# CSci 4271W
# Development of Secure Software Systems
# Day 15: OS Protection and Isolation

Stephen McCamant

University of Minnesota, Computer Science & Engineering

## Outline

Good technical writing, cont'd

Secure OS interaction

OS: protection and isolation

Announcements intermission

More choices for isolation

Bonus: qmail

## Inclusive language

- Avoid words and grammar that implies relevant people are male
- My opinion: avoid using he/him pronouns for unknown people
- Some possible alternatives
  - "he/she"
  - Alternating genders
  - Rewrite to plural and use "they" (may be less clear)
  - Singular "they" (least traditional, but spreading)

## Outline

Good technical writing, cont'd

Secure OS interaction

OS: protection and isolation

Announcements intermission

More choices for isolation

Bonus: qmail

## Avoid special privileges

- Require users to have appropriate permissions
  - Rather than putting trust in programs
- Dangerous pattern 1: setuid/setgid program
- Dangerous pattern 2: privileged daemon
- But, sometimes unavoidable (e.g., email)

## Prefer file descriptors

- Maintain references to files by keeping them open and using file descriptors, rather than by name
- References same contents despite file system changes
- Use `openat`, etc., variants to use FD instead of directory paths

## Prefer absolute paths

- Use full paths (starting with /) for programs and files
- `$PATH` under local user control
- Initial working directory under local user control
  - But FD-like, so can be used in place of `openat` if missing

## Prefer fully trusted paths

- Each directory component in a path must be write protected
- Read-only file in read-only directory can be changed if a parent directory is modified

## Don't separate check from use

- Avoid pattern of e.g., `access` then `open`
- Instead, just handle failure of open
  - You have to do this anyway
- Multiple references allow races
  - And `access` also has a history of bugs

## Be careful with temporary files

- Create files exclusively with tight permissions and never reopen them
  - See detailed recommendations in Wheeler (q.v.)
- Not quite good enough: reopen and check matching device and inode
  - Fails with sufficiently patient attack

## Give up privileges

- Using appropriate combinations of `set*id` functions
  - Alas, details differ between Unix variants
- Best: give up permanently
- Second best: give up temporarily
- Detailed recommendations: Setuid Demystified (USENIX'02)

## Allow-list environment variables

- Can change the behavior of called program in unexpected ways
- Decide which ones are necessary
  - As few as possible
- Save these, remove any others

## For more details. . .

- The external reading on this topic is chapters from a web-hosted book by David A. Wheeler
- Recall reading questions are due Thursday evening

## Outline

Good technical writing, cont'd

Secure OS interaction

OS: protection and isolation

Announcements intermission

More choices for isolation

Bonus: qmail

## OS security topics

- Resource protection
- Process isolation
- User authentication (will cover later)
- Access control (already covered)

## Protection and isolation

- Resource protection: prevent processes from accessing hardware
- Process isolation: prevent processes from interfering with each other
- Design: by default processes can do neither
- Must request access from operating system

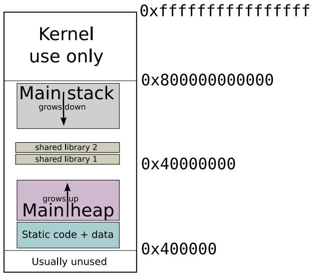
## Reference monitor

- Complete mediation: all accesses are checked
- Tamperproof: the monitor is itself protected from modification
- Small enough to be thoroughly verified

## Hardware basis: memory protection

- Historic: segments
- Modern: paging and page protection
  - Memory divided into pages (e.g. 4k)
  - Every process has own virtual to physical page table
  - Pages also have R/W/X permissions

## Linux example



Kernel use only

Main stack

grows down

shared library 2

shared library 1

grows up

Main heap

Static code + data

Usually unused

0xffffffffffffffff

0x800000000000

0x40000000

0x400000

## Hardware basis: supervisor bit

- Supervisor (kernel) mode: all instructions available
- User mode: no hardware or VM control instructions
- Only way to switch to kernel mode is specified entry point
- Also generalizes to multiple "rings"

## Outline

Good technical writing, cont'd

Secure OS interaction

OS: protection and isolation

Announcements intermission

More choices for isolation

Bonus: qmail

## Prof. McCamant extra office hour Wednesday

- 1:30-2:30pm in 4-225E Keller
- Most demand seems to be about the project, but any topic is OK

## Project questions in lab sections

- Secondary priority compared to discussions about the lab
- May set you behind for the future, but the project is a large part of your grade
- Other students are around, so can't be too spoiler-y

## Check Piazza for project-related materials

- Based on views, not so popular yet
- There now: suggestions, pointer to walk-through video
- We welcome non-spoiler public discussions

## Upcoming project and problem set schedule

- Problem set 2 will be postponed, and not due 11/3
- But we will go straight into project 1 after project 0.5

## Outline

Good technical writing, cont'd

Secure OS interaction

OS: protection and isolation

Announcements intermission

More choices for isolation

Bonus: qmail

## Ideal: least privilege

- Programs and users should have the most limited set of powers needed to do their job
- Presupposes that privileges are suitably divisible
  - Contrast: Unix `root`

## "Trusted", TCB

- In security, "trusted" is a bad word
- $X$ is trusted: $X$ can break your security
- "Untrusted" = okay if it's evil
- Trusted Computing Base (TCB): minimize

## Restricted languages

- Main application: code provided by untrusted parties
- Packet filters in the kernel
- JavaScript in web browsers
  - Also Java, Flash ActionScript, etc.

## SFI

- Software-based Fault Isolation
- Instruction-level rewriting
  - Analogous to but predates control-flow integrity
- Limit memory stores and sometimes loads
- Can't jump out except to designated points
- E.g., Google Native Client

## Separate processes

- OS (and hardware) isolate one process from another
- Pay overhead for creation and communication
- System call interface allows many possibilities for mischief

## System-call interposition

- Trusted process examines syscalls made by untrusted
- Implement via `ptrace` (like strace, gdb) or via kernel change
- Easy policy: deny

## Interposition challenges

- Argument values can change in memory (TOCTTOU)
- OS objects can change (TOCTTOU)
- How to get canonical object identifiers?
- Interposer must accurately model kernel behavior
- Details: Garfinkel (NDSS'03)

## Separate users

- Reuse OS facilities for access control
- Unit of trust: program or application
- Older example: qmail
- Newer example: Android
- Limitation: lots of things available to any user

## `chroot`

- Unix system call to change root directory
- Restrict/virtualize file system access
- Only available to root
- Does not isolate other namespaces

## OS-enabled containers

- One kernel, but virtualizes all namespaces
- FreeBSD jails, Linux LXC, Solaris zones, etc.
- Quite robust, but the full, fixed, kernel is in the TCB

## (System) virtual machines

- Presents hardware-like interface to an untrusted kernel
- Strong isolation, full administrative complexity
- I/O interface looks like a network, etc.

## Virtual machine designs

- (Type 1) hypervisor: 'superkernel' underneath VMs
- Hosted: regular OS underneath VMs
- Paravirtualizaion: modify kernels in VMs for ease of virtualization

## Virtual machine technologies

- Hardware based: fastest, now common
- Partial translation: e.g., original VMware
- Full emulation: e.g. QEMU proper
  - Slowest, but can be a different CPU architecture

## Modern example: Chrom(ium)

- Separates "browser kernel" from less-trusted "rendering engine"
  - Pragmatic, keeps high-risk components together
- Experimented with various Windows and Linux sandboxing techniques
- Blocked 70% of historic vulnerabilities, not all new ones
- `http://seclab.stanford.edu/websec/chromium/`

## Outline

Good technical writing, cont'd

Secure OS interaction

OS: protection and isolation

Announcements intermission

More choices for isolation

Bonus: qmail

## Historical background

- Traditional Unix MTA: Sendmail (BSD)
  - Monolithic setuid root program
  - Designed for a more trusting era
  - In mid-90s, bugs seemed endless
- Spurred development of new, security-oriented replacements
  - Bernstein's qmail
  - Venema et al.'s Postfix

## Distinctive qmail features

- Single, security-oriented developer
- Architecture with separate programs and UIDs
- Replacements for standard libraries
- Deliveries into directories rather than large files

## Ineffective privilege separation

- Example: prevent Netscape DNS helper from accessing local file system
- Before: bug in DNS code
  - → read user's private files
- After: bug in DNS code
  - → inject bogus DNS results
  - → man-in-the-middle attack
  - → read user's private web data

## Effective privilege separation

- Transformations with constrained I/O
- General argument: worst adversary can do is control output
  - Which is just the benign functionality
- MTA header parsing (Sendmail bug)
- `jpegtopnm` inside `xloadimage`

## Eliminating bugs

- Enforce explicit data flow
- Simplify integer semantics
- Avoid parsing
- Generalize from errors to inputs

## Eliminating code

- Identify common functions
- Automatically handle errors
- Reuse network tools
- Reuse access controls
- Reuse the filesystem

## The "qmail security guarantee"

- $500, later $1000 offered for security bug
- Never paid out
- Issues proposed:
  - Memory exhaustion DoS
  - Overflow of signed integer indexes
- Defensiveness does not encourage more submissions

## qmail today

- Originally had terms that prohibited modified redistribution
  - Now true public domain
- Latest release from Bernstein: 1998; netqmail: 2007
- Does not have large market share
- All MTAs, even Sendmail, are more secure now