Ether
Malware Analysis via Hardware Virtualization Extensions

Artem Dinaburg*, Paul Royal†, Monirul Sharif† and Wenke Lee†

*Georgia Institute of Technology
†Damballa

ACM CCS 2008
Agenda

- Motivation
  - The malware problem
- The Ether Framework
  - Transparency and transparent malware analysis
- Evaluation
  - Comparing Ether to current approaches
- Conclusion
The Malware Problem

- A centerpiece of current security threats
  - Botnets
  - Spam
  - Information Theft
  - Financial Fraud

- Real Criminals
  - Criminal infrastructure
  - Domain of organized crime
Malware Analysis

- There is a profound need to understand malware behavior
  - Forensics and Asset Remediation
  - C&C Detection
  - Threat Analysis

- Malware authors make analysis very challenging
  - Direct financial motivation
Two Types of Malware Analysis

- **Static Analysis**
  - What a program would do
  - Complete view of program behavior
  - Requires accurate disassembly of x86 machine code
  - Often impossible to do in practice

- **Dynamic Analysis**
  - Shows what a program actually did when executed
  - Only gives a partial view of program behavior
  - Misses trigger based actions
  - How do you hide your analyzer?
The Malware Uncertainty Principle

- An important practical problem
- Observer affecting the observed environment
- Robust and detailed analyzers are typically invasive
  - In-memory presence
  - Hooks
  - CPU Emulation
- Malware will refuse to run
The Malware Uncertainty Principle, Commercialized

- Dynamic analyzer detection is a standard malware feature
Explaining the Malware Uncertainty Principle

- Why such a high detection rate?
- Detection of In-Guest presence
  - PolyUnpack, CWSandbox
- Detection of Whole-System emulation
  - Anubis, Renovo
- Detection of API Emulation
  - Norman Sandbox
Contributions

- **Transparency**
  - The theory
- **Ether: A transparent malware analysis platform**
  - The implementation
- **An externally reproducible evaluation of our results**
  - Source Code
  - Malware Samples
Solving the Malware Uncertainty Principle

- An analyzer’s aim should be *transparency*.
  - Defining transparency
- The execution of the malware and the malware analyzer is governed by the principle of *non-interference*. 
Transparency Requirements

- Higher Privilege
- No non-privileged side effects
- Same instruction execution semantics
- Identical exception handling
- Identical notion of time
Additional Analyzer Requirements

- **Semantic information**
  - Process names, system call arguments, etc.

- **Coarse grained (system call level) tracing**
  - Behavioral anti-virus
  - Malware Analysis Services

- **Fine grained (instruction by instruction) tracing**
  - Dynamic taint analysis
  - Automated unpacking
  - Multipath exploration
Fulfilling Transparency Requirements

- Debugging API
  - In-guest presence
  - Exception Handling

- Reduced Privilege Guests (VMWare, etc)
  - Non-privileged side effects

- Emulation (QEMU, Simics)
  - Instruction execution semantics
Fulfilling Transparency Requirements

- Idea: Use hardware assisted virtualization
- Provides several attractive transparency features
  - External
  - Capable
  - Equivalent
- Poses complex analysis challenges
  - Different goals
Challenges

- A transparent yet functional malware analyzer
- Use features of Intel VT in novel ways to achieve:
  - Guest memory analysis
  - Coarse grained tracing
  - Fine grained tracing
- Maintaining transparency
The Ether Framework

Dom0

Ether Userspace Component

DomU (Windows Guest)

DomU (Windows Guest)

Ether Hypervisor Component

CPU / Hardware

Xen
Detecting Ether

- Detecting Intel VT
  - Increasingly irrelevant
  - Not the same

- Timing attacks
  - Network-based clock sources
  - Nothing we can really do

- Memory Hierarchy Attacks
  - Use AMD...
About EtherTrace

- An implementation of a coarse grained tracer using the Ether framework
- Traces the Windows equivalent of system calls (Native API)
  - Concept extends to other OSes
- Information Provided:
  - Call name
  - Typed arguments
  - Return values
  - Context (Process ID, Thread ID)
About EtherUnpack

- Precision universal automated unpacker
- Uses instruction-by-instruction tracing (fine grained tracing) to detect unpack execute behavior
- If code written is later executed, unpack-execution occurred
  - First proposed in Renovo
- Able to handle multiple packing layers
- Dumps unpacked memory images to disk
Evaluation: EtherTrace

- Examine trace logs for expected actions
  - File
  - Registry
Evaluation: EtherTrace

- Obfuscation tools traced ranked by popularity
Evaluation: EtherTrace

 Ether is more transparent
Evaluation: EtherUnpack

- Looked for a 32 byte string present in the original code section
- Not a random string
  - Avoid API calls
  - Not at entry point
  - On code path
Evaluation: EtherUnpack

- **Obfuscation tools unpacked ranked by popularity**

- **Automated Unpacking: Renovo**
  - Armadillo: 29%
  - Themida: 8%
  - UPX: 17%
  - PECompact: 5%
  - Aspack: 4%
  - FSG: 4%
  - WinUPack: 2%
  - Yoda’s Prot: 2%
  - MEW: 1%
  - Molebox: 1%
  - Morphine: 1%

- **Automated Unpacking: PolyUnpack**
  - Armadillo: 29%
  - Themida: 8%
  - UPX: 17%
  - PECompact: 5%
  - Aspack: 5%
  - FSG: 4%
  - WinUPack: 2%
  - Yoda’s Prot: 2%
  - MEW: 1%
  - Molebox: 1%
  - Morphine: 1%
  - UPX 5
Evaluation: EtherUnpack

Automated Unpacking: EtherUnpack

- Ether is more transparent
Conclusion

- An inadequacy of current tools
- Theoretically, we can do better
- Ether is an implementation of a different approach
- Evaluation confirms Ether is more transparent
Questions?

Source code and samples available at:

http://ether.gtisc.gatech.edu