect Ethernet device with promiscuous mode on.
## Sample Attacks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>nic</th>
<th>raw</th>
<th>sig</th>
<th>int</th>
<th>lie</th>
<th>mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>cdoor</td>
<td>Stealth user level remote backdoor</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t0rn</td>
<td>Precompiled user level rootkit</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ramen</td>
<td>Linux Worm</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lrk3</td>
<td>Source based user level rootkit</td>
<td>P</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>knark-0.59</td>
<td>LKM based kernel backdoor/rootkit</td>
<td>D</td>
<td></td>
<td>D</td>
<td></td>
<td></td>
<td>P</td>
</tr>
<tr>
<td>adore-0.42</td>
<td>LKM based kernel backdoor/rootkit</td>
<td>D</td>
<td></td>
<td>D</td>
<td></td>
<td></td>
<td>P</td>
</tr>
<tr>
<td>dsniff 2.4</td>
<td>All-purpose packet sniffer for switched networks</td>
<td>P</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td>P</td>
</tr>
<tr>
<td>SUCKIT</td>
<td>/dev/kmem patching based kernel backdoor</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>P</td>
</tr>
</tbody>
</table>
Performance Overhead

Figure 2. Performance of Polling Policy Modules
Potential Issues

- OS library interface is complex and could be evaded.
  - In Limewire OS interface library is run as a separate OS.
- VMM existence could be detected
  - Check I/O performance time
- VMM could be subverted.
- Policy Engine could be attacked.
  - Sanitize inputs
  - Simpler High level policy language
  - Failing closed (suspend VMM if something goes wrong)
  - Potential bugs?
Such systems could be used for auditing purposes in the cloud.
Performance overhead is important.
IDS typically have false positive issues.
Complex attacks may be harder to detect—(slowly stealing user private information)