EECS 388 – Winter 2015
Lecture 21: Access Control and Isolation

Access Control

General setting: need to control access to resources (malicious programs, vulnerable programs)

subject: user, process, entity accessing resources
object: file, device, etc. being accessed

Access Control Matrix

	file1	file2
Alice	read	write
Bob	-	read
Charlie	-	-

Principle: "Complete Mediation"

control all resource requests through mediator that checks access control policy

Specifying policies:

ACL (acecss control lists):	Capabilities:
- list associated w/ object	- unforgeable ticket
- user checked against list	- maybe can be passed from process to process
- need to authenticate user	- ticket checker does not need to know user's id
	delegation?
- process run under current user	- process can pass capability at runtime
	revocation?
- remove user/group from list	- possible if bookkeeping

Unix file permissions:

- Processes associated w/ user IDs (can change)
- Files have permission bits, simple ACL:

setid	owner	group	other	
-	rwx	rwx	rwx	} vector of 4 octal value (e.g., chmod file 0644)

- Owner, root can change permissions
- if user == owner => owner permissions
  if user in group => group permissions
  else other permission

Capabilities

Instead of root vs. non-root, allow unprivileged processes to do whitelist of things

CAP_NET_ADMIN, CAP_MKNOD, CAP_SYS_NICE, CAP_SYS_RAWIO

Isolation

Traditional thinking: How to run bad/untrustworthy programs safely?
New thinking: Should be skeptical of all programs, isolate all you can!

Confinement: Keep app from harming rest of system

Implementations:

air gap	virtual machines	syscall interposition
[diagram]	[diagram]	[diagram]

Traditional OS mechanisms:

Process isolation

Separate memory address spaces, explicit IPC

chroot

- used, e.g., for guest access to ftp sites
  - chroot /tmp/guest -- root dir / is mapped to /tmp/guest
  - su guest -- Process ID becomes guest
- apps can't access files outside of jail
- utilities (ls, ps) must live in jail
- network access? (not restricted)
  raw disk devices?
  signals to other processes?

FreeBSD jails

- hardened chroot
- can only bind to authorized sockets
- only communicate w/ processes inside jail

Not all programs can run in a jail (e.g., web browser cannot)

Problems:

- coarse policies
- all-or-nothing filesystem access
- malicious apps can access netowkr, crash host OS

System call interposition

- to damage host system, app makes system calls
  - filesystem: unlink, open, write
  - network: socket, bind, connect, send
- so, monitor syscalls, block unauthorized calls

Linux: ptrace

complications:

- forks? fork monitor
- monitor crash? kill app
- must maintain state

ptrace isn't granular -- all system calls or none

security issues:

process 1: open ("me")
monitor: OK check is not atomic
process 2: link me -> /etc/passwd
OS executes open("me")

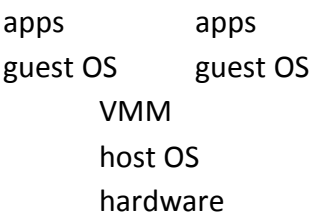
"Time of check vs. time of use" vulnerability (TOCTOU)

Modern variants: (avoid TOCTOU correctly, but how to specify the right policies?)

fine-grained access control + system call interposition
policy defined in external file

e.g., App Armor: profiles app during learning phase, restricts privileges not in profile
e.g., SE Linux: similar, NSA developed

Virtual Machines



security assumptions:

- malware can infect guest OS, guest apps, but can't escape VM to host OS, other VMs
- problems: covert channels, etc.
  malware might detect it's running inside VM, behave differently
  have to monitor, manage, secure many OS instances

Networks:

Physical isolation, VLANs, Firewalls

Sandboxes:

Apply mechanisms like above to individual app or code

Separate tabs, SOP, etc. (browsers becoming more like OS)

Javascript in browser: API sandbox (no calls to do "dangerous" things)
Native client: Run untrusted x86 code in browser (Chrome)
Android, iOS application sandboxes, code signing