Access Control

• Describe the permissions available to computing processes
  – Originally, all permissions were available
• Clearly, some controls are necessary
  – Prevent bugs in one process from breaking another
• But, what should determine access?
Permissions for Processes

• What permissions should be granted to...
  – An editor process?
  – An editor process that you run?
  – An editor process that someone else runs?
  – An editor process that contains malware?
  – An editor process used to edit a password file?

• Q: How do we determine/describe the permissions available to processes?

• Q: How are they enforced?

• Q: How might they change over time?
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 how security goals are met
  – Enforcement mechanism
    • Enforce protection state on “system”
Protection State

• Describes the conditions under which the system is **secure**
  – Secrecy
  – Integrity
  – Availability

• Described in terms of
  – **Subjects**: Users and processes
  – **Objects**: Files and sockets
  – **Operations**: Read and write
### 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?

### Recall
- Access Control Lists
- Capabilities

<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 Matrix Policy

- 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>
Options for Subjects

• Possible subjects
UNIX Protection State

• Subjects
  – Users
  – Groups
  – Processes make accesses on behalf of users belonging to particular groups

• Objects
  – Files
  – Directories

• Operations
  – Read
  – Write
  – Execute
Subjects

• Process
  – User ID (UID)
  – Group ID (GID)
  – Supplementary Groups

• Command: id
  – Provide info for that shell
UNIX represents subjects with a combination of UIDs

- Effective UID/GID -- used for access control
- Real UID/GID -- identify real owner of a process - control signals
- Saved UID -- privileged process - lower privilege temporarily
- File system UID -- reduce permission to file system

UID transitions

- For `login` process: UIDs are root
- After authentication, the shell’s UIDs are: tjaeger
- Exec su: real is tjaeger; effective is root

(c) An FSA describing `setresuid` in Linux
Groups

- Users belong to one or more groups
- **Primary group**: defined in `/etc/passwd` - becomes GID
- All **supplementary groups**: defined in `/etc/group`
  - `group_name:group_passwd:GID:list_of_users`

- Commands to change group membership
  - e.g., `newgrp`
- Group membership gives additional permissions beyond UID
UNIX Mode Bits

• Operations
  – Read, write, execute

• Users
  – Owner, Group, World

• File
  – Owner UID, Owner GID

• File type
  – Semantics of operations
    • Based on file type
  – Different semantics between files and dirs
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 `passwd` in process B with setuid bit set and owner of root
  - process A has UID=X
  - process B has UID=root
UNIX Limitations

- How do I create a subject with no permissions?
  - You don’t

- How do I give one person access to a file?
  - Make them owner
  - Make a group of one
  - However, group creation is privileged

- How do I give all but one user access to a file?
  - You don’t

- Setuid - root or user

- UNIX model is easy to use
  - But, you can’t express every case
Windows Protection System

- What we will discuss was designed for Windows 2000
- 0 to full speed
  - No protection system in early versions
- Advantage
  - Know the limits of the UNIX security model
- Disadvantage
  - Legacy approaches from insecure environment
    - Will they conflict with new protection system?
- Protection State
  - Fine-grained access control model
    - Flexible, but complex
  - Flexible definition of subjects and objects
  - Extensible set of operations
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
Windows Objects

• Many types
  – Executive (processes and threads)
  – Filesystem (files and directories)
  – Others (Registry keys and devices)

• Securable objects have a security descriptor
  – Owner SID
    • READ_CONTROL: read access to security descriptor
    • WRITE_DAC: write access to DACL
  – Primary group
    • Compliance
  – Discretionary ACL
    • Permissions
  – System ACL
    • Audit policy
Windows Objects -- Active Directory

- Tree of typed objects
  - Extensible set of object types

- Object Types
  - A set of “properties” (attributes)
  - A globally unique ID for each type
  - Even properties have GUIDs

- “Directories” are containers of objects
  - May contain objects of different types

- Access expressed on containers or objects
  - Objects inherit access rights of containers
  - Amazingly complex combinations!
Windows Permissions

• Permissions
  – To display permissions for a file
    • Select file, properties, security

• Standard access rights
  – Apply to most objects
  – Delete, write owner, synchronize, read control, and write dac

• Otherwise, specific access rights for each type (2000)
  – Some generic rights to build on (e.g., read, write, all)

• Access rights are stored in an access mask form
  – 32-bit consisting of
    • type-specific rights
    • standard rights (above)
    • generic rights (read, write, etc)
Access Control Entries

• DACL in the security descriptor of an object
  – List of access control entries (ACEs)
• ACE structure (proposed by Swift et al)
  – Type (grant or deny)
  – Flags
  – Object Type: global UID for type (limit ACEs checked)
  – InheritedObjectType: complex inheritance
  – Access rights: access mask
  – Principal SID: principal the ACE applies to
• Checking algorithm
  – ACE matches SID (user, group, alias, etc)
  – ACE denies access for specified right -- deny
  – ACE grants access for some rights -- need full coverage
Access Checking with ACEs

- Example

```
Access Token (user 1)
- user 1 SID
- Group SIDs
- Privilege information
- Other access information

Jo
Group 1
Group 2

Process or Thread

Access Token (user 2)
- user 2 SID
- Group SIDs
- Privilege information
- Other access information

Carl
Group 2
Group 3

Process or Thread

Object

Security Descriptor
- Owner SID
- Group SID
- DACL

DACL
- ACE1 – Denied
  - Jo
  - Read, Write, Execute

ACE2 – Allowed
- Group 1
- Read, Write, Execute

ACE3 – Allowed
- Group 3
- Read, Write, Execute

Access is denied

Access is allowed
```
Windows vs. UNIX

• Is the additional expressive power of Windows worth it?
  – Who is supposed to use it?
Take Away

- Access Control is expressed in terms of Protection Systems
- Protection Systems consist of
  - Protection State representation (e.g., access matrix)
  - Enforcement Mechanisms (e.g., reference monitor)
- Protection States
  - Represented via Access Control Matrix
  - Although use Access Control Lists or Capabilities
- Challenge is to choose subjects for control
  - Subjects in UNIX and Windows
  - Subject transitions
- UNIX vs. Windows
  - Is the complexity of the Windows model worth it?