Building Trustworthy Intrusion Detection Through Virtual Machine Introspection

Fabrizio Baiardi\textsuperscript{1} \quad Daniele Sgandurra\textsuperscript{2}

\textsuperscript{1}Polo G. Marconi - La Spezia, University of Pisa

\textsuperscript{2}Department of Computer Science, University of Pisa

IAS Conference, 2007
Outline

1 Problem
   - Attacks and Evasion of Controls

2 Overall Architecture
   - Virtual Machine Introspection
   - Psyco-Virt

3 Evaluation
   - Security Evaluation
   - Performance

4 Conclusion
   - Results and Future Works
Rootkits have become more sophisticated over the years.

- **User-level** rootkits: usually, modify system binaries.
- **Kernel-level** rootkits: for example, a module inserted into the kernel.

Unfortunately, rootkits and IDSes work at the same level. A rootkit can attack or evade the IDS controls.
Virtual Machine Introspection: Standford University.

- **Visibility**: access VM’s state from a **lower level**.
- **Robustness**: detect intrusions from another VM.

![Diagram of Virtual Machine Introspection]

**Proposed Approach**

**Virtual Machine Introspection**

Monitored VM

![Diagram of Virtual Machine Introspection]

**Monitor**

**Control Interface**

**Virtual Machine Monitor**

**Virtual Machine Introspection VM**

**Introspection**
Semantic Problem

- How to detect **intrusions/attacks** inside the VM?
- **Semantic problem**: the data accessed through introspection are **raw data**.
- We also need to **protect** the IDS.
Solution #1

Modify an IDS to work at the hardware level.

[Diagram showing the architecture of a virtual machine with monitoring and introspection components.]
Solution #2

Build a complex introspection library to export an OS view of the VM’s state.
Our Solution: a Multi-Level Approach

1. **Build a simple introspection library** to check the kernel.
2. **Extend the kernel** to monitor the IDSes inside the monitored VM.
3. **Use standard IDSes** to detect attacks against the VM.
Chain of Trust

- **Level 3**: chkrootkit, Snort, Tripwire, ...
- **Level 2**: Extensions to the kernel, SELinux, LIDS, ...
- **Level 1**: idT/syscall table integrity, text area, list open files, ...

**Monitored VM**
- File System
- Network
- HIDS
- NIDS

**Introspection VM**
- INTROSPECTOR
- SIMPLE INTROSPECTION LIBRARY

**Virtual Machine Monitor**
- CONTROL INTERFACE

**HARDWARE**
Psyco-Virt merges Host and Network IDSes with VMI.

- The first prototype is written in C, based on Xen.
- **Introspection VM**: monitors all the VMs.
- **Monitored VM**: executes the system to be monitored.
- **Control Network**: to exchange the alerts and commands among the VMs.
Introspection VM: monitors all the VMs.

- The **introspector** protects kernel integrity.
- The **director**:
  1. collects the **alerts**;
  2. executes **actions** in response to an alert: stops a VM.
Monitored VM: executes the system to be monitored.

- Runs **IDSes** to detect attacks/intrusions.
- The **collector** receives all the alerts from the local IDSes.
- The **kernel** checks IDS integrity.
Control Network: to exchange the alerts and commands among the VMs.
Currently, Psyco-Virt detects:

- Attacks to the kernel code also those inserting a malicious module.
- Updates to the IDT and syscall table.
- Updates to the text area of a critical processes.
- Replacing ps and lsof.
- Interfaces set into promiscuous mode.
IOzone Read Performance

Overhead is less than 10%.
IOzone Write Performance

Overhead is less than 10%.
Antisniff implemented as a module or through introspection.
Current limitations of the prototype:

- No checks on kernel dynamic data, such as stack.
- Other critical kernel data structures, besides IDT and syscall table, have to be protected.
- Attacks to the VMM.
- Attacking the kernel between each execution of the checks.
Results

- Using **unmodified IDSes with virtual machine introspection**.
- Preventing **evasion** of the controls and **attacks** to IDSes.
- **Multi-Level** approach to form a **chain of trust**:
  1. IDSes.
  2. Kernel.
  3. VMM.
- Acceptable **overhead**.
Future Works

- Checking at runtime memory invariants.
  - Using abstract interpretation of kernel code.
- Tracing a VM, such as using ptrace.
  - Verifying system call parameters.
- Using introspection as an attestation of the VM.
  - Attesting the software to a remote party.