Outline

- Techniques for privilege separation
- OS security: protection and isolation
- OS security: authentication
- Announcements intermission
- Basics of access control
- Unix-style access control

Last time

- Restrict languages, SFI
- Separate process, syscall interposition
- Separate user, chroot
- OS containers

(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

Outline

- Techniques for privilege separation
- OS security: protection and isolation
- OS security: authentication
- Announcements intermission
- Basics of access control
- Unix-style access control

OS security topics

- Resource protection
- Process isolation
- User authentication
- Access control

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
- Techniques for privilege separation
- OS security: protection and isolation
- OS security: authentication
- Announcements intermission
- Basics of access control
- Unix-style access control

Authentication factors
- Something you know (password, PIN)
- Something you have (e.g., smart card)
- Something you are (biometrics)
- CAPTCHAs, time and location, ...
- Multi-factor authentication

Passwords: love to hate
- Many problems for users, sysadmins, researchers
- But familiar and near-zero cost of entry
- User-chosen passwords proliferate for low-stakes web site authentication

Password entropy
- Model password choice as probabilistic process
- If uniform, $\log_2 |S|$ controls difficulty of guessing attacks
- Hard to estimate for user-chosen passwords
  - Length is an imperfect proxy
**Password hashing**

- Idea: don’t store password or equivalent information
- Password ‘encryption’ is a long-standing misnomer
  - E.g., Unix crypt(3)
- Presumably hard-to-invert function \( h \)
- Store only \( h(\text{p}) \)

**Dictionary attacks**

- Online: send guesses to server
- Offline: attacker can check guesses internally
- Specialized password lists more effective than literal dictionaries
  - Also generation algorithms (\(s \rightarrow s\), etc.)
- \( \sim 25\% \) of passwords consistently vulnerable

**Better password hashing**

- Generate random salt \( s \), store \((s, h(s, p))\)
  - Block pre-computed tables and equality inferences
  - Salt must also have enough entropy
- Deliberately expensive hash function
  - AKA password-based key derivation function (PBKDF)
  - Requirement for time and/or space

**Password usability**

- User compliance can be a major challenge
  - Often caused by unrealistic demands
- Distributed random passwords usually unrealistic
- Password aging: not too frequently
- Never have a fixed default password in a product

**Backup authentication**

- Desire: unassisted recovery from forgotten password
- Fall back to other presumed-authentic channel
  - Email, cell phone
- Harder to forget (but less secret) shared information
  - Mother’s maiden name, first pet’s name
- Brittle: ask Sarah Palin or Mat Honan

**Centralized authentication**

- Enterprise-wide (e.g., UMN ID)
- Anderson: Microsoft Passport
- Today: Facebook Connect, Google ID
- May or may not be single-sign-on (SSO)
Biometric authentication

- Authenticate by a physical body attribute
  + Hard to lose
  - Hard to reset
  - Inherently statistical
  - Variation among people

Example biometrics

- (Handwritten) signatures
- Fingerprints, hand geometry
- Face and voice recognition
- Iris codes

Error rates: ROC curve

Outline

- Techniques for privilege separation
- OS security: protection and isolation
- OS security: authentication
- Announcements intermission
- Basics of access control
- Unix-style access control

Deadlines reminder

- Project progress reports: tonight
- HW1 final submission: Friday night
- Exercise set 2: week from Thursday

HW1 GDB pitfalls

- By default, does not follow fork
  set follow-fork-mode child
- Setuid bit ignored under GDB
  Partial workaround: run GDB and BCVS as root
- Stack locations change when run under GDB
  Make exploit more flexible w.r.t. position
  Search for position that works
Changed office hours this Thu/Fri

- John is traveling
  - Should have already submitted HW1 groups
- Thursday 10-11am will substituted by Stephen in his office 4-225E
- Friday will substituted by Mike, 3-4pm in 2-209

Outline

- Techniques for privilege separation
- OS security: protection and isolation
- OS security: authentication
- Announcements intermission
- Basics of access control
- Unix-style access control

Mechanism and policy

- Decision-making aspect of OS
- Should subject $S$ (user or process) be allowed to access object (e.g., file) $O$?
- Complex, since admin must specify what should happen

Access control matrix

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
<th>Carol</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>rw</td>
<td>r</td>
</tr>
<tr>
<td>rw</td>
<td>-</td>
<td>r</td>
</tr>
<tr>
<td>r</td>
<td>-</td>
<td>rx</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>grades.txt</th>
<th>/dev/hda</th>
<th>/opt/bcvs/bcvs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>r</td>
<td>rw</td>
</tr>
<tr>
<td>Bob</td>
<td>rw</td>
<td>-</td>
</tr>
<tr>
<td>Carol</td>
<td>r</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rx</td>
</tr>
</tbody>
</table>

Slicing the matrix

- $O(nm)$ matrix impractical to store, much less administer
- Columns: access control list (ACL)
  - Convenient to store with object
  - E.g., Unix file permissions
- Rows: capabilities
  - Convenient to store by subject
  - E.g., Unix file descriptors

Groups/roles

- Simplify by factoring out commonality
- Before: users have permissions
- After: users have roles, roles have permissions
- Simple example: Unix groups
- Complex versions called role-based access control (RBAC)
Outline

Techniques for privilege separation
OS security: protection and isolation
OS security: authentication
Announcements intermission
Basics of access control
Unix-style access control

UIDs and GIDs

- To kernel, users and groups are just numeric identifiers
- Names are a user-space nicety
  - E.g., `/etc/passwd` mapping
- Historically 16-bit, now 32
- User 0 is the special superuser `root`
  - Exempt from all access control checks

File mode bits

- Core permissions are 9 bits, three groups of three
- Read, write, execute for user, group, other
- `ls` format: `rwx r-x r--`
- Octal format: `0754`

Interpretation of mode bits

- File also has one user and group ID
- Choose one set of bits
  - If users match, use user bits
  - If subject is in the group, use group bits
  - Otherwise, use other bits
- Note no fallback, so can stop yourself or have negative groups
  - But usually, `O ⊆ G ⊆ U`

Directory mode bits

- Same bits, slightly different interpretation
- Read: list contents (e.g., `ls`)
- Write: add or delete files
- Execute: traverse
- `X` but not `R` means: have to know the names

Process UIDs and `setuid(2)`

- UID is inherited by child processes, and an unprivileged process can’t change it
- But there are syscalls root can use to change the UID, starting with `setuid`
  - E.g., login program, SSH server
Setuid programs, different UIDs

- If 04000 “setuid” bit set, newly exec’d process will take UID of its file owner
- Other side conditions, like process not traced
- Specifically the effective UID is changed, while the real UID is unchanged
  - Shows who called you, allows switching back

More different UIDs

- Two mechanisms for temporary switching:
  - Swap real UID and effective UID (BSD)
  - Remember saved UID, allow switching to it (System V)
- Modern systems support both mechanisms at the same time
- Linux only: file-system UID
  - Once used for NFS servers, now mostly obsolete

Setgid, games

- Setgid bit 02000 mostly analogous to setuid
- But note no supergroup, so UID 0 is still special
- Classic application: setgid games for managing high-score files

Special case: /tmp

- We’d like to allow anyone to make files in /tmp
- So, everyone should have write permission
- But don’t want Alice deleting Bob’s files
- Solution: “sticky bit” 01000

Special case: group inheritance

- When using group to manage permissions, want a whole tree to have a single group
- When 02000 bit set, newly created entries will have the parent’s group
  - (Historic BSD behavior)
- Also, directories will themselves inherit 02000

Other permission rules

- Only file owner or root can change permissions
- Only root can change file owner
  - Former System V behavior: “give away chown”
- Setuid/gid bits cleared on chown
  - Set owner first, then enable setuid
Non-checks

- File permissions on `stat`
- File permissions on link, unlink, rename
- File permissions on read, write
- Parent directory permissions generally
  - Except traversal
  - I.e., permissions not automatically recursive

"POSIX" ACLs

- Based on a withdrawn standardization
- More flexible permissions, still fairly Unix-like
- Multiple user and group entries
  - Decision still based on one entry
- Default ACLs: generalize group inheritance
- Command line: `getfacl, setfacl`

ACL legacy interactions

- Hard problem: don’t break security of legacy code
  - Suggests: “fail closed”
- Contrary pressure: don’t want to break functionality
  - Suggests: “fail open”
- POSIX ACL design: old group permission bits are a mask on all novel permissions

“POSIX” “capabilities”

- Divide root privilege into smaller (~35) pieces
- Note: not real capabilities
- First runtime only, then added to FS similar to setuid
- Motivating example: ping
- Also allows permanent disabling

Privilege escalation dangers

- Many pieces of the root privilege are enough to regain the whole thing
- Access to files as UID 0
- CAP_DAC_OVERRIDE
- CAP_FOWNER
- CAP_SYS_MODULE
- CAP_MKNOD
- CAP_PTRACE
- CAP_SYS_ADMIN (mount)

Legacy interaction dangers

- Former bug: take away capability to drop privileges
- Use of temporary files by no-longer setuid programs
- For more details: “Exploiting capabilities”, Emeric Nasi
Next time

- Object capability systems
- Mandatory access control
- Information-flow security