Low-Observable Physical Host Instrumentation for Malware Analysis

Chad Spensky†, Hongyi Hu§ and Kevin Leach‡

cspensky@cs.ucsb.edu    hongyihu@alum.mit.edu    kjl2y@virginia.edu
lophi@mit.edu

The Network and Distributed System Security Symposium 2016

This work was sponsored by the Assistance Secretary of Defense for Research and Engineering under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the authors and are not necessarily endorsed by the United States Government.

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited.
Outline

- Overview of LO-PHI
- Instrumentation
- Semantic Gap Reconstruction
- Automated Binary Analysis
- Evaluation (Windows Malware)
- Summary
- Demo (Time Permitting)
The Problem

- Binary dynamic analysis is becoming increasingly difficult in security-critical scenarios

  - Environment-aware malware can detect various artifacts exposed by most existing dynamic analysis frameworks and leverage them to avoid detection, or subvert the analysis all together

  - The observer effect, i.e. the effects of the measurement itself, can interfere with the analysis, making the results untrustworthy
    - E.g., software-based instrumentation may result in a different memory layout
The Problem

- **Introspection** techniques offer solutions that have fewer artifacts, but must also bridge the **semantic gap**
  - i.e., translate low-level data to semantically rich output for analysis
Introspection Options

• **Software**
  – Pros: cheap, easy to implement
  – Cons: OS dependent, can affect analysis, easily subverted

• **Virtual machines**
  – Pros: development in software, scalable
  – Cons: easily detectable artifacts (E.g. *Redpill*)

• **Hardware**
  – Pros: potentially very few artifacts, better ground truth
  – Cons: difficult to implement, expensive
• Primary goal
  - *Low- Observable Physical Host Instrumentation (LO-PHI)* aims to obtain ground truth information about a system under test (SUT) while introducing as few artifacts as possible.
Overview

- Zero software-based artifacts

- Simple Python APIs to interact with a system under test
  - Same code for either physical or virtual machines

- A suite of both sensors and actuators

- A suite of semantic-gap reconstruction tools

- Python-based framework for automated binary analysis
  - Analysis “scripts” can be submitted and executed on automatically provisioned machines
Virtual Instrumentation

LO-PHi
System Under Test

block.c

Disk Introspection Server

CPU Physical Memory Map

cpu_physical_memory_map

Memory Introspection Server

Semantic Analysis

UNIX Socket

UNIX Socket
Physical Instrumentation

- Power, Keyboard, Mouse (USB/GPIO)
- Network Tap (Ethernet)
- Disk Introspection (SATA)
- Memory Introspection (PCIe)
- Semantic Analysis
Semantic Gap

• Fictional Hollywood example: *The Matrix*

1. Input Raw Data  
2. Parse Data Structures  
3. Extract Features

• **Memory** (Volatility)
  – Reader raw memory to extract attributes of the system
  – E.g., running processes, kernel modules, descriptor tables

• **Hard Disk** (Sleuthkit)
  – Translate low-level disk activity into file system activities
  – E.g., file creation, deletion, read, write
Stream-based Disk Forensics
Bare Metal

- Multiple layers of abstraction that we must bridge
  - Analog Signal → Digital bits
  - Digital bits → SATA Frames

- SATA Frames → Sector manipulation

- Sector manipulation → File System Manipulation

1. Data Collection
2. Semantic Reconstruction
3. Analysis

Sleuthkit (TSK)
analyzeMFT

SATA Reconstruction

Xilinx ML507 FPGA
**SATA Reconstruction**

A Brief Primer on SATA

- Serial ATA – bus interface that replaces older IDE/ATA standards
- SATA uses frames (FIS) to communicate between host and device

<table>
<thead>
<tr>
<th>Type field value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>27h</td>
<td>Register FIS – Host to Device</td>
</tr>
<tr>
<td>34h</td>
<td>Register FIS – Device to Host</td>
</tr>
<tr>
<td>39h</td>
<td>DMA Activate FIS – Device to Host</td>
</tr>
<tr>
<td>41h</td>
<td>DMA Setup FIS – Bi-directional</td>
</tr>
<tr>
<td>46h</td>
<td>Data FIS – Bi-directional</td>
</tr>
<tr>
<td>58h</td>
<td>BIST Activate FIS – Bi-directional</td>
</tr>
<tr>
<td>5Fh</td>
<td>PIO Setup FIS – Device to Host</td>
</tr>
<tr>
<td>A1h</td>
<td>Set Device Bits FIS – Device to Host</td>
</tr>
<tr>
<td>A6h</td>
<td>Reserved for future Serial ATA definition</td>
</tr>
<tr>
<td>B8h</td>
<td>Reserved for future Serial ATA definition</td>
</tr>
<tr>
<td>BFh</td>
<td>Reserved for future Serial ATA definition</td>
</tr>
<tr>
<td>C7h</td>
<td>Vendor specific</td>
</tr>
<tr>
<td>D4h</td>
<td>Vendor specific</td>
</tr>
<tr>
<td>D9h</td>
<td>Reserved for future Serial ATA definition</td>
</tr>
</tbody>
</table>

FIS – Frame Information Structure

Figure 194 – Register - Host to Device FIS layout
SATA Reconstruction
A Brief Primer on SATA

HOST

Register - Host to Device (HTD)

Contains logical block address (LBA/sector), number of sectors, operation, etc.

Data A

Data B

Data C

Example – DMA Write

DEVICE

Direct Memory Access (DMA) - Activate

Register – Device to Host (DtH)
• Native Command Queuing (NCQ) complicates reconstruction

• NCQ allows for up to 32 separate, concurrent, asynchronous disk transactions
  – Many SATA devices implement NCQ

• NCQ identifies transactions by 5-bit TAG field (0-31)
SATA Reconstruction

• Wrote a Python module to handle all of these transactions
  – Consumes raw SATA frames
  – Supports all of the existing SATA versions
  – Outputs stream of logical sector operations

• Traditional SATA analyzers are expensive and don’t provide analysis-friendly interfaces
**File System Reconstruction**

- **Current Solution**
  - Uses PyTSK to keep a unified codebase in Python
  - Naïve approach requires analyzing the *entire* image at every interval

- **Optimization**: Uses AnalyzeMFT for NTFS optimization

Extract file system state using TSK from initial *clean* image

Check previous state

- if known sector: Update structures
- else: report as UNKNOWN

0 \[\rightarrow\] t \[\rightarrow\] t+1
Automated Binary Analysis
Physical Machines

• Machine/hard disk reset

1. Power down machine

2. Re-image disk with selected OS (CloneZilla)
Automated Binary Analysis
Physical Machines

- Download binary onto SUT

3. Wait for OS to appear on the network (ping)

4. Download binary from controller using ftp (key presses)
Automated Binary Analysis
Physical Machines

- Execute binary

  5. Dump clean state of memory
  6. Start capturing network and disk activity
  7. Run Binary (Start moving mouse)
  8. Dump interim state of memory
  7. Identify and click all buttons (Volatility)
  8. Dump dirty state of memory
Evaluation: Semantic Output
(on WinXPSP3)

• Homemade Rootkit
  – **Comparison**: Anubis failed to execute the binary, and Cuckoo sandbox failed to detect/execute our ftp server

• Labeled Malware (*213 well-labeled samples*)
  – Blind analysis identified various behaviors, all of which were confirmed by ground truth

• Unlabeled Malware (*1091 samples*)
  – Similar findings

<table>
<thead>
<tr>
<th>Observed Behavior</th>
<th>Number of Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Created new process(es)</td>
<td>765</td>
</tr>
<tr>
<td>Opened socket(s)</td>
<td>210</td>
</tr>
<tr>
<td>Started service(s)</td>
<td>300</td>
</tr>
<tr>
<td>Loaded kernel modules</td>
<td>20</td>
</tr>
<tr>
<td>Modified GDT</td>
<td>58</td>
</tr>
<tr>
<td>Modified IDT</td>
<td>10</td>
</tr>
</tbody>
</table>
Evaluation: Evasive Malware
(on Windows 7)

• **Paranoid Fish** (*Evasive malware proof-of-concept*)
  - Failed to detect LO-PHI
  - **Comparison**: Anubis and Cuckoo sandbox were both detected due to virtualization artifacts

• **Labeled Malware** (429 coarsely-labeled samples)
  - LO-PHI detected *suspicious activity* in almost every sample
    - Some appeared to be targeting a different OS version

<table>
<thead>
<tr>
<th>Technique Employed</th>
<th># Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait for keyboard</td>
<td>3</td>
</tr>
<tr>
<td>Bios-based</td>
<td>6</td>
</tr>
<tr>
<td>Hardware id-based</td>
<td>28</td>
</tr>
<tr>
<td>Processor feature-based</td>
<td>62</td>
</tr>
<tr>
<td>Exception-based</td>
<td>79</td>
</tr>
<tr>
<td>Timing-based</td>
<td>251</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Malware Label</th>
<th>Volatility Module</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>envars</td>
</tr>
<tr>
<td>Keyboard</td>
<td>0</td>
</tr>
<tr>
<td>Bios</td>
<td>3</td>
</tr>
<tr>
<td>Hardware</td>
<td>28</td>
</tr>
<tr>
<td>Processor</td>
<td>53</td>
</tr>
<tr>
<td>Exception</td>
<td>76</td>
</tr>
<tr>
<td>Timing</td>
<td>229</td>
</tr>
</tbody>
</table>
Summary

• Deployed and tested LO-PHI an extremely low-artifact, hardware and VM-based, dynamic-analysis environment

• Developed hardware, and supporting tools, for stream-based disk forensics on SATA-based physical machines¹

• Constructed a framework, and accompanying infrastructure, for automating analysis of binaries on both physical and virtual machines
  – Open Source (BSD License): http://github.com/mit-ll/LO-PHI

• Demonstrated the scalability and fidelity of LO-PHI by analyzing thousands of labeled and unlabeled malware samples

Demonstration of VM-based binary analysis.