Sandboxing and Isolation

Dawn Song
dawnsong@cs.berkeley.edu

Review

- Preventing privilege escalation
  - Drop privileges asap
  - Privilege separation

Sandboxing

- Building a controlled environment for untrusted third-party applications
  - Can only access the given resources
  - Ensure application does not deviate from pre-approved behavior
- Examples
  - Filesystem isolation
  - Disk quotas
  - I/O rate limiting
  - Memory limits
  - CPU quotas
  - Network control and rate limiting

Different Examples

- chroot, BSD jail commands
  - Only for file permissions
  - Coarse grained
- System call interposition
  - More general
- Java virtual machine
  - More fine grained
- Virtual machine
  - Whole system

System Call Interposition

- Malicious programs usually need to make system calls to do harm to the system
- System call interface is a natural place to place security checks & enforce security policies
- What kind of policies do we want to enforce?
  - A process cannot open certain files
  - A process may have restricted network access
  - A process may not send network packets after reading certain files
- Policy can be written as
  - Whether a single action is allowed
  - Whether a sequence of action is allowed
  - An automata

How to Enforce a Policy?

- Intercept system calls
- Information passed on to policy checker before system call is processed
- Policy checker
  - In kernel
  - User space
  - Be careful with race conditions
Sandboxing Case Study: iPhone & Android

- Miller attacks on iPhone & Android
- Security architecture & consequences

iPhone design weakness & consequences

- Security design weakness:
  - All processes of interest run with administrative privileges
- Consequences: iPhone attack (Miller Jul 2007)
  - iPhone Safari downloads malicious web page
  - Arbitrary code is run with administrative privileges
  - Can read SMS log, address book, call history, other data
  - Can perform physical actions on the phone.
    - System sound and vibrate the phone for a second
    - Could dial phone numbers, send text messages, or record audio (as a bugging device)
  - Can transmit any collected data over network to attacker

See http://www.security evaluators.com/iphone/

Android

- Operating system for T-Mobile Google phone
  - Open Handset Alliance
- Miller’s attack: Oct 24, 2008
  - Exploit an unpatched vulnerability
  - Surfing malicious website can exploit browser
- Consequences
  - Get full privilege of the browser
    - E.g., access to cookies, keystrokes entered in browser
  - However, can’t do other things such as dial the phone directly
  - Contrast to iPhone

Android Security Architecture

- Each application runs in its own process
  - Its own Java Virtual Machine
- Application signing
  - Each application (.apk files) is signed
- Sandboxing
  - Each application package (.apk file) installed is given unique userID
  - Can only access to its own data
  - Default setting: no other permission
  - Explicitly declare permission needed at install time and get approval from user
  - Grant data access permission to other processes

Challenges for Sandboxing

- Complete mediation
- Tradeoff between usability/convenience and security

Administrivia

- Midterm Statistics:
  - Mean: 34.29
  - Standard deviation: 7.77
  - 1st quartile: 28.38
  - 2nd quartile (median): 34.25
  - 3rd quartile: 39.75
  - Extra Credit:
    - Mean: 1.48
    - Standard deviation: 1.76
    - 1st quartile: 0
    - 2nd quartile (median): 1
    - 3rd quartile: 2
Administravia
• Slides format?
  – 3 or 6 slides a page
• Project 2

Fault Isolation
• Fault isolation
  – The fault in one program or one part of the code cannot harm other programs or other parts of the code
  – Very important for security in running untrusted or untrustworthy code
  – “Harmness”
    » E.g., read/write memory it’s not supposed to
• Hardware fault isolation
  – Processes
  – What properties/protection does process provide?
    » Memory protection
    » Other resources such as file handles are separated as well
  – Works well for some applications

Disadvantage of Hardware Fault Isolation
• Process inter communication is expensive
  – Add significant performance overhead if often
• Why is process inter communication expensive?
  – Trap from user to kernel back to user
  – Context switch is expensive
    » Flush TLB, cache miss, etc.
  – Often 2-3 orders of magnitude more expensive than normal procedure call

How to Address This?
• Software Fault Isolation (SFI)
• Question: how to protect a piece of code from harming other parts of the program even though they run in the same address space?

Motivation
• Today’s systems are designed to be extensible
  – OS kernel module/drivers
• Extension accounts for over x% of Linux kernel code
  – x=70 [Chou et. al.]
• Windows XP desktops
  – Over 35,000 drivers with over 120,000 versions [Swift et. al.]
• Drivers cause 85% of reported failures in Windows XP [Swift et. al.]

Desired Properties of Extensible Architecture
• Efficiency
• Security model: extension code may be
  – Malicious
  – Buggy
• Protection
  – Extension should not read and/or write to certain regions in host isolation, sandbox
    » Do no harm to others
    » Why do we care about Read?
  – Other more sophisticated security policies
  – Need more efficient mechanisms than hardware 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?