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

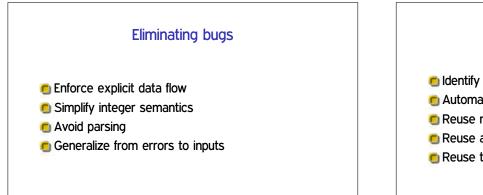
- Example: prevent Netscape DNS helper from accessing local file system
 Before: bug in DNS code

 → read user's private files

 After: bug in DNS code
 - \rightarrow inject bogus DNS results
 - \rightarrow man-in-the-middle attack
 - \rightarrow read user's private web data

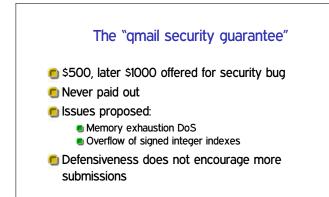
Effective privilege separation

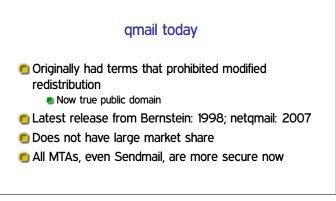
- Transformations with constrained I/O
- General argument: worst adversary can do is control output
 - Which is just the benign functionality
- 🖲 MTA header parsing (Sendmail bug)
- 🖲 jpegtopnm **inside** xloadimage



Eliminating code

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







Announcements intermission

Techniques for privilege separation

- 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

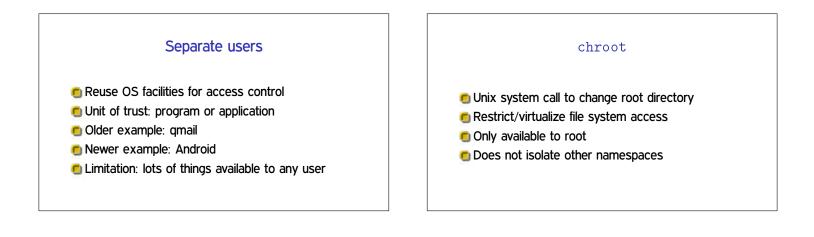
- OS (and hardware) isolate one process from another
- Pay overhead for creation and communication
- System call interface allows many possibilities for mischief

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

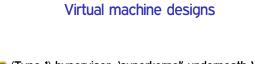


OS-enabled containers

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.



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



- 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/

Next time

Protection and isolation

Basic (e.g., classic Unix) access control