CSci 5271
Introduction to Computer Security
Day 8: Defensive programming and design, part 2
Stephen McCamant
University of Minnesota, Computer Science & Engineering

Outline
- Bernstein's perspective
- Announcements intermission
- Techniques for privilege separation

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
- Does not have large market share
- All MTAs, even Sendmail, are more secure now

Outline
- Bernstein’s perspective
- Announcements intermission
- Techniques for privilege separation

Note to early readers
- This is the section of the slides most likely to change in the final version
- If class has already happened, make sure you have the latest slides for announcements

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 like (but predates) CFI
- 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
- Paravirtualization: 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

Next time

- Protection and isolation
- Basic (e.g., classic Unix) access control