Protection

CSE473 - Spring 2008

Professor Jaeger

www.cse.psu.edu/~tjaeger/cse473-s08/
Protection

• Protect yourself from untrustworthy users in a common space
  – They may try to access your resources
  – Or modify your resources
  – Or they may just make a mistake
    • Protect yourself from their errors
Shared Platforms

Operating System

Security

Scheduling

Resource Mechanisms

Memory
Disk
Network
Display

Memory Device
Disk Device
Network Device
Display Device

Process 1
Program
Data

Process 2
Program
Data

Process n
Program
Data

...
Access Control/Authorization

• An *access control* system determines what rights a particular entity has for a set of objects

• It answers the question
  – E.g., do you have the right to read `/etc/passwd`
  – Does *Alice* have the right to *view* the EECS website?
  – Do *students* have the right to *share* project data?
  – Does *Dr. Jaeger* have the right to *change* your *grades*?

• An *Access Control Policy* answers these questions
Basic Access Control

- **Subjects** are the active entities that do things
  - E.g., you, Alice, students, Dr. Jaeger

- **Objects** are passive things that things are done to
  - E.g., /etc/passwd, CSE website, project data, grades

- **Operations** are actions that are taken
  - E.g., read, view, share, change
Protection System

• Any “system” that provides resources to multiple subjects needs to control access among them
  – Operating system
  – Servers

• Consists of:
  – Protection State
    • Description of permission assignments (i.e., policy)
    • Determines protection from others
  – Protection State Operations
    • Modify that state
Access Matrix

- **Subjects**
- **Objects**
- **Operations**
- **Can determine**
  - Who can access an object
  - What objects can be accessed by a subject
  - What operations a subject can perform on an object

<table>
<thead>
<tr>
<th></th>
<th>O₁</th>
<th>O₂</th>
<th>O₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₁</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>S₂</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>S₃</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
Access Control

- Suppose the private key file for J is object $O_1$
  - Only J can read
- Suppose the public key file for J is object $O_2$
  - All can read, only J can modify
- Suppose all can read and write from object $O_3$
- What's the access matrix?

<table>
<thead>
<tr>
<th></th>
<th>$O_1$</th>
<th>$O_2$</th>
<th>$O_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>$S_2$</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>$S_3$</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
Least Privilege

- Limit permissions to those required and no more
- Consider three processes for user J
  - Restrict privilege of the process J_1 to prevent leaks

<table>
<thead>
<tr>
<th></th>
<th>O_1</th>
<th>O_2</th>
<th>O_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>J_1</td>
<td>R</td>
<td>R</td>
<td>N?</td>
</tr>
<tr>
<td>J_2</td>
<td>N</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R</td>
<td>W</td>
</tr>
<tr>
<td>J_3</td>
<td>N</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R</td>
<td>W</td>
</tr>
</tbody>
</table>
Protection Domains

- The protection domain restricts access of external parties to our computing system’s resources
- How is this done today?
  - Memory protection
  - E.g., UNIX protected memory, file-system permissions (rwx...)
- A protection state describes access of all programs
Protection State Transitions

• Transition
  – From one access matrix state to another
  – Add/delete subject, object, operation assignment

• Transition semantics
  – Owner-driven
  – Delegation
  – Administrator-driven
  – Administrative permissions

• Attenuation of Rights Principle
  – Can’t grant a right that you do not possess
UNIX System

- Originated in the late 60’s, early 70’s
  - Bell Labs: Ken Thompson, Dennis Ritchie, Douglas McIlroy

- Multiuser Operating System
  - Enables protection from other users
  - Enables protection of system services from users
UNIX Protection State

• **Subjects**
  – Users -- UIDs: real, effective, file system, saved
  – Groups -- GIDs
  – Processes make accesses on behalf of users belonging to particular groups

• **Objects**
  – Files
  – Directories

• **Operations**
  – Read
  – Write
  – Execute
UNIX UIDs

- **UIDs:** *real, effective, file system, saved*
- **UID transitions**
  - For *login* process: UIDs are root
  - After authentication, the shell’s UIDs are: tjaeger
  - Exec su: real is tjaeger; effective is root
- **Transitions among UIDs are complex**

![Finite State Automata](image-url)
UNIX Access Control

• Write the access matrix (access type x object) mc当地an, fcse, and world for the following files:

  drwxr-xr-x   26 mc当地an fcse  884 Feb 21 11:56 slides/
  -rw-r--r--   1 mc当地an fcse  7098 Feb 20 16:01 www/index.html

• Assume “slides” is an O₁ and index.html is O₂

<table>
<thead>
<tr>
<th></th>
<th>R</th>
<th>W</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>O₁</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>O₂</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
Changing permissions

• Change permissions of a file
  – chmod
    • chmod 644 file -- owner can read/write, group, others can read only
    • chmod u+x file -- adds execute permission for owner

• Change owner of a file
  – chown
    • chown new_owner file

• Change group of a file
  – chgrp
    • chgrp new_group file
UID Transition: Setuid

- A special bit in the mode bits
- Execute file
  - Resulting process has the effective (and fs) UID/GID of file owner
- Enables a user to escalate privilege
  - For executing a trusted service
- User defines execution environment
  - e.g., Environment variables
- Service must protect itself or user can gain root access
Setuid Execution

• Process A running as
  – UID=X

• Fork process A to create process B
  – Both running with UID=X

• The exec file C in process B with setuid bit set and owner of root
  – process A has UID=X
  – process B has UID=root
Confused Deputy Problem

• Situation
  – A program has authority (setuid root file)
  – Is confused into using that authority incorrectly

• Example
  – Call httpd and supply libexecdir argument
  – Add your own libraries to overwrite passwd (if httpd runs as root)

• Also a concern for network daemons
  – Why?

• A motivation for capability systems
  – Discuss later
Windows Security

• 0 to full speed
  – No protection system in early versions

• Advantage
  – Know the limits of the UNIX security model
    • What are these?

• Disadvantage
  – Legacy approaches from insecure environment
    • Will they conflict with new protection system?
Windows Subjects (Access Tokens)

- User SID (subject identifier)
  - Authenticated SID

- Group and Alias SIDs
  - Groups and Aliases that apply to this user

- Privileges
  - Ad hoc rights
    - E.g., Take ownership of files
    - Like POSIX capabilities in UNIX

- Defaults for New Objects
  - Access rights for new objects created (like umask)

- Miscellaneous
  - login session ID
  - token ID
Access Checking with ACEs

• Example
Windows Vista

• Despite this expressive model, Windows suffers from many security problems
  – Does anyone really use it?
• Are the right problems being addressed?
  – Are the threats from other users?
• Vista considers a different threat
  – The source of programs on your system
• Vista defines a set of *integrity levels*
  – Program at one level cannot *write* data at a higher level
  – Six levels
    • installer, system, high (administrator), medium (normal user), low (internet), untrusted
Confused Deputy Problem, Again

• Imagine a multi-client server
  – Each client has a different set of objects that they can access

• In an ACL system, the server always has access to all the objects
  – What happens if a client tricks the server into accessing into another client’s objects?
  – Shouldn’t the server only have access to that client’s objects for its requests?
Real OS Capabilities

- The OS kernel manages capabilities in the process table, out of reach of the process
- Capabilities added by user requests (that comply with policy)
A (fictional) Capability Example

• We use the “ls -lt” command to view the contents of our home directory in a OS implementing capabilities:
  – Initially, our shell process has RWX capabilities for our home directory, and RX capabilities for all the directories to the root.
  – The “ls -lt” command is exec()ed, and the shell delegates the directory permissions by giving “ls” the capabilities
    • Note that the capabilities are _not_ tied to any subject
  – The “ls -lt” process exercises the rights to read the directories structure all the way down to the local
  – Of course, the “ls -lt” process now need to obtain read rights to the files (to get their specific meta-information), and obtains them by appealing to the security manager (in kernel) -- the request fulfills the policy, and they are added and exercised
    – The “ls -lt” uses access rights given to the terminal to write output
• Note: there are many ways that the policy can be implemented, rights handed off, etc. We will talk about a couple in the following discussions.
Unforgeability

• Anyone can give their capabilities to anyone else
  – Well, anyone they have the capability to send messages to
  – This will present some problems later...

• So capabilities are analogous to house keys
  – Access to a capability permit its associated operations

• To get any real security, they have to be unforgeable
  – Hardware tags (to protect capabilities)
  – Protected address space/registers
  – Language based techniques
    • Enforce access restrictions on caps.
  – Cryptography
    • Make them unforgeable
Revocation

- Anyone can give their capabilities to anyone else
  - Well, anyone they have the capability to send messages to
  - I can give the capabilities you gave to me to someone else
  - And so on...

- To get any real security, they have to be revocable
  - Without deleting the object
  - E.g., Give them an indirect capability
    - They can access the entry where the house key is stored
  - You can then revoke the indirect capability
    - All copies become useless
Take Away

• Protection is defined by a **Protection System**
  – **Protection States** define what can be done by all subjects
    • Results in **protection domains**
  – **Protection State Transition** enable changes in the state

• Two forms of protection states
  – Access Control Lists
    • UNIX, Windows, most other systems
  – Capabilities
    • Used within systems (UNIX file descriptors)

• Result
  – You can define a security policy
  – You can enforce it
  – But, you are not building a **secure system** -- stay tuned