Virtualization-based testing of hardware-software security architecture

9/16/2008
Ruby B. Lee, Jeffrey Dwoskin, Mahadevan Gomathisankaran
(rblee@princeton.edu)

VMworld 2008 – Las Vegas, NV
Academic Summit
Motivation

• Security architectures difficult to test
  – Flexibility to test HW & SW features
  – Test during design phase
  – Controlled environment
  – Check that security properties hold under simulated attacks

• Virtualization environment speeds up development of testing infrastructure
What is a security architecture?

- New trusted hardware components
  - Hardware TCB
  - Protected data
  - New hardware instructions/behavior

- Protect application behavior
  - Verify integrity and confidentiality of code and/or data

- HW protection; SW flexibility
Threats and Attacks

Sample System Architecture

- Protected App
- Normal App
- Operating System
- OS Kernel
- Hardware TCB
  - CPU
  - General Registers
  - Keys
  - Secure Data
- Main memory
- Disk
- Secure Info
- Display, I/O
- Network

Network
- Network attacks
  - App buffer overflows
  - Protocols/MIM
  - Eavesdropping

OS software attacks
- Processes
- Kernel
- System calls
- Interrupts
- Registers
- Memory/TLB
- Devices

Hardware attacks
- Memory
- Disk and Network
- Display, I/O buses
- Processor chip?

HW probe
Testing Framework Goals

• Emulate the behavior of a new security architecture
  – Run real-world operating system and applications
  – Aim for a generic platform for any secure architecture

• Build a testing system that can control the System Under Test (SUT), and observe and modify its behavior

• Launch realistic attacks on hardware & software components
Testing Framework

System Under Test (SUT)  Testing System (TS)

User App  TS Proxy  TS

Linux Guest OS  Attack App  Attack Lib

Security HW Emulator

VMM

Software Events & Attacks
Outline

• Introduction
• Example: SP Architecture
• Emulation of secure architecture features
• Testing Framework architecture
• Testing Framework components
• Conclusions
• VMAP Wishlist
Example: Secret Protection (SP) Architecture

Sensitive App → Trusted Software Module

Operating System

Disk

Processor Chip
- Storage Root Hash
- Device Root Key
- Concealed Execution Mode

Main Memory

User I/O

User App 1

User App 2

User App 1

User App 2

9/16/2008

PRINCETON UNIVERSITY
Example: SP Architecture

- Trusted software modules (TSM)
  - Concealed Execution Mode
- New instructions, hw operations & faults
  - Code integrity checking
  - Secure memory
  - Register protection on interrupt
- Master Secrets
  - Device Root Key (DRK) and Storage Root Hash (SRH)
  - Never leaves chip and accessible only by TSM


J. Dwoskin, R.B. Lee, "Hardware-rooted Trust for Secure Key Management and Transient Trust",
SP Scenario: Emergency Response

Crisis Management Authority

K₁  K₂  ...  Kₙ

SP Device 1

SP Device 2

SP Device n

9/16/2008
**SP Scenario: Emergency Response**

**Remote**
- Crisis Management Authority
  - TSM code
  - Keys
  - Policies
  - Initial data
  - Emergency state

**Local User**
- Requests
- Display data

**SP Device**
- Device Root Key
- Storage Root Hash
- Derived Keys
- Encrypt & MAC
- Keychain
  - Items
  - Keys
  - Policies
  - Data

**Trusted Software Module**

*Princeton University*
SP Architecture Emulation in VMM

- **SP Hardware features**
  - SP instructions, SP registers, and SP faults added to VCPU

- **Concealed Execution Mode**
  - Intercept interrupts/faults to protect TSM state
  - Use binary translation during TSM execution to protect code integrity (future)

- **Secure memory implemented as “Secure Areas”**
Outline

• Introduction
• Example: SP Architecture
• Emulation of secure architecture features
• Testing Framework architecture
• Testing Framework components
• Conclusions
• VMAP Wishlist
Testing Framework Architecture

System Under Test (SUT) vs. Testing System (TS)

- **App**
- **TSM**
- **TS Proxy App**
- **TS Proxy Kernel**
- **TS Kernel**
- **Attack App**
- **Attack Lib**
- **SP HW Emulator**
- **VMM (hooks added)**
- **Event module**
- **Hypercall**
- **Syscall**
- **Upcall**
- **Events**
- **Attacks**

Network (TCP)
Testing Framework Components: Events

• Basic System Architecture
  – Application Level Events
  – OS Events
  – Basic HW Events

• New Security Architecture
  – Trusted Software Module Events
    • Network messages
    • Keychain management
    • Key usage/policy checks
  – SP Events
    • Events generated by executing SP Instructions
    • Faults generated by SP Hardware
    • SP hardware operations
Testing Framework
Components: Attacks

• Attack mechanisms/APIs
  – Hardware
    • Internal processor state and instruction behavior
    • Physical memory
    • I/O devices (raw network, disk access)
  – Software
    • Virtual memory access and translations
    • OS behaviors
    • Application
    • File/socket/process abstractions

• Attack Suites
  – Combine events and individual attack API calls to successfully violate the security of the system.
TF Implementation Example

System Under Test (SUT)

1. Hypercall
2. SP HW Emulator
3. Syscall
4. Network (TCP)
5. TS Attacks

Testing System (TS)

- TS
- Linux
- VMM (hooks added)
- Event module
- TS Proxy Kernel

App

TSM

Linux

TS Proxy
Conclusions

• Rapid prototyping vehicle for black-box or white-box security testing of new architectures
  – Can utilize and *integrate* multiple event sources and attack mechanisms (hardware and software)
  – Can put low-level attacks together to form high level attacks and detect high-level events
• Combined HW + OS + App level attacks are realizable
  – Provides access to TSM & system state at runtime
• Platform to test SP arch. with different apps and TSMs
VMAP Wishlist/Questions

• Documentation (internal-quality)
  – APIs/interfaces for internal modules
  – Key entry points into the source code from guest
  – Data structures: locations, scope, limitations on modification/use

• Features
  – More communication and data sharing between VMM processes
    • VMCI?
    • Track & examine VMM and guest state
    • Temporarily pause/resume VM execution
  – External control of VM behavior
    • Hook into certain VMM and x86 events
    • Hook certain uArch events, such as a cache miss
    • Callbacks or suspend execution
  – Control over Binary Translation & Direct Execution
  – Virtual disk tools to access filesystem without running guest
Q & A