Hypervisor Support for Identifying Covertly Executing Binaries

Lionel Litty$^1$  H. Andrés Lagar-Cavilla$^1$  David Lie$^2$

$^1$Dept. of Computer Science
University of Toronto

$^2$Department of Elec. and Comp. Eng.
University of Toronto

June 22, 2008/ Security Lunch
Outline

1. Quick Overview
2. The Problem
3. Their Solution
   - Patagonix Framework (Figure)
   - Patagonix Framework (Structure)
   - Patagonix Framework (Mechanism)
   - Oracles
4. Discussion
   - Assumptions
   - Limitations
5. Conclusion
Outline

1. Quick Overview
2. The Problem
3. Their Solution
   - Patagonix Framework (Figure)
   - Patagonix Framework (Structure)
   - Patagonix Framework (Mechanism)
   - Oracles
4. Discussion
   - Assumptions
   - Limitations
5. Conclusion
Quick Overview

- Rootkits are hard to detect and remove due to their administrative privilege
- People tried to use hypervisor technique to detect rootkits
- Previous approaches were problematic since they relied on untrustful channels (e.g., source code and symbol information)
- This paper solves the problem by leveraging the capabilities of MMU hardware (NX bit and writable bit)
Outline

1 Quick Overview
2 The Problem
3 Their Solution
   * Patagonix Framework (Figure)
   * Patagonix Framework (Structure)
   * Patagonix Framework (Mechanism)
   * Oracles
4 Discussion
   * Assumptions
   * Limitations
5 Conclusion
Virtual Machines, also known as VMs, or Hypervisors, are useful for detecting stealthy malware with administrative privileges (e.g. the rootkits).

Virtualization brought the challenge of *semantic gap*, (the hypervisor acquires the low-level states of the guest system at the cost of losing the high-level semantic view)

Previous works (e.g. Livewire, Copilot, SBCFI and VMWatch) bridged the gap by leveraging either symbol information or guest system source code, both of which are unbounding.

Rootkits can easily hide themselves from those Hypervisor scanning.
Outline

1. Quick Overview
2. The Problem
3. Their Solution
   - Patagonix Framework (Figure)
   - Patagonix Framework (Structure)
   - Patagonix Framework (Mechanism)
   - Oracles
4. Discussion
   - Assumptions
   - Limitations
5. Conclusion
Figure 1: The Patagonix architecture.
Patagonix is divided into two components: the Patagonix VM component and the Patagonix hypervisor component.

The Patagonix VM component consists of:
- Identity Oracles: identify code pages being executed in the monitored VM
- Management Console: implements the interface between user and Patagonix
- Control Logic: coordinates events between management console, oracles, and hypervisor components

Only the oracles are binary file specific; other parts are OS independent.
Non-Executable (NX) bit (available for all recent AMD and Intel x86 processors)
When set on a virtual page, it traps the processor into the hypervisor component whenever code is executed on that page.

Patagonix initially set the NX-bit and the writable bit on all pages.

After Patagonix hypervisor component receives the trap, it
1. invokes the Patagonix VM to identify the code page;
2. clears the NX-bit;

Subsequent execution on the code page will not cause any trap, unless the page is modified.
The oracle compares the hash values of the executing code page against those from authenticated binary files.

Model the binary loader

\[ L(B, S) = (M, A) \]

Model the oracle for a particular binary loader

\[ O_L(M, A, P) = B \]

The oracle’s task is to search \( MA' \) for any given \((M, A)\), where \( MA' \) contains all page/address combinations that the loader could have generated for all binaries and all legitimate OS states.
Outline

1. Quick Overview
2. The Problem
3. Their Solution
   - Patagonix Framework (Figure)
   - Patagonix Framework (Structure)
   - Patagonix Framework (Mechanism)
   - Oracles
4. Discussion
   - Assumptions
   - Limitations
5. Conclusion
The hypervisor will protect both itself and Patagonix from tampering.

The hypervisor provides a secure communication channel between patagonix and its user.

All legitimate software binaries installed on the OS are known beforehand.
Generally the Patagonix framework has difficulty to deal with self-modifying programs.

But in practice, there indeed are legitimate software which executes self-modifying code. e.g.

- JITs
- Dynamically obfuscated code like winlogon.exe and WGA(Windows Genuine Advantage application)

False Positives: Any code originating from sources other than disk files will cause a suspicion.

Hard to maintain a huge database for all trusted software.
Detecting stealthy rootkits is made possible by using Patagonix, who trust nothing but the hardware, a third party binary database, and the hypervisor.