CSci 4271W
Development of Secure Software Systems
Day 12: OS auditing and isolation

Stephen McCamant

University of Minnesota, Computer Science & Engineering

Outline

Exercise: auditing for OS-related bugs

OS: protection and isolation

More choices for isolation

Next: the web from a security perspective

Understanding the OS context

- Which code is running with privileges?
- Which parts of the environment are untrusted?
- Which directories are trusted or untrusted?

Common problems to look for

- Attacker-controlled shell commands
- Effects of attacker-controlled environment
- TOCTTOU vulnerabilities in filesystem checks
- Races in filesystem modifications

OS context for BCLPR

- Printer management on system with untrusted users
- BCLPR binary is setuid root
- Printer-related directories under `/var/bclpr` are trusted
- Normal usage: print a user's own text or PDF file

Generic UNIX local threat model

- Ultimate attacker goal of privilege escalation to root
- Direct: inject shellcode into setuid program
- Examples of indirect attacks:
  - Write privileged config file (e.g., `/etc/passwd`, `root crontab`)
  - Read secret config file (e.g., `root` SSH private key)
  - Set an attacker binary to be setuid `root`
  - (Trick human sysadmin into doing something)

Your task for BCLPR

- Find places in the code that indicate OS-related vulnerabilities
- Prioritize by which are most likely/easiest to exploit
- Make list of line numbers and bug types to share via chat

Outline

Exercise: auditing for OS-related bugs

OS: protection and isolation

More choices for isolation

Next: the web from a security perspective

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

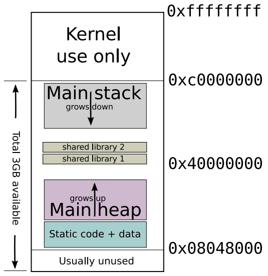
Reference monitor

- 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 32-bit example



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

Exercise: auditing for OS-related bugs

OS: protection and isolation

More choices for isolation

Next: the web from a security perspective

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

Exercise: auditing for OS-related bugs

OS: protection and isolation

More choices for isolation

Next: the web from a security perspective

Once upon a time: the static web

- HTTP: stateless file download protocol
  - TCP, usually using port 80
- HTML: markup language for text with formatting and links
- All pages public, so no need for authentication or encryption

Web applications

- The modern web depends heavily on active software
- Static pages have ads, paywalls, or "Edit" buttons
- Many web sites are primarily forms or storefronts
- Web hosted versions of desktop apps like word processing

Server programs

- Could be anything that outputs HTML
- In practice, heavy use of databases and frameworks
- Wide variety of commercial, open-source, and custom-written
- Flexible scripting languages for ease of development
  - PHP, Ruby, Perl, etc.

JavaScript and the DOM

- JavaScript (JS) is a dynamically-typed prototype-OO language
  - No real similarity with Java
- Document Object Model (DOM): lets JS interact with pages and the browser
- Extensive security checks for untrusted-code model

GET, POST, and cookies

- `GET` request loads a URL, may have parameters delimited with `?`, `&`, `=`
  - Standard: should not have side-effects
- `POST` request originally for forms
  - Can be larger, more hidden, have side-effects
- *Cookie*: small token chosen by server, sent back on subsequent requests to same domain

Client-side programming

- Java: nice language, mostly moved to other uses
- ActiveX: Windows-only binaries, no sandboxing
  - Glad to see it on the way out
- Flash and Silverlight: most important use is DRM-ed video
- Core language: JavaScript

Same-origin policy

- *Origin* is a tuple (scheme, host, port)
  - E.g., (http, www.umn.edu, 80)
- Basic JS rule: interaction is allowed only with the same origin
- Different sites are (mostly) isolated applications

User and attack models

- "Web attacker" owns their own site (`www.attacker.com`)
  - And users sometimes visit it
  - Realistic reasons: ads, SEO
- "Network attacker" can view and sniff unencrypted data
  - Unprotected coffee shop WiFi