CSE543 - Introduction to Computer and Network Security

Module: Access Control

Professor Trent Jaeger
Access Control

• Method for restricting the operations that processes may perform on a computer system
  • aka Authorization
Access Control

• Why do you need access control?
Access Control

• Why do you need access control?
  • Protection
    • Prevent errors - oops, I overwrote your files
  • Security
    • Prevent unauthorized access under all conditions
Access Control

• What is needed for “security”?
  • Protect the process - limit others’ access to your resources
  • Confine the process - limit your access to others’ resources
Security Policies

• A security policy specifies the rules of security
  ‣ Some statement of secure procedure or configuration that parameterizes the operation of a system
  ‣ Example: Airport Policy
    • Take off your shoes
    • No bottles that could contain > 3 ozs
    • Empty bottles are OK?
    • You need to put your things through X-ray machine
    • Laptops by themselves, coat off
    • Go through the metal detector

• Goal: prevent on-airplane (metal) weapon, flammable liquid, dangerous objects … (successful?)
... when policy goes wrong

• Driving license test: take until you pass
  ‣ Mrs. Miriam Hargrave of Yorkshire, UK failed her driving test 39 times between 1962 and 1970!!!!
  ‣ … she had 212 driving lessons ….
  ‣ She finally got it on the 40th try.
  ‣ Some years later, she was quoted as saying, “sometimes I still have trouble turning right”

“A policy is a set of acceptable behaviors.”

- F. Schneider
Access Control Policy

• What is access control policy?
  ‣ Check whether a process is authorized to perform operations on an object

• Authorize
  ‣ Subject: Process
  ‣ Object: Resource that is security-sensitive
  ‣ Operations: Actions taken using that resource

• An object+operations is called a permission
  ‣ Sets of permissions for subjects and objects in a system is called an access control policy
Access Control Policy

• Access control policy determines what *operations* a particular *subject* can perform for a set of *objects*

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

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

- **Subjects** are the active entities that do things
  - E.g., you, Alice, students, Prof. 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
Access Policy Model

- A *protection system* answers authorization queries using a protection state (S), which can be modified by protection state methods (M)
  - Authorization query: Can *subject* perform requested *operation* on *object*? Y/N
- A *protection state* (S) relates subjects, objects, and operations to authorization query results
  - E.g., in mode bits, ACLs, … — the policy
- A *protection state methods* (M) can change the protection state (i.e., policy)
  - Add/remove rights for subjects to perform operations on objects — change the policy
The Access Matrix

- **An access matrix** is one way to represent a protection state.
  - Conceptual

- Columns are objects, subjects are rows.
  - To determine if $S_i$ has right to access object $O_j$, find the appropriate entry.
  - Often entries list the set of operations permitted for that subject-object pair

- The access matrix represents $O(|S|\times|O|)$ rules
The Access Matrix

- 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><strong>J</strong></td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td><strong>S_2</strong></td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td><strong>S_3</strong></td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
ACLs and Capabilities

• An access matrix is one way to represent a protection state.
  ‣ Conceptual

• Columns are objects
  ‣ Access control lists define the subjects that can access each object - and the operations

• Subjects are rows
  ‣ Capabilities define the objects that can be accessed by each subject - and the operations

• This is how access policies are stored
Access Control Problem

- Identify subjects, objects, and operations in each system
  - Minimize effort of parties that specify policies
  - Minimize likelihood of failures
    - Protection — failures due to benign errors
    - Security — failures due to malicious activities
    - Function — failures because programs don’t run
- Design an **Access Control Model**
  - **Subjects** - Per process or group a set of processes?
  - **Objects** - Per object or group a set of objects or permissions (object/ops)?
  - **Rules** - How to compose multiple requirements?
Access Control Problem

• You run three programs
  ‣ One from the system - **passwd**
  ‣ One application - **editor**
  ‣ One from the Internet - **email attachment**

• What access control policies should be assigned to each program? For protection? For security?

• How to make specifying access control policies easy?
Commodity OS Security

• UNIX and Windows Protection Systems
  ‣ How do they identify subjects/objects to express access control policies?
The UNIX FS access policy

- Really, this is a bit string ACL encoding an access matrix.
- E.g.,

```
rwx rwx rwx
```

- And a policy is encoded as “r”, “w”, “x” if enabled, and “-” if not, e.g,

```
rwxrw--x
```

- Says owner can read, write and execute, group can read and write, and world can execute only.
Caveats: UNIX Mode Bits

• Access is often not really this easy: you need to have certain rights to parent directories to access a file (execute, for example)
  ‣ The reasons for this are quite esoteric
• The preceding policy may appear to be contradictory
  ‣ A member of the group does not have execute rights, but members of the world do, so …
  ‣ A user appears to be both allowed and prohibited from executing access
  ‣ Not really: these policies are monotonic … the absence of a right does not mean they should not get access at all. If any of your identities have that right in any class (world, group, owner), you are authorized.
UNIX UIDs

• Processes and files are associated with user IDs (UIDs)
• File UID indicates its owner (who gets owner perms)
  ‣ Group UID also (who gets group perms)
• Process UID indicates the owner of the process
  ‣ Normal user
  ‣ System (root)
• Now, some special UIDs for some programs
  ‣ Also, a process may run under multiple Group UIDs

• How do we switch UIDs (e.g., run a privileged program)?
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
• **Downside**: User defines execution environment
  ‣ e.g., Environment variables, input arguments, open descriptors, etc.
• Service must protect itself or user can gain unauthorized access
  ‣ UNIX services often run as *root UID* -- many via setuid!
Windows Grows Up ...

- Windows 2000 marked the beginning of real access control for Windows systems ...
Tokens

• Like the UID/GID in a UNIX process
  ‣ User
  ‣ Group
  ‣ Aliases
  ‣ Privileges (predefined sets of permissions)

• May be specific to a domain
• Composed into global SID

• Subsequent processes inherit access tokens
  ‣ Different processes may have different rights
Access Control Entries

- DACL in the security descriptor of an object
  - e.g., like “rwx”
  - List of access control entries (ACEs)

ACE structure (proposed by Swift et al)

1. **Type** (grant or deny)
2. Flags
3. **Object Type**: global UID for type (limit ACEs checked)
4. **InheritedObjectType**: complex inheritance
5. **Access rights**: access mask
6. **Principal SID**: principal the ACE applies to
ACE Authorization

• The ACEs for a particular object are totally ordered.
• Start form the top and check each:
• Checking algorithm
  ‣ Authorizing for SIDs in token on set of rights

1. if ACE matches SID (user, group, alias, etc)
   a. ACE denies access for specified right -- deny
   b. ACE grants access for some rights -- need full coverage

2. If reach the bottom and not all granted, request denied
Access Checking with ACEs

- Example
Groups

- Groups are collections of identities who are assigned rights as a collective
- Important in that it allows permissions to be assigned in aggregates of users …

- This is really about “membership”
  - Group-Permission assignments are transient
Job Functions

• In an enterprise, we don’t really do anything as ourselves, we do things as some job function
  ‣ E.g., student, professor, doctor

• One could manage this as groups, right?
  ‣ We are assigned to groups all the time, and given similar rights as them, i.e., mailing lists
Roles

- A role is a collection of privileges/permissions associated with some function or affiliation.
- NIST studied the way permissions are assigned and used in the real world, and this is it …

- **Important**: the permission-role membership is static, the user-role membership is transient.
Role Based Access Control

• Most formulations are of the type
  ‣ \textbf{U}: users -- these are the subjects in the system
  ‣ \textbf{R}: roles -- these are the different roles users may assume
  ‣ \textbf{P}: permissions --- these are the rights which can be assumed

• There is a \textit{many-to-many relation} between:
  ‣ Users and roles
  ‣ Roles and permissions

• Relations define the role-based access control policy
Take Away

• **Goal**: Define protection states to restrict the operations that each process may perform
  ‣ For protection from bugs and security from adversaries
  ‣ Operating systems do that by
    • Associating processes with IDs (subjects)
    • Authorizing objects and operations (permissions)

• **Approach**: Protection system
  ‣ **Protection state**: Relates subjects to authorized permissions
  ‣ **Methods** for modifying the protection state

• UNIX and Windows implement protection systems
  ‣ Have different notions of subjects and permissions
  ‣ Trade-off complexity and expressive power

• Compared with role-based access control models