Using Hardware Features for Increased Debugging Transparency

Fengwei Zhang, Kevin Leach, Angelos Stavrou, Haining Wang, and Kun Sun. In S&P'15.

Fengwei Zhang

Overview

- Motivation
- Background: System Management Mode (SMM)
- System Architecture
- Evaluation: Transparency and Performance
- Conclusions and Future Directions

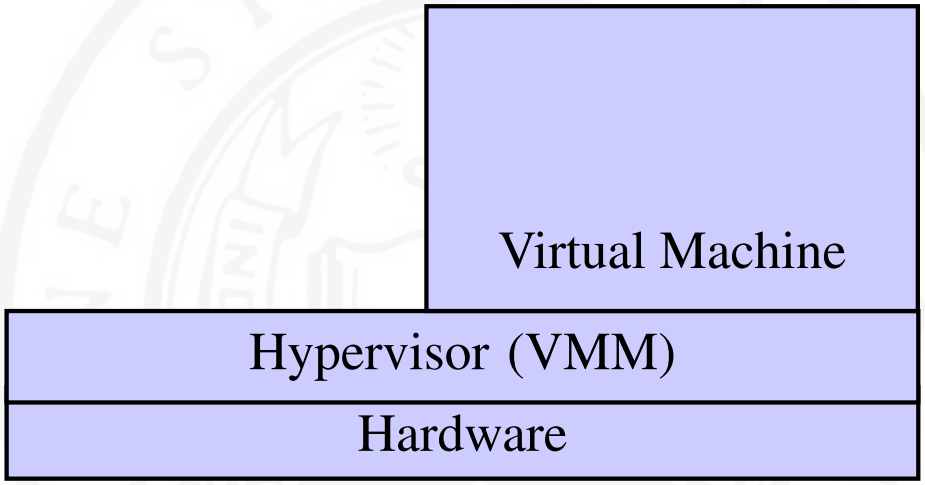
Overview

- Motivation
- Background: System Management Mode (SMM)
- System Architecture
- Evaluation: Transparency and Performance
- Conclusions and Future Directions

Motivation

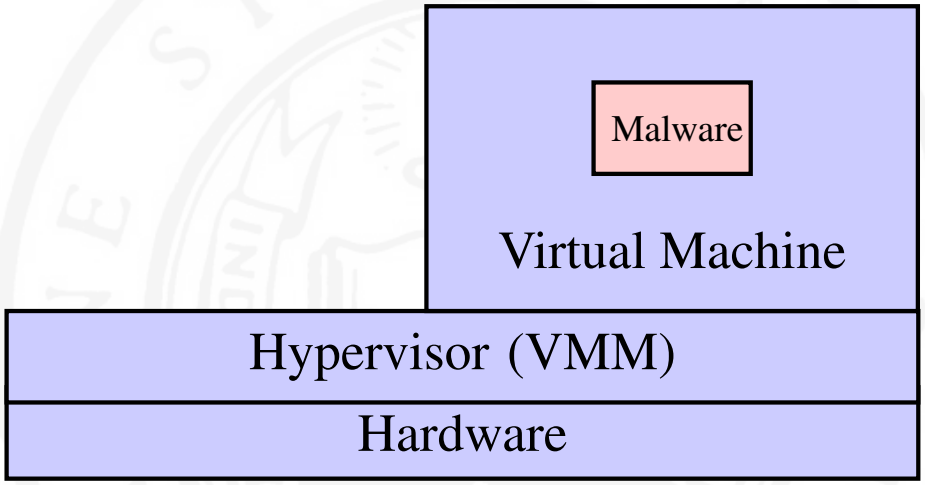
- Malware attacks statistics
  - Symantec blocked an average of 247,000 attacks per day [1]
  - McAfee (Intel Security) reported 8,000,000 new malware samples in the first quarter in 2014 [2]
  - Kaspersky reported malware threats have grown 34% with over 200,000 new threats per day last year [3]
- Computer systems have vulnerable applications that could be exploited by attackers.

Traditional Malware Analysis



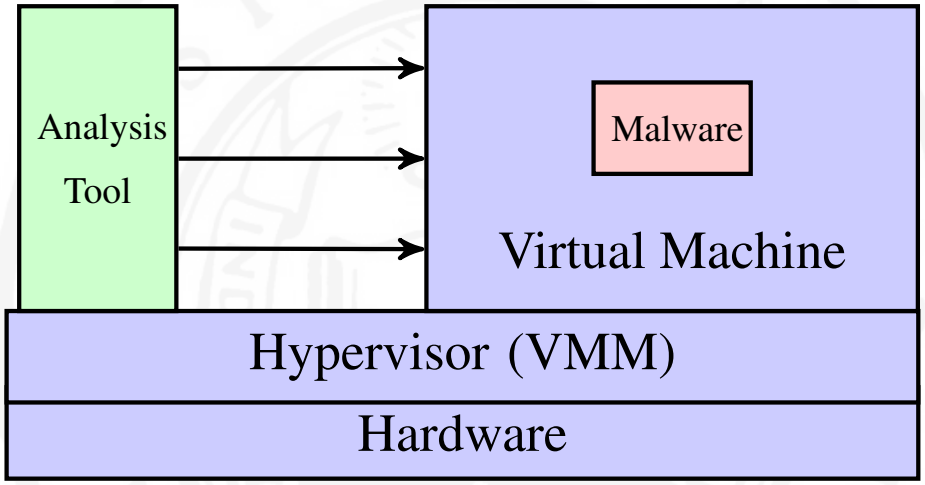
- Using virtualization technology to create an isolated execution environment for malware debugging

Traditional Malware Analysis



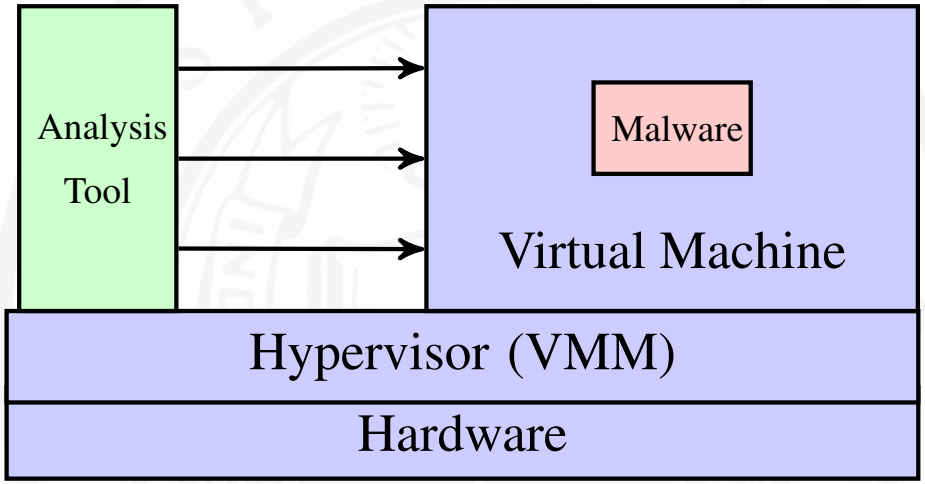
- Using virtualization technology to create an isolated execution environment for malware debugging
- Running malware inside a VM

Traditional Malware Analysis



- Using virtualization technology to create an isolated execution environment for malware debugging
- Running malware inside a VM
- Running analysis tools outside a VM

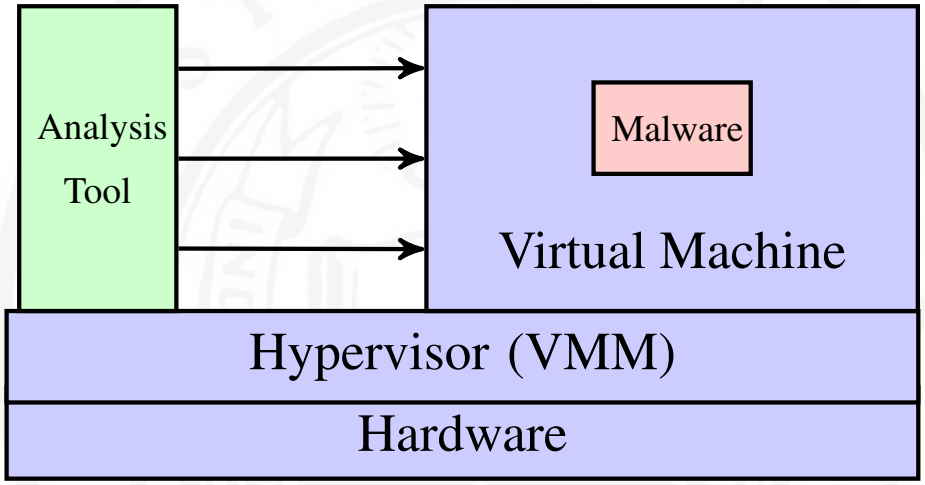
Traditional Malware Analysis



Limitations:

- Depending on hypervisors that have a large TCB (e.g., Xen has 500K SLOC and 245 vulnerabilities in NVD)
- Incapable of analyzing rootkits with the same or higher privilege level (e.g., hypervisor and firmware rootkits)
- Unable to analyze armored malware with anti-virtualization or anti-emulation techniques

Our Approach



We present a bare-metal debugging system called MalT that leverages System Management Mode for malware analysis

- Uses System Management Mode as a hardware isolated execution environment to run analysis tools and can debug hypervisors
- Moves analysis tools from hypervisor-layer to hardware-layer that achieves a high level of transparency

Overview

- Motivation
- Background: System Management Mode (SMM)
- System Architecture
- Evaluation: Transparency and Performance
- Conclusions and Future Directions

Background: System Management Mode

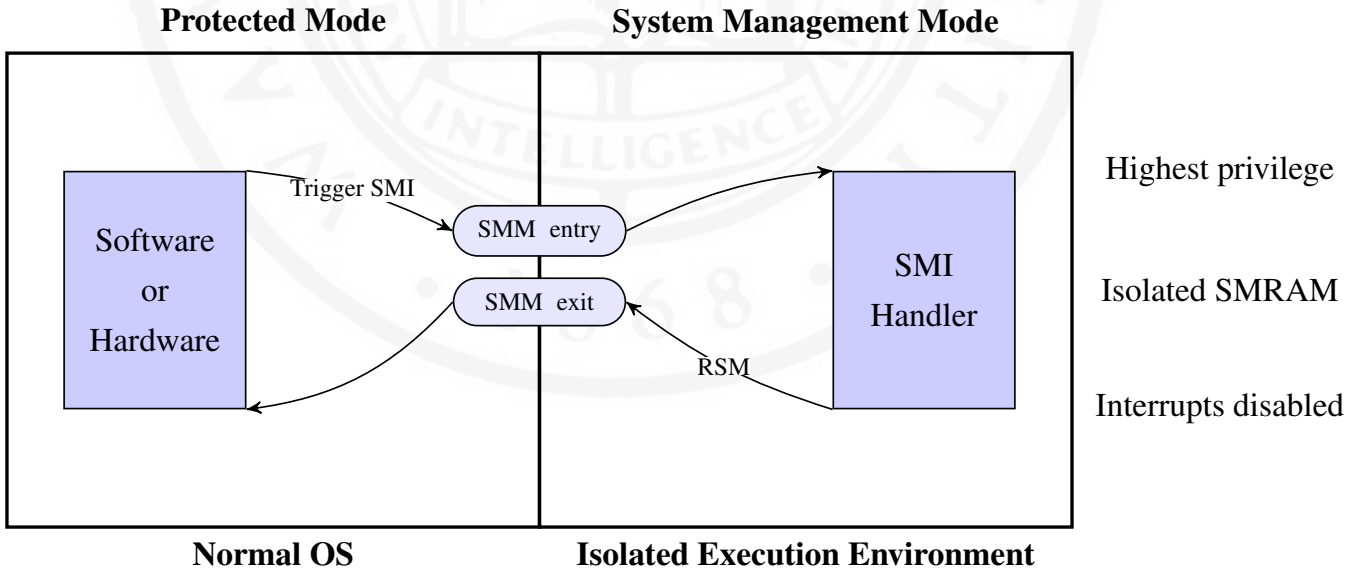
System Management Mode (SMM) is special CPU mode existing in x86 architecture, and it can be used as a hardware isolated execution environment.

- Originally designed for implementing system functions (e.g., power management)
- Isolated System Management RAM (SMRAM) that is inaccessible from OS
- Only way to enter SMM is to trigger a System Management Interrupt (SMI)
- Executing RSM instruction to resume OS (Protected Mode)

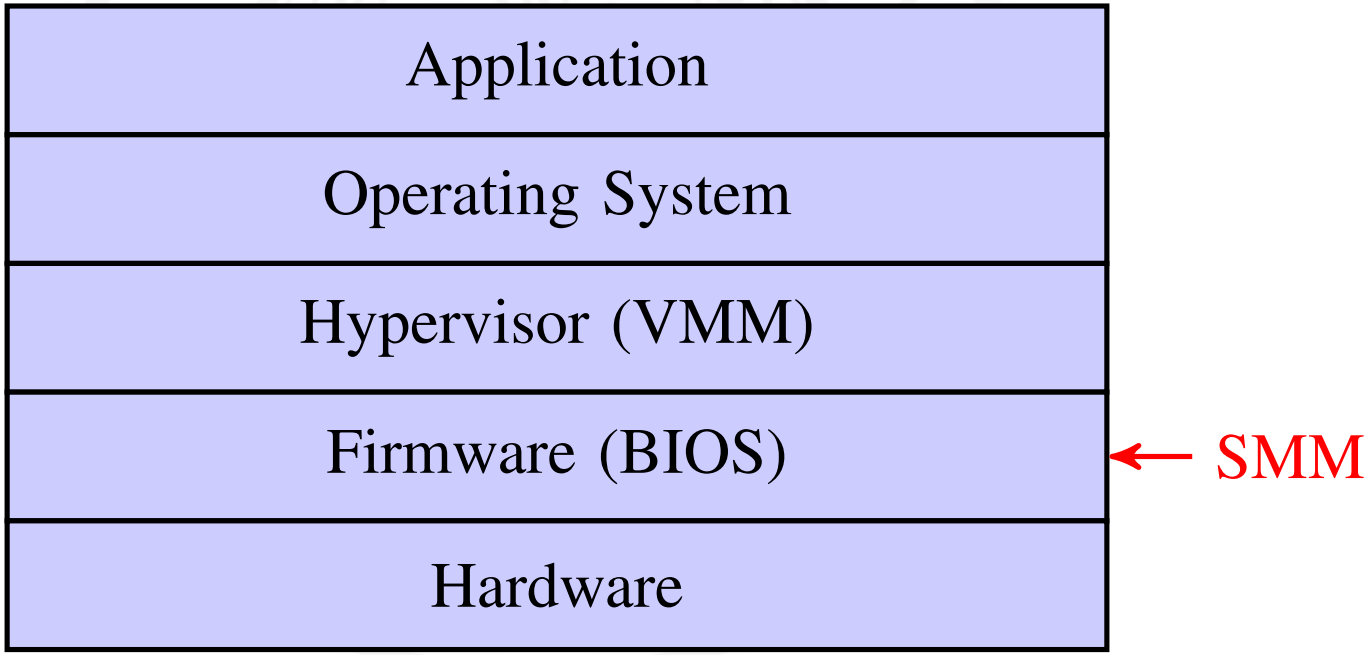
Background: System Management Mode

Approaches for Triggering a System Management Interrupt (SMI)

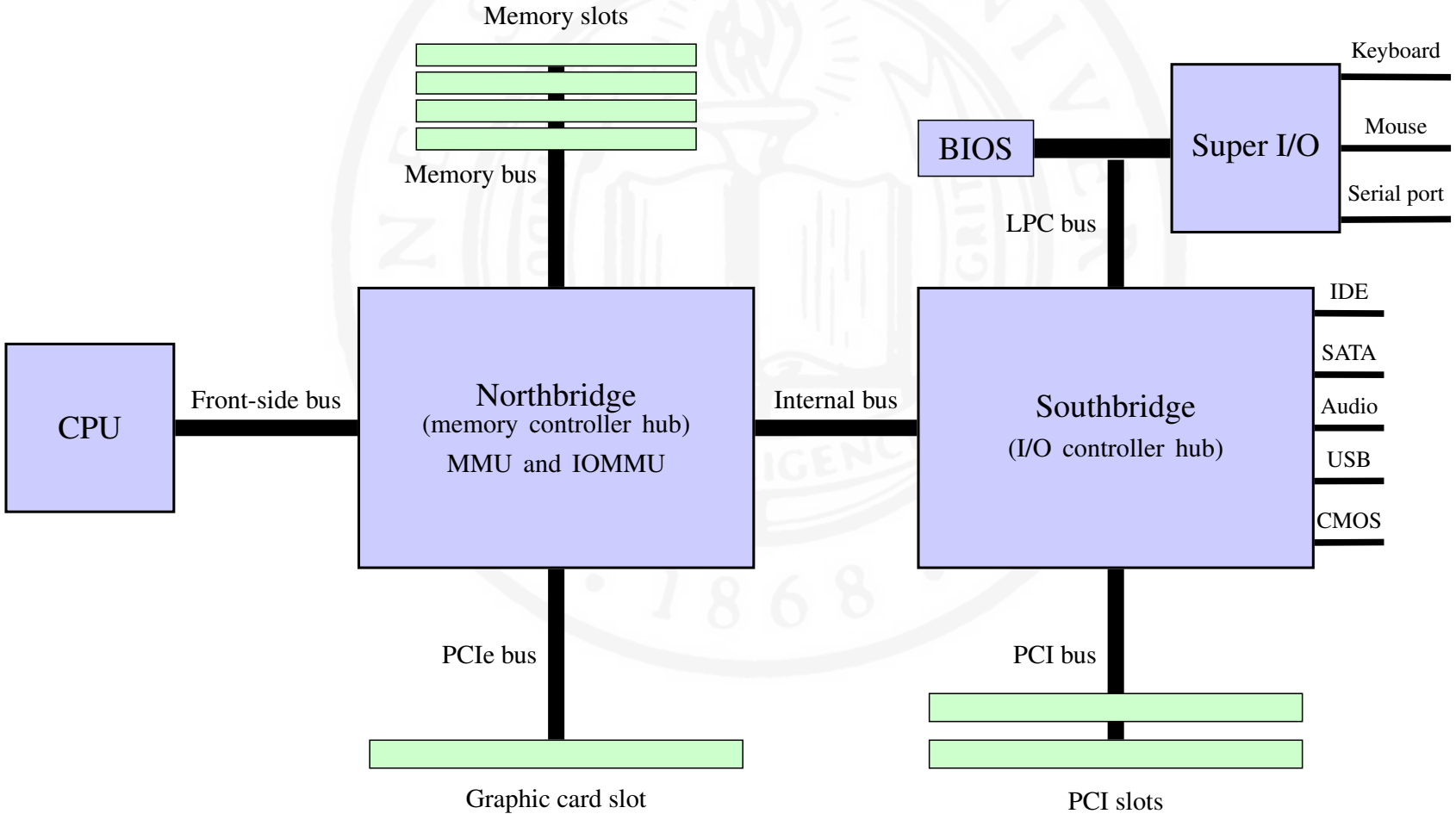
- Software-based: Write to an I/O port specified by Southbridge datasheet (e.g., 0x2B for Intel)
- Hardware-based: Network card, keyboard, hardware timers



Background: Software Layers



Background: Hardware Layout

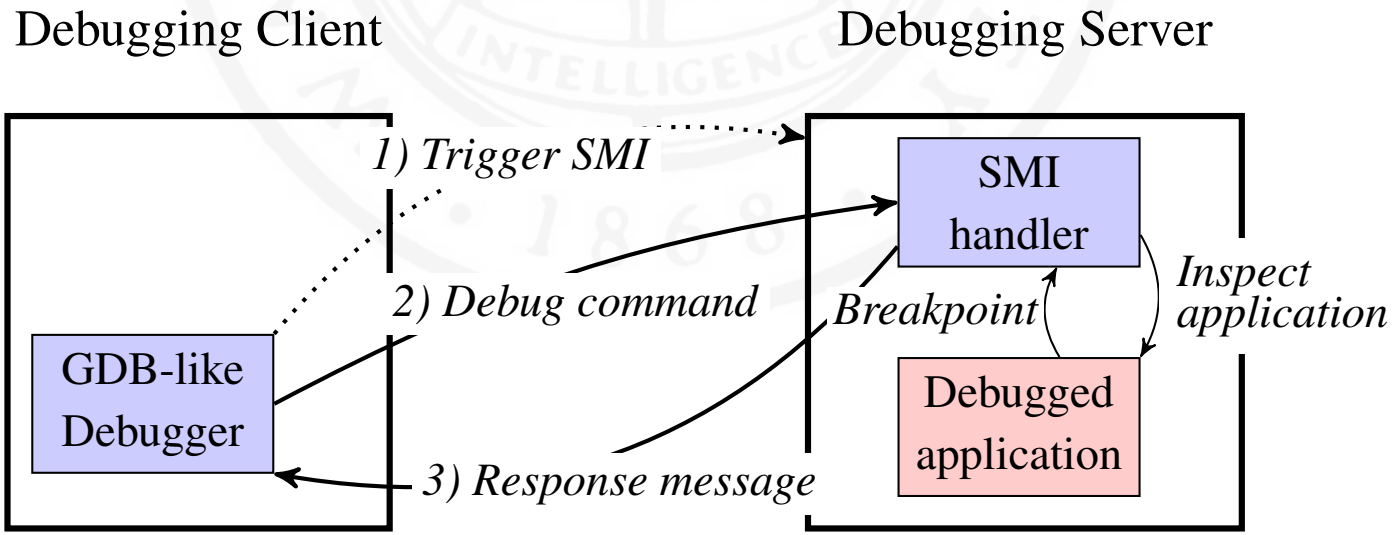


Overview

- Motivation
- Background: System Management Mode (SMM)
- System Architecture
- Evaluation: Transparency and Performance
- Conclusions and Future Directions

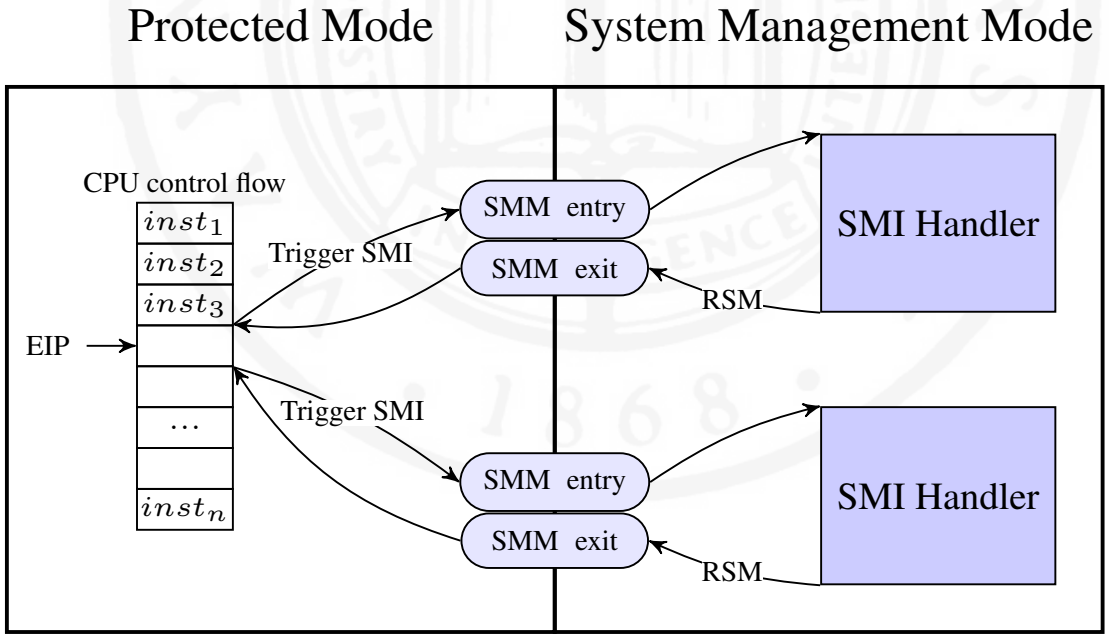
System Architecture

- Traditionally malware debugging uses virtualization or emulation
- MalT debugs malware on a bare-metal machine, and remains transparent in the presence of existing anti-debugging, anti-VM, and anti-emulation techniques.



Step-by-step Debugging in MalT

- Debugging program instruction-by-instruction
- Using performance counters to trigger an SMI for each instruction



Overview

- Motivation
- Background: System Management Mode (SMM)
- System Architecture
- Evaluation: Transparency and Performance
- Conclusions and Future Directions

Evaluation: Transparency Analysis

- Two subjects: 1) running environment and 2) debugger itself
  - Running environments of a debugger
    - SMM v.s. virtualization/emulation
  - Side effects introduced by a debugger itself
    - CPU, cache, memory, I/O, BIOS, and timing
- Towards true transparency
  - MalT is not fully transparent (e.g., external timing attack) but increased
  - Draw attention to hardware-based approach for addressing debugging transparency

Evaluation: Performance Analysis

- Testbed Specification
  - Motherboard: ASUS M2V-MX SE
  - CPU: 2.2 GHz AMD LE-1250
  - Chipsets: AMD K8 Northbridge + VIA VT8237r Southbridge
  - BIOS: Coreboot + SeaBIOS

Table: SMM Switching and Resume (Time: μs)

Operations	**Mean**	**STD**	**95% CI**
SMM switching	3.29	0.08	[3.27,3.32]
SMM resume	4.58	0.10	[4.55,4.61]
Total	7.87		

Evaluation: Performance Analysis

Table: Stepping Overhead on Windows and Linux (Unit: Times of Slowdown)

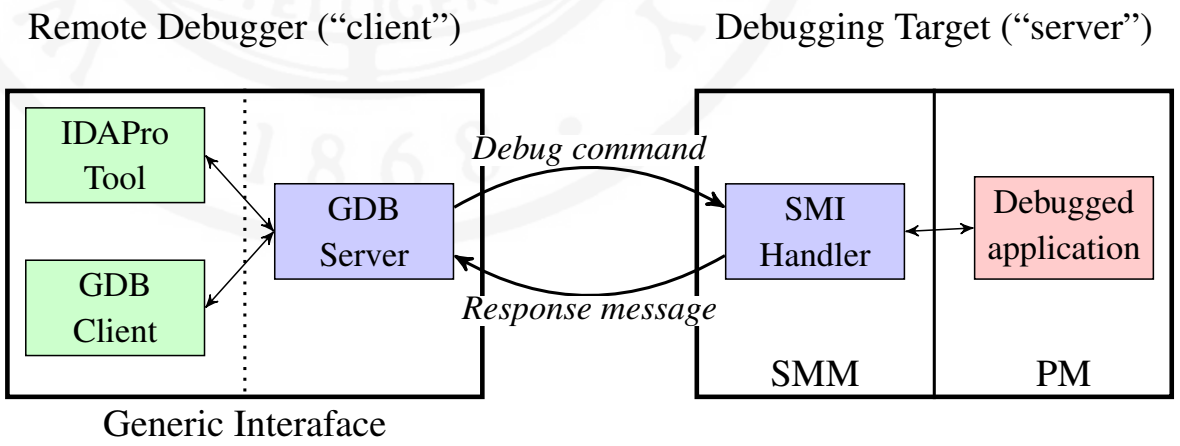
Stepping Methods	**Windows**	**Linux**
	π	π
Far control transfer	2	2
Near return	30	26
Taken branch	565	192
Instruction	973	349

Overview

- Motivation
- Background: System Management Mode (SMM)
- System Architecture
- Evaluation: Transparency and Performance
- Conclusions and Future Directions

Conclusions and Future Work

- We developed MalT, a bare-matal debugging system that employs SMM to analyze malware
  - Hardware-assisted system; does not use virtualization or emulation technology
  - Providing a more transparent execution environment
  - Though testing existing anti-debugging, anti-VM, and anti-emulation techniques, MalT remains transparent
- Future work



References

[1] Symantec, "Internet Security Threat Report, Vol. 19 Main Report," http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v19_21291018.en-us.pdf, 2014.

[2] McAfee, "Threats Report: First Quarter 2014," http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q1-2014-summary.pdf.

[3] Kaspersky Lab, "Kaspersky Security Bulletin 2013," http://media.kaspersky.com/pdf/KSB_2013_EN.pdf.

Paper Discussion

- Nicholas Burton

- MALT is a System Management Mode based debugging framework used to analyze malware. It is a bare metal debugging system that allows high transparency. Bare metal debugging is used because malware often has anti-virtualization measures that change its behavior when it discovers it is in a virtual machine or emulation environment. Using SMM MALT has ring -2 privilege and has a smaller Trusted Code Base than any debugger that depends on virtualization. MALT is an effective debugger that is generally unhindered by armored malware that has anti-VM and anti-debugging software, however it is incapable of debugging rootkits at the ring -2 privilege level. MALT is initially triggered by a serial message arriving at the COM1 port, which has been reconfigured to send an SMI. During debugging the current EIP value is checked against the breakpoint, when they are equal an event in LAPIC is set to overflow to trigger the SMI. Vulnerabilities that SMM has can be used to stop MALT being that it is SMM-based. Attacks such as cache poisoning and memory reclamation, however these issues have been fixed by implementation of SMRR and locking the SMRAM respectively.

Paper Discussion

- Jacob Bednard

- This paper proposes and implements a new technique for transparent debugging based in System Management Mode called MaLT. The motivation for this technique is that malware can detect the presence of virtual machines and emulation and choose to remain stealthy by not unpacking it's contents. MaLT shows that a debugger placed into SMM by Coreboot on boot can remain transparent to malware. In short, the core process of MaLT allows the placement of breakpoints into code that modify the O/S instruction set to call an SMI and open the MaLT environment for introspection. When the current cycle is complete for MaLT, it the next instruction in the registers to resume the previous operation. The benefit that MaLT has is that it operates in Ring -1/-2 space. That is, MaLT operates close to bare metal. The MaLT program can be accessed and used through a serial terminal which allows a user to read memory and launch break points. The only signature that MaLT may leave behind is a side-channel based timing detection method in which malware monitors 3rd party time stamps to see if there have been any breaks in processor execution.

Paper Discussion

- Surya Mani

- This paper talks about the deficiency of advanced malware analysis techniques using virtualization and emulation techniques to prevent malware attack. The malware has the ability to detect the presence of above techniques and hides itself, making the system more vulnerable. The paper discusses in detail about MALT a debugging framework using System Management Mode(SMM). The following are the advantages of using MALT techniques. It is hardware assisted malware analysis which can do rootkit analysis and kernel debugging without using OS. In MALT, either serial port or performance counter is used to trigger SMI (System Management Interrupt) and also uses hardware breakpoint techniques thereby increasing transparency and reducing vulnerability. MALT executes in SSM Ring -2 level hence it is capable of debugging user mode, kernel mode and hypervisor level rootkits. Since MALT code does run in bare metal machine, it does not change any code in operating system. MALT uses reboot approach to restore a system to clean state hence by leaving it vulnerable to malware attack during reboot.

Reminders

- Paper reviews
- Research Topics
- Next Class: Transportation Security
- Next Week