IntroLib: Efficient and Transparent Library Call Introspection for Malware Forensics

By

Zhui Deng, Dongyan Xu, Xiangyu Zhang and Xuxian Jiang

Presented At

The Digital Forensic Research Conference

DFRWS 2012 USA Washington, DC (Aug 6th - 8th)

IntroLib: Efficient and Transparent Library Call Introspection for Malware Forensics

Zhui Deng[†], Dongyan Xu[†], Xiangyu Zhang[†], Xuxian Jiang[*]

[†]Purdue University
[*]North Carolina State University

PURDUE UNIVERSITY

Motivation

- Malware Analysis
  - Reveal goals and detailed behavior of malware
- Dynamic Analysis
  - Complement static analysis to capture and analyze runtime behavior
- Tamper Resistance
  - Higher privilege over malware in virtualization environment
- Anti-analysis Methods
  - Thwart dynamic analysis

Existing Approach:

- Ether (Dinaburg et al., 2008), MAVMM (Nguyen et al., 2009)
  - Leverage hardware virtualization
    - Avoid problem of instruction semantic discrepancies in emulators
  - Suffer from significant performance overhead when performing fine-grained live malware analysis
    - Single-stepping triggers transitions between guest and hypervisor

Existing Approach:

- Kang et al. (2009)
  - Guide the execution of malware in emulators using transparent reference systems
  - High performance penalty when obtaining execution traces
- V2E (Yan et al., 2012)
  - Selectively emulating instructions to boost performance
  - enumerate all such instructions remains a challenge

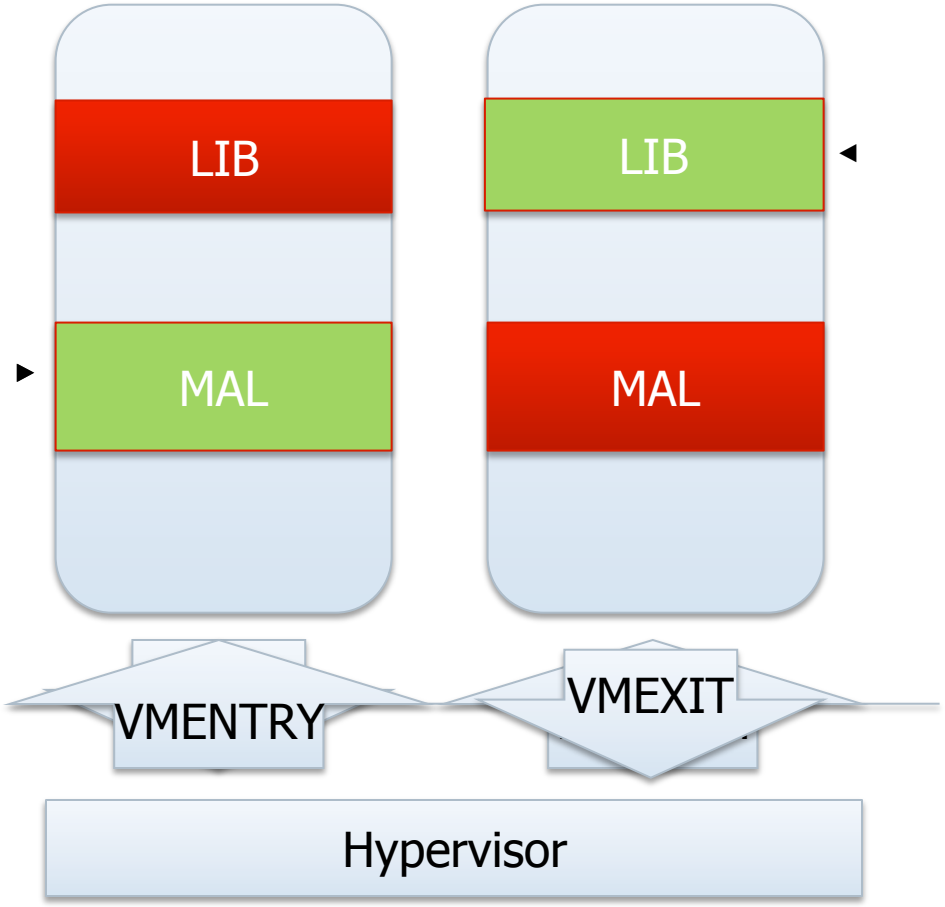
Our Approach

- IntroLib
  - Track and log the sequence of user-level library calls made by the malware
  - More informative and provides more insights into malware
    - Compare with system call based introspection
  - More lightweight and suitable for live malware forensics
    - Compare with instruction-level dynamic analysis
  - More immune to malware's emulation detection logic
    - Compare with emulation-based tools
  - Cover all kinds of user-mode library calls
    - such as Windows API library functions and C library functions.

Goals

- Trustworthiness
  - Cover all user-mode library calls
- High transparency to malware
  - Difficult to detect by advanced anti-analysis malware
- Efficiency
  - The performance overhead should be low for live forensic analysis

Design: Intercept Control flow Transitions



Lib!Fn:
  push ebp
  …
  ret

Mal:
  call Lib!Fn
  cmp eax, 1

Design: Identify Memory Layout

- Categorize user-mode pages into three types:
  - malware code, library code and data pages
- Malware program might generate new code and load additional libraries at runtime
  - Data pages => malware code | library code pages
- Lazy Identification
  - Code pages are not identified until there is an instruction fetched
  - Hypervisor leverages VMI to determine the type of memory area

Design: Logging

- Read the source/destination addresses of the transition from the Last Branch Record (LBR) stack
- Parse library function names/addresses from library files
  - Copy-on-write disk to prevent tampering
- Read function arguments from stack according to function prototype
- Match function return by recording return address
- Extract return value from EAX register

Design: Improving Transparency

- Avoid timing attack
  - Modify execution time queried from guest
- Shadow LBR Stack
  - Conceal utilization of LBR from guest

Evaluation

- Evaluate in three aspects
  - Functionality
  - Transparency
  - Performance
- Hardware configuration
  - Host CPU: Intel(R) Core(TM) i5-2410M 2.30GHz
  - Host memory: 4GB
  - Host OS: Ubuntu Linux 11.04 64bit with kernel version 2.6.38
  - Guest memory: 1GB
  - Guest disk: 10GB
  - Guest OS: windows xp with no service pack

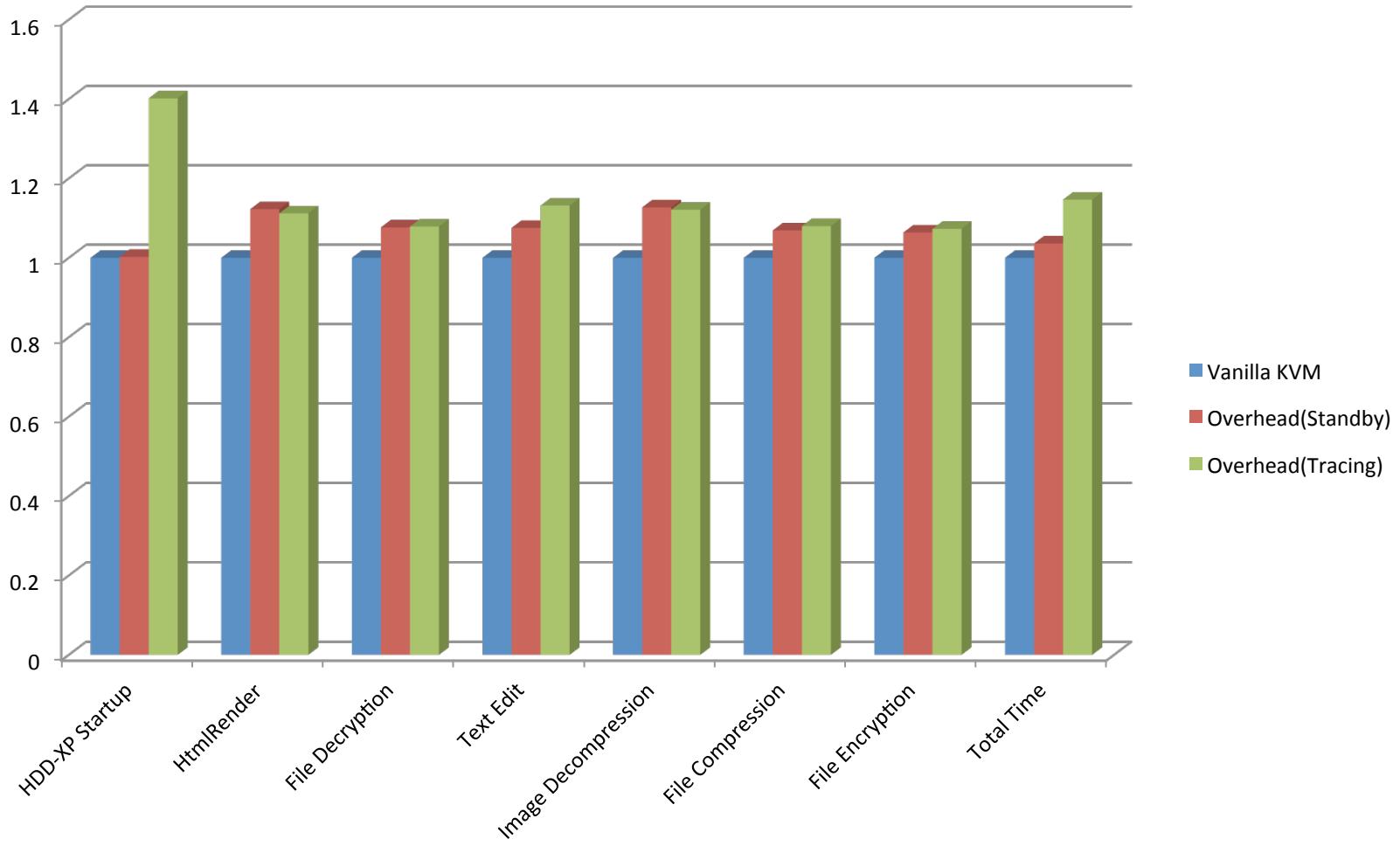
Evaluation: Functionality

- Evaluate with a pool of 93 real-world, Windows-based malware samples
- Case study 1
  - Win32/FakeRean
  - Disguised as a rogue anti-virus tool
- Case study 2
  - Win32/Dorkbot.AJ
  - multiple levels of packing and encryption/decryption to prevent static analysis
  - several anti-debugging and anti-VM tricks to thwart dynamic analysis.
- Fine-grained library call tracing logs is more helpful than system call logs for understanding malware

Evaluation: Transparency

- Crafted synthetic anti-emulation samples using detection code in previous work
  - None of these samples were able to detect our system
- 93 real-world malware samples
  - 3 of them crashed before showing any behavior
    - Crashed on Anubis and CWSandbox
    - Crashed when being executed in an unmodified KVM
    - The crashes were not due to detection of IntroLib
  - Obtain library call logs of all the remaining 90 samples
- IntroLib could maintain transparency to all similar attacks which detect the presence of analysis system in memory

Evaluation: Performance



Discussion

- IntroLib is not completely undetectable
  - TLB flush
  - TLB-based detection incurs high false positive rate
- IntroLib conceals itself but not the underlying virtualization platform
  - Virtualization has become a universal platform
- IntroLib relies on page level protection, so intra-page transitions could not be intercepted
  - Co-location of malware code and library code in the same page is rare and difficult
- IntroLib relies on some in-guest data when performing memory layout identification and library call logging
  - Possible solution: content-based identification of library code area and functions

Conclusion

- Perform efficient library calls tracing for malware forensics
- Utilize hardware virtualization to elevate its transparency
- Page table-based mechanism
- Uncover richer information than system call tracing-based approaches
- Incur low overhead

Thank You

{deng14,dxu,xyzhang}@cs.purdue.edu
jiang@cs.ncsu.edu