LO-PHI: Low-Observable Physical Host Instrumentation for Malware Analysis

Presentation of paper

Pietro De Nicolao

Politecnico di Milano

June 13, 2016

Challenges in dynamic malware analysis

LO-PHI

Artifacts

Limitations

Experiment: evasive malware

Criticism

Related and future work

Challenges in dynamic malware analysis

Environment-aware Malware

Observer effect:

- Execution of software into a debugger or VM leaves *artifacts*
- Artifacts are evidence of an "artificial" environment
- They can be reduced or be subtle, but still detectable
- Malware can detect artifacts and hide its true behavior

Malware can resist to traditional dynamic analysis tools.

If the malware "feels" that it's being analyzed, it could:

1. Remain inactive (not trigger the payload)
2. Abort or crash its host
3. Disable defenses or tools

Artifacts: examples

<! – TODO: cite [20] –>

- Hardware
  - Special devices or adapters in VMs
  - Specific manufacturer prefixes on device names
  - Drivers and adapters for host-guest communication
  - Bugs in CPU implementation by VMs
- Memory
  - Hypervisors placing interrupt table in different position
  - Too little RAM (typical of VMs)
- Software
  - Presence of installed tools on the system
  - Suspicious registry keys
  - `isDebuggerPresent()` Windows API
- Behaviour
  - Timing differences

Semantic Gap

Our aim is to **understand what the malware is doing**. Not simple, we need to mine **semantics** from the raw data that we extract.

- From raw data:
  - SATA frame *XYZ*
  - TCP packet *ABC*
- To concise, high-level event descriptions:
  - A file has been written
  - A connection has been opened

Tradeoff between:

1. *Low-artifact, semantically poor* tools (Virtual Machine Introspection)
2. *High-artifact, semantically-rich* frameworks (debuggers)

LO-PHI

The idea

LO-PHI: **Low-Observable Physical Host Instrumentation for Malware Analysis**

- Malware can "feel" the presence of VMs and debuggers.
  - So we remove them: **inspect actual machine with real hardware**
  - Physical sensors and actuators
- Bridging the semantic gap
  - Physical sensors collect raw data.
  - Modified open source tool for disk (Sleuthkit) and memory (Volatility) analysis.
- Extensible to new OSs and filesystem as long as hardware tapping is feasible.
- Also works with Virtual Machines.

Threat model

Assumptions on our model of malware: they are **limitations** of the approach.

- Malware can interact with the system in any way
- Malicious modifications evident either in memory or on disk
- No infection delivered to hardware
- Malware not actively trying to thwart semantic-gap reconstruction
- Instrumentation is in place before malware is executed
  - Malware cannot analyze the system without LO-PHI in place
  - Harder to compare and detect artifacts

Sensors

Sensor: any data collection component.

- **Memory**. Xilinx ML507 board connected to PCIe, reads and writes arbitrary memory locations via DMA.
- **Disk**. ML507 board intercepting all the traffic over SATA interface. Sends SATA frames via Gigabit Ethernet and UDP.
  - Completely passive. . .
  - except when SATA data rate exceeds Ethernet bandwidth: throttling of frames.
- **Network interface**. Mentioned in paper, but the technology used is unclear.

Actuators

Actuator: any component which provides inputs for the system.
Arduino Leonardo used to emulate:

- USB keyboard
- USB mouse

Infrastructure

Restoring physical machines

- We cannot simply “restore a snapshot” like in VMs.
- **Preboot Execute Environment** (PXE) with **CloneZilla**
  - Allows to restore the disk to a previously saved state
  - No interaction with the OS
- Also, DNS and DHCP servers.

Scalable infrastructure

- Job submission system: jobs are sent to a scheduler
- The scheduler executes the routine on an appropriate machine

Common interface (1)

Python script for running a malware sample and collecting the appropriate raw data for analysis.

```python
disk_tap.start()
# Send key presses to download binary
machine.keypress_send(ftp_script)
# Dump memory (clean)
machine.memory_dump(memory_file_clean)
network_tap.start()
# Get a list of current visible buttons
button_clicker.update_buttons()
# Start our binary and click any buttons
machine.keypress_send('SPECIAL:RETURN')
# Move our mouse to imitate a human
machine.mouse_wiggle(True)
time.sleep(MALWARE_START_TIME)
# ...
# Click any new buttons that appeared
button_clicker.click_buttons(new_only=True)
time.sleep(MALWARE_EXECUTION_TIME-elapsed_time)
machine.screenshot(screenshot_two)
machine.memory_dump(memory_file_dirty)
machine.power_shutdown()
```

Common interface (2)

The framework supports:

- Real, physical machines
- Traditional Virtual-Machine Introspection

The abstracted software interface written in Python is the same.
We can focus on high-level functionality.

Artifacts

Memory throughput

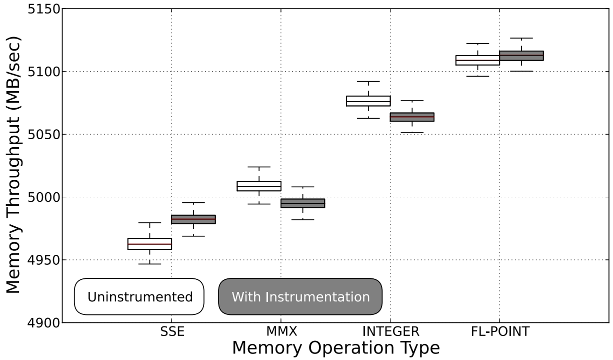
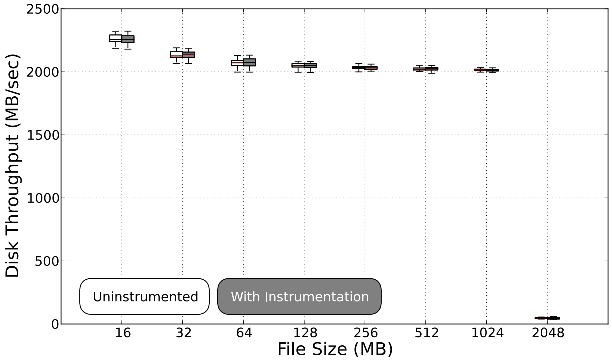


Figure 1: Average memory throughput comparison as reported by RAMSpeed, with and without instrumentation. Deviation from uninstrumented trial is only 0.4% in worst case.

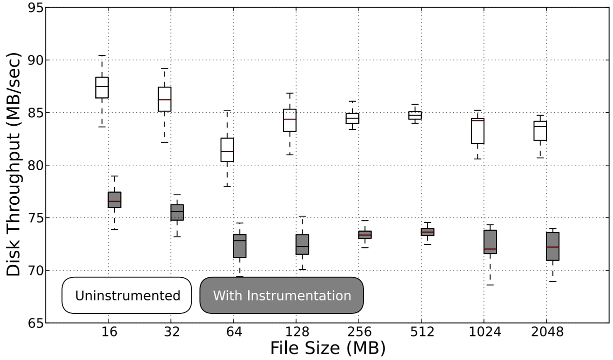
Disk throughput: reads



(b) File reads

Figure 2: File system read throughput comparison as reported by IOZone on Windows XP, with and without instrumentation on a physical machine.

Disk throughput: writes



(a) File writes

Figure 3: There are significant differences for write throughput since here the cache does not help.

Limitations

Experiment: evasive malware

Criticism

Related and future work

Reftest

Hello (Pasticcio 2011).

References

Pasticcio, Ciccio. 2011. “Security Engineering.” *SecProceedings*.