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

Chad Spensky, Hongyi Hu and Kevin Leach

Pietro De Nicolao

Politecnico di Milano

June 13, 2016

The paper and the code

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

Chad Spensky*†, Hongyi Hu*§ and Kevin Leach*‡
*MIT Lincoln Laboratory *lophi@mit.edu*
†University of California, Santa Barbara *cspensky@cs.ucsb.edu*
§Dropbox *hongyihu@alum.mit.edu*
‡University of Virginia *kjl2y@virginia.edu*

Network and Distributed System Symposium (NDSS) '16
21-24 February 2016, San Diego, CA, USA

- GitHub repository: `https://github.com/mit-ll/LO-PHI`

Open challenges in Dynamic Malware Analysis

Dynamic Malware Analysis

Idea: **run the malware** in a sandbox (VM, debugger, . . .) and use tools to analyze its behavior.

- We are interested in observing:
  - Memory accesses
  - Network activity
  - Disk activity
- Observation tools must be placed **outside** the sandbox
- At a **lower level** w.r.t. the malware
- In theory, completely *transparent* to the malware
- *Not so simple. . .*

Virtualization is like dreaming

- Unsure if you're living in a dream, or awake?
  - Look for **artifacts** (i.e. anomalies) in your reality!

Malware can do the same...

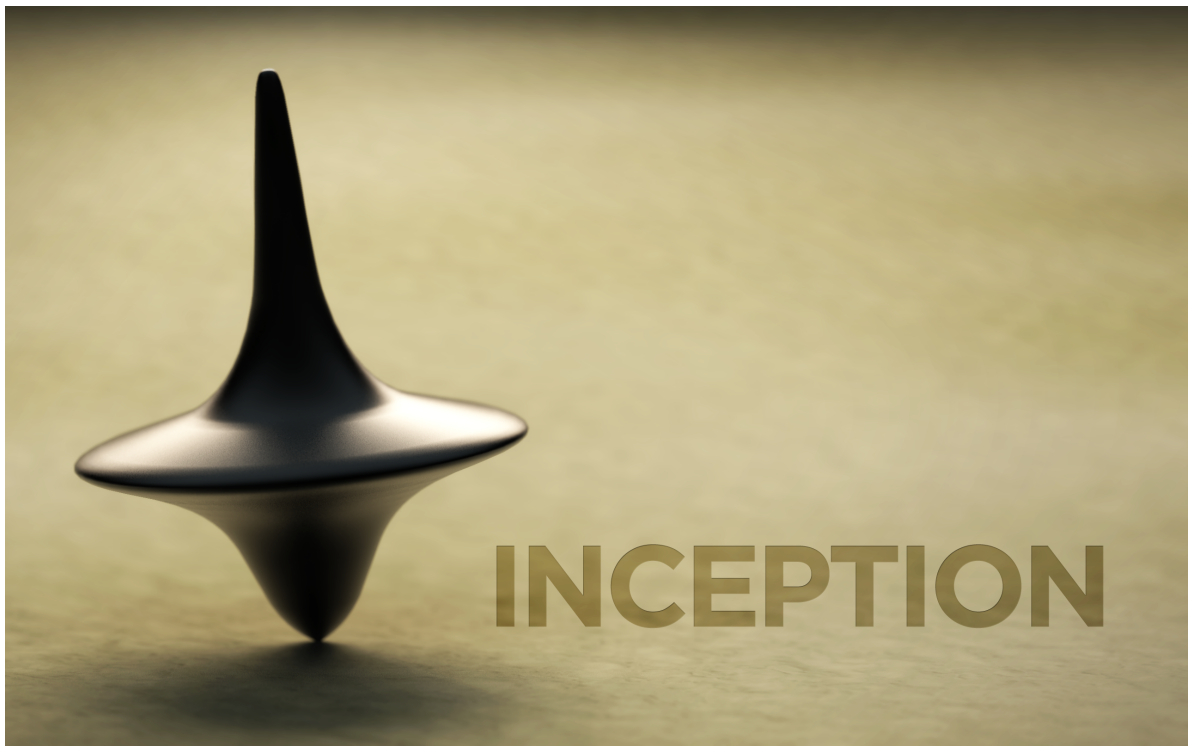

Figure 1: Never stopping spinning top: a possible artifact in dreams.

Artifacts and environment-aware malware

Observer effect

Execution of software into a debugger or VM leaves **artifacts**.

- Artifacts are evidences of an "artificial" environment
- According to [Garfinkel, 2007], *building a transparent VMM is fundamentally infeasible*.

Malware resistance to dynamic analysis

If the malware is able to detect artifacts, it **can resist to traditional dynamic analysis tools.**

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

Artifacts: some examples

[Chen, 2008], [Garfinkel, 2007] provide a taxonomy of artifacts:

- **Hardware**
  - Special devices or adapters in VMs
  - Specific manufacturer prefixes on device names
- **Memory**
  - Hypervisors placing interrupt table in different positions
  - Too little RAM (typical of VMs)
- **Software**
  - Presence of installed tools on the system
  - `isDebuggerPresent()` Windows API
- **Behavior**
  - Timing differences (e.g. interception of privileged instructions in VMMs)

Semantic Gap

What is the malware doing?

Need to mine **semantics** from the extracted raw data.

- From raw data:
  - Disk read at sector *123*
  - TCP packet *ABC*
  - Some bytes at memory address `0xDEADBEEF`
- To concise, high-level event descriptions:
  - `/etc/passwd` has been read
  - A connection to `badguy.com` has been opened

Forensics to the rescue

LO-PHI uses well-known **forensic tools**, adapted for live analysis:

- Disk forensics: **Sleuthkit**
- Memory forensics: **Volatility**

Malware Analysis Framework Evaluation

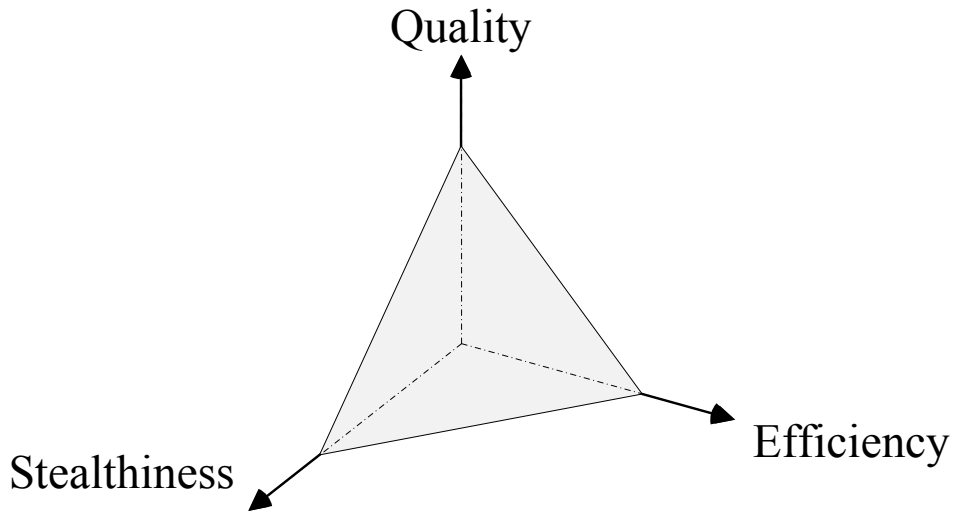

Figure 2: From [Kirat, 2011]

Tradeoff between:

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

LO-PHI framework implementation

Goals

- **No virtualization**: run malware on bare metal machines!
  - Physical sensors and actuators
- Bridging the **semantic gap**
  - Physical sensors collect raw data
- **Automated restore** to pre-infection state
- **Stealthiness**: very few, undetectable artifacts
- **Extendability**: support new OSs and filesystems

Threat model

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

1. Malicious modifications evident either in memory or on disk
2. No infection delivered to hardware
3. Instrumentation is in place before malware is executed
    - Malware cannot analyze the system without LO-PHI in place
    - Harder to compare and detect artifacts: **no baseline**

Sensors and actuators (1)

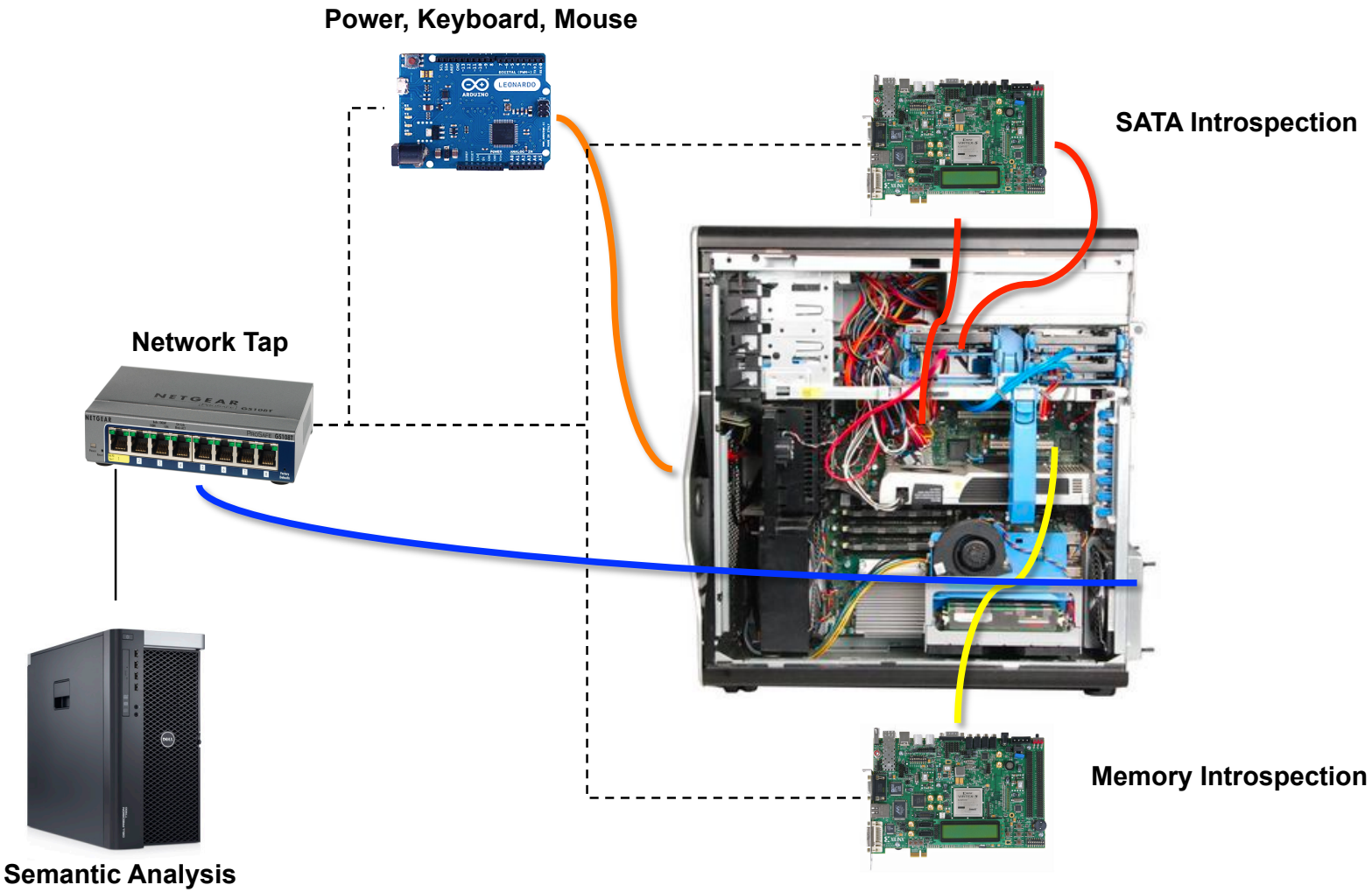

Figure 3: Hardware instrumentation to inspect a bare-metal machine.

Sensors and actuators (2)

Sensors

- **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

An Arduino Leonardo emulates **USB keyboard and 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

Scalable infrastructure

- Job submission system: jobs are sent to a **scheduler**
- The scheduler executes the routine on an appropriate machine
- **Python interface** to control the machine and run malware and analysis

Bridging the semantic gap via forensic analysis

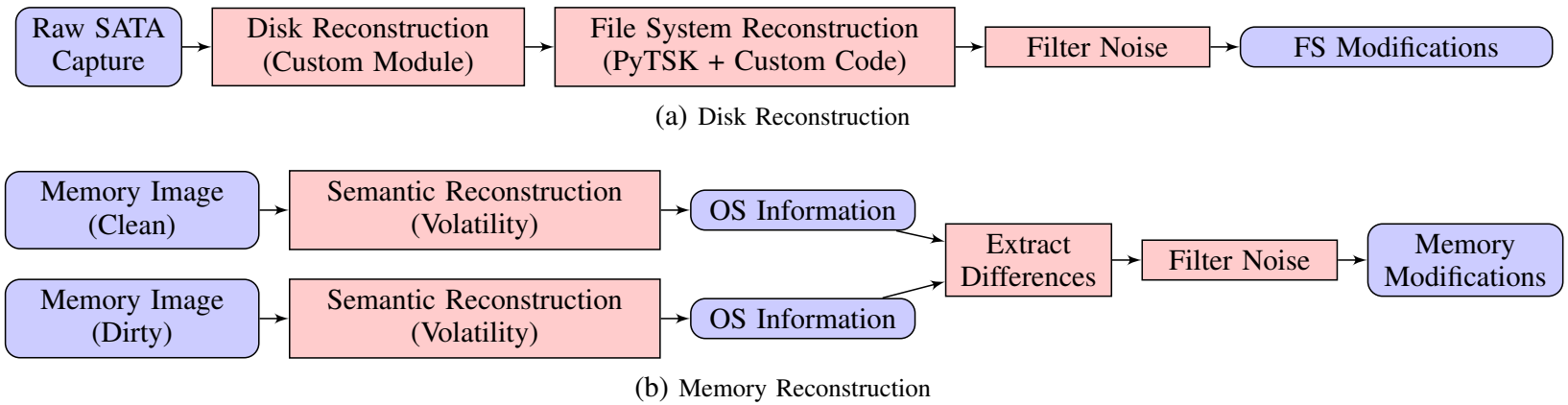

Figure 4: Binary analysis workflow. Rounded nodes represent data and rectangles represent data manipulation.

- **Background noise** was removed by analyzing non-malicious software.

Example of output of the analysis toolbox

Offset	Name	PID	PPID
0x86292438	AcroRd32.exe	1340	1048
0x86458818	AcroRd32.exe	1048	1008
0x86282be0	AdobeARM.exe	1480	1048
0x864562a0	*$$_rk_sketchy_server.exe*	1044	1008

Figure 5: New processes.

PID	Port	Protocol	Address
1048	1038	UDP	127.0.0.1
1044	*21*	*TCP*	*0.0.0.0*

Figure 6: New sockets.

Created Filename
/.../lophi/$$_rk_sketchy_server.exe
/.../lophi/hookssdt.sys
/.../lophi/sample_0742475e94904c41de1397af5c53dff8e.exe

Figure 7: Disk event log.

Experiments

Experiment on evasive malware

Malware previously labeled as "evasive" was executed on Windows 7 and analyzed using LO-PHI.

- Many samples clearly exhibited **typical malware behavior**.
- Some samples failed because of no network connection, or wrong Windows version.

[. . .] we feel that our findings are more that sufficient to showcase LO-PHI's ability to analyze evasive malware, without being subverted, and subsequently produce high-fidelity results for further analysis.

Timing

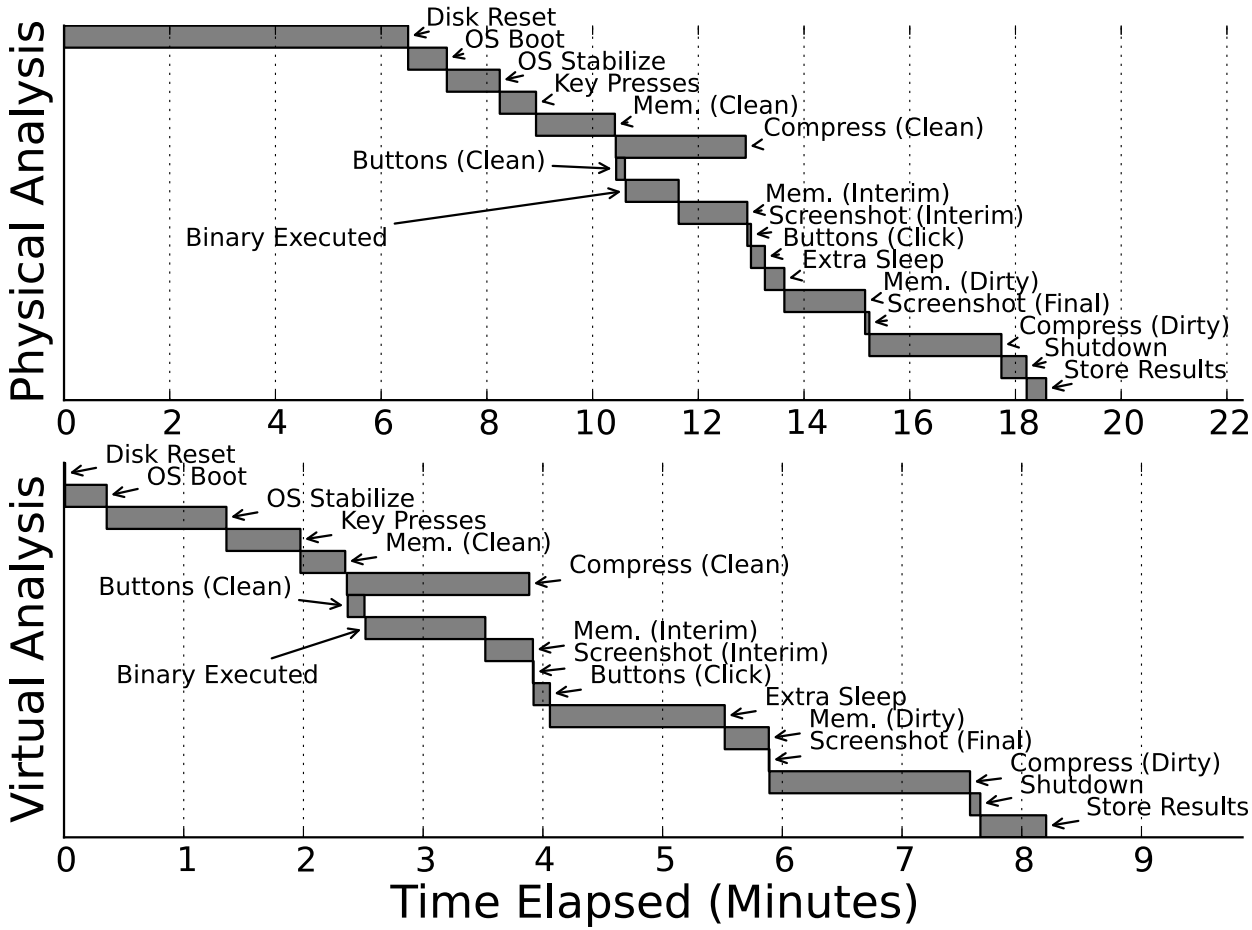

Figure 8: Time spent in each step of binary analysis. Both environments were booting a 10 GB Windows 7 (64-bit) hibernate image and were running on a system with 1 GB of volatile memory.

Critique

Known limitations

- Newer chipsets use IOMMUs, **disabling DMA** from peripherals
  - Current memory acquisition technique will become unusable
- **Smearing**: the memory can change *during* the acquisition
  - Inconsistent states
  - Faster polling rates can help
- **Filesystem caching**: some data will not pass through SATA interface
  - Malware could write a file to disk cache, execute and delete it before the cached is flushed to disk.
  - However, the effects would be visible in memory.

Issues with the technique and the experiments

- The malware is left to run only 3 minutes.
  - Many malwares need much more time to fully uncover their effects (e.g. ransomware).
- **No memory polling** during the execution of the malware
  - Only snapshot before and after the execution
  - Temporary data used by the malware is never seen
- Assumption: **malware does not modify BIOS or firmware**.
  - But if it does, the physical machine could not be recoverable.
  - Costly!
- **No Internet access**: the authors always run the malware on disconnected machines.
  - Most malware becomes useless without Command&Control infrastructure.
  - Network access could expose further, unseen, artifacts.

Methodological issues

- The article claims that the artifacts from LO-PHI are unusable by malware because there's **no baseline** (i.e. the malware cannot see the machine before the installation of LO-PHI).
  - This can also be true for traditional approaches.
- **No statistical test** used to discover whether difference in disk/memory throughput is statistically significant (presence of artifacts).
  - Very simple and standard procedure, should really be done in a scientific paper.
- **Network analysis** technique is not described and unclear.
  - *We exclude the network trace analysis from much of our discussion since it is a well-known technique and not the focus of our work.*

Memory throughput with and without LO-PHI

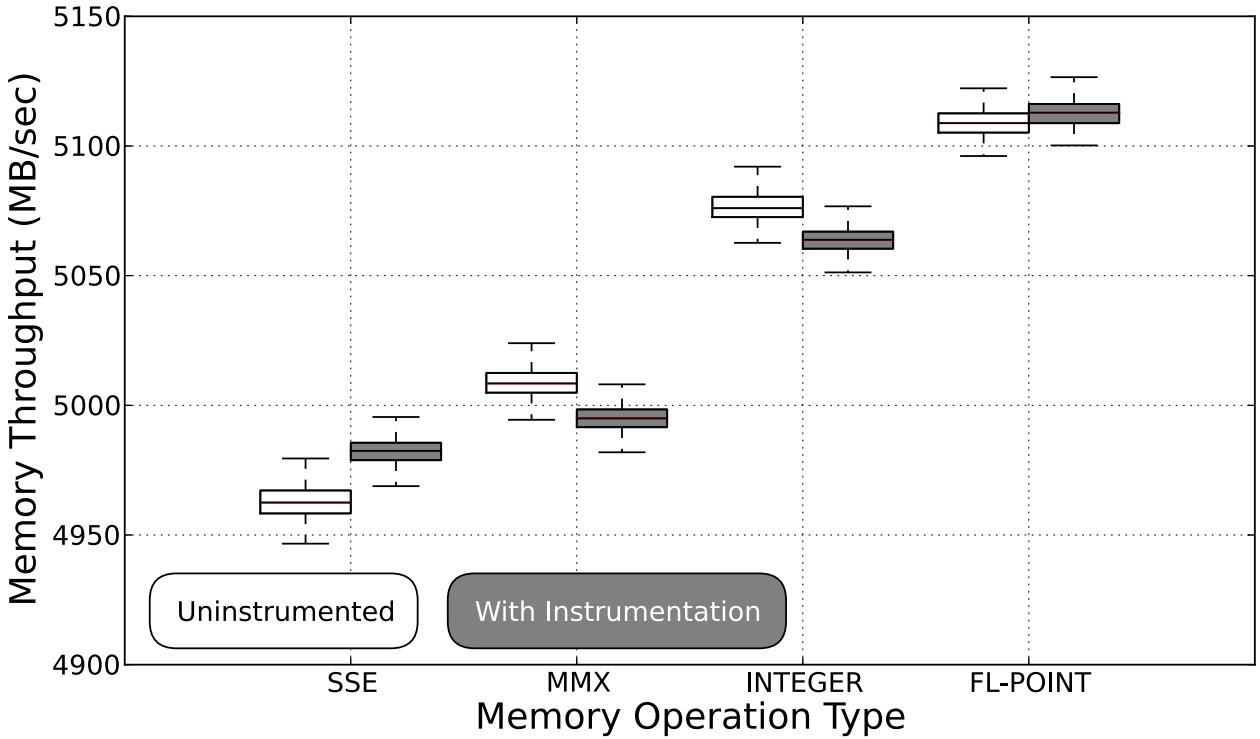

Figure 9: Average memory throughput comparison as reported by RAMSpeed, with and without instrumentation. Deviation from uninstrumented trial is only 0.4% in worst case. **No statistical test used.**

Another point...

- **Virtualization is increasingly used** in production contexts.
  - Just think about **cloud computing**!
- Attackers will want to target those VMs [Garfinkel, 2007]
- Malware that deactivates in VMs will be less common

Related and future work

Related work

Traditional dynamic analysis

Many dynamic malware analysis tools rely on virtualization: Ether, BitBlaze, Anubis, V2E, HyperDbg, SPIDER.

- We already saw the limitations of VM approaches: **artifacts**

Bare-metal dynamic analysis

BareBox [Kirat, 2011]: malware analysis framework based on a bare-metal machine without virtualization or emulations

- Only targets user-mode malware
- Only disk analysis (no memory tools)

Future evolutions

Automated, repeated analyses

- **Disk restore** phase is the lengthiest ($>$ 6 min)
- *The resetting and boot process could be decreased significantly by writing a custom PXE loader, or completely mitigated by implementing copy-on-write into our FPGA.*
- *While snapshots are trivial with virtual machines, it is still an open problem for physical machines.*

Extensions

- Analyze **transient behavior** of binary
  - Continuous memory polling
  - Need to deal with DMA artifacts
- Cover **malware that infects hardware** (BIOS, firmware)

References

Chen X., Andersen J, Morley M., Bailey M., Nazario J.
Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware
In Proceedings of the International Conference on Dependable Systems and Networks (2008)
doi: 10.1109/DSN.2008.4630086

Kirat D., Vigna G., Kruegel C.
BareBox: Efficient Malware Analysis on Bare-metal
Proceedings of the 27th Annual Computer Security Applications Conference (2001)
doi: 10.1145/2076732.2076790

Garfinkel T., Adams K., Warfield A., Franklin J.
Compatibility is Not Transparency: VMM Detection Myths and Realities
Proceedings of the 11th USENIX Workshop on Hot Topics in Operating Systems (2007)