SFI and VMM

Dawn Song

dawnsong@cs.berkeley.edu

Credit: Some slides from John Mitchell

Review

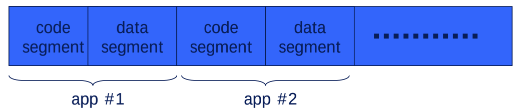
- **Preventing privilege escalation**
  - **Drop privileges asap**
  - **Privilege separation**
- **Sandboxing untrusted code**
  - **System call interposition**
  - **Hardware-based fault isolation**

Software Fault Isolation

- **Idea: insert code in extension code to ensure certain security properties**
- **SFI [Wahbe et. al. 93]**
  - **Software fault isolation**
  - **Security property to guarantee: Extension code only writes and jumps to dedicated data and code region**
  - **How to ensure this?**

Segments

- **Divide application's virtual address space into segments**
  - **With upper bits the same: segment identifier**
- **A fault domain has two segments**
  - **Code segments**
  - **Data segments**
- **Security property to ensure**
  - **Distrusted code only jumps to its code segment, only writes to its data segment**



Idea

- **Locate unsafe instructions: jmp, store**
  - **At compile time, add guards before unsafe instructions to check whether the target is within dedicated region**
  - **When loading code, ensure all guard are present**
- **Optimization:**
  - **instead of checking, simply sets the high-order bits to be segment identifier**
- **Where to store the value of the masks?**
  - **Dedicated registers**
- **How to prevent jumping over the inserted check code?**
  - **Use dedicated registers**

Segment Matching

- **Designed for MIPS processor. Many registers available.**
- **dr1, dr2: dedicated registers not used by binary**
  - **Compiler pretends these registers don't exist**
  - **dr2 contains segment ID**
- **Indirect store instruction** [addr] ← R12 **becomes:**

dr1 ← addr
scratch-reg ← (dr1 >> 20) : get segment ID
compare scratch-reg and dr2 : validate

Address Sandboxing

- dr2: holds segment ID followed by the proper number of zero's
- Indirect store instruction [addr] ← R12 becomes:

dr1 ← addr & segment-mask	: zero out seg bits
dr1 ← dr1 | dr2	: set valid seg ID
[dr1] ← R12	: do store

- Fewer instructions than segment matching
  … but does not catch offending instructions
- Untrusted jump instruction handled similarly
- Why use dedicated register?
- What happens if untrusted code jumps to the middle of the sequence?

Instrumentation and Verification

- Instrumentation
  - Modify gcc compiler to emit encapsulated object code
- Verification
  - Verify when module is loaded
  - Why verification?
    - » Module is untrusted
    - » Verifier can be much simpler than the instrumentor
  - How to verify?
    - » Dedicated registers are only used for the added instrumentations
    - » Each store and jump instruction is properly guarded

SFI Summary

- Security property ensured:
  Distrusted code only jumps to its code segment, only writes to its data segment
- Tradeoff btw computation overhead & communication overhead
- More information:
  - Efficient Software-based Fault Isolation, by Robert Wahbe, Steven Lucco, Thomas Anderson, Susan Graham

Generalization: In-line Reference Monitor

- In-line reference monitors/dynamic checks
  - IRMs enforce security policies by inserting into subject programs the code for validity checks and also any additional state that is needed for enforcement
- Idea
  - Add dynamic checks to enforce properties at run time
  - Combine with static analysis to reduce dynamic checks
  - Ensure dynamic checks are not by-passed
    - » Control & data property enforcements are intertwined
  - Verifier:
    - » Ensure dynamic checks are properly inlined

A Whole Spectrum

- Tradeoff
  - Complexity of properties enforced
  - Runtime overhead
  - Assumptions required
  - Complexity of priori analysis needed
- Properties enforced entail
  - What dynamic checks to add
  - How to add these dynamic checks
- The spectrum
  - SFI, CFI (control flow integrity), DFI (data flow integrity), XFI, …
  - Interpreter/emulator is one end of the spectrum

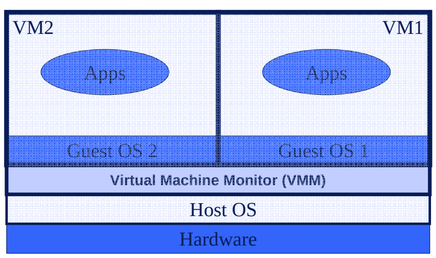
Administravia

- Project 2

Virtual Machine Monitor

- **Virtualization**
  - **Creating a simulated computer environment (Virtual Machine) for the guest software**
  - **Guest software (often including a complete OS) runs as if it's on a stand-alone hardware**
  - **Virtual Machine Monitor (VMM): virtualization platform**
    - » **Also called hypervisors**
- **Hypervisors:**
  - **Type I: runs directly on hardware**
    - » **Guest OS runs at the second level above hardware**
    - » **E.g., VMWare ESX, Microsoft Hyper-V, Xen**
  - **Type II: runs within a host OS**
    - » **Guest OS runs at the third level above hardware**
    - » **E.g., VMWare Workstation, Microsoft Virtual PC, Parallels**

NSA NetTop



- single HW platform used for both classified and unclassified data

History of VM Technology

- **VMs in the 1960's:**
  - **Few computers, lots of users**
  - **VMs allow many users to shares a single computer**
- **VMs 1970's – 2000: non-existent**
- **VMs since 2000:**
  - **Too many computers, too few users**
    - » **Print server, Mail server, Web server, File server, Database server, …**
  - **Wasteful to run each service on a different computer**
    - » **VMs save power while isolating services**

VMM for Security

- **VMM Security assumption:**
  - **Provides isolation**
  - **Malware can infect guest OS and guest apps**
  - **But malware cannot escape from the infected VM**
    - » **Cannot infect Host OS**
    - » **Cannot infect other VMs on the same hardware**
- **Requires that VMM protect itself and is not buggy**
  - **VMM is much simpler than full OS, easier to verify/get right**
- **Natual place to enforce security policies**
  - **Policy checker does not need to rely on security of OS**

Intrusion Detection / Anti-virus

- **Runs as part of OS kernel and user space process**
  - **Kernel root kit can shutdown protection system**
  - **Common practice for modern malware**
- **Standard solution: run IDS system in network**
  - **Problem: insufficient visibility into user's machine**
- **Better: run IDS as part of VMM (protected from malware)**
  - **VMM can monitor virtual hardware for anomalies**
  - **VMI: Virtual Machine Introspection**
    - » **Allows VMM to check Guest OS internals**

Sample Applications (I)

Stealth malware:

- **Creates processes that are invisible to "ps"**
- **Opens sockets that are invisible to "netstat"**

1. Lie detector check

- **Goal: detect stealth malware that hides processes and network activity**
- **Method:**
  - » **VMM lists processes running in GuestOS**
  - » **VMM requests GuestOS to list processes (e.g. ps)**
  - » **If mismatch, kill VM**

Sample Applications (II)

2. Application code integrity detector
- VMM computes hash of user app-code running in VM
- Compare to whitelist of hashes
  - » Kills VM if unknown program appears

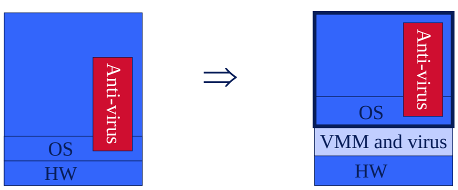
3. Ensure GuestOS kernel integrity
- example: detect changes to sys_call_table

4. Virus signature detector
- Run virus signature detector on GuestOS memory

5. Detect if GuestOS puts NIC in promiscuous mode

VM-based Malware

- **Idea (blue pill/Subvirt):**
  - Once on the victim machine, install a malicious VMM
  - Virus hides in VMM
  - Invisible to virus detector running inside VM



Conclusion

- **SFI**
- **VMM**