Course Policies and Logistics

- Midterm November 3!
- HW due 10/13!
- Next HW 4 posted!!
- Sample Exams on Web
- Read Chapter 2 and 15 of text
Access Control

- Any system consists of objects and subjects (active objects such as processes, users etc.) which *access* these objects.

- The security policy of a system defines
  - What a subject is allowed to do
  - What may be done with an object

- In other words – *Access Control*

- Two issues –
  - How do you specify access control to be performed?
  - How do you implement access control policy?
Reference Monitor Concept

- Above is just a conceptual model
- An actual system may not include an explicit reference monitor
- But we need to define functionality of the reference monitor and design mechanisms for its implementation.
Protection System

- The state of a system. For computer systems – values of all memory locations including secondary storage, registers, components etc.
- Protection state – subset that deals with protection (access control).
- Protection system – system that controls the state transitions that affect the protection states.
**Access Control Matrix (ACM) Model**

- Simplest framework for describing protection system.
- A Matrix $A$ with $a[s, o] \subseteq R$, where $s \in S$ - the set of subjects, $o \in O$ the set of objects and $R$ is the set of rights the subject has over the object.
- The triple $(S, O, A)$ represents the protection state of the system.

<table>
<thead>
<tr>
<th></th>
<th>File 1</th>
<th>File 2</th>
<th>Process 1</th>
<th>Process 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process 1</td>
<td>r,w,o</td>
<td>r</td>
<td>r,w,x,o</td>
<td>w</td>
</tr>
<tr>
<td>Process 2</td>
<td>a</td>
<td>r,o</td>
<td>r</td>
<td>r,w,x,o</td>
</tr>
</tbody>
</table>
Meaning of Rights

- The set of objects, subjects and rights vary from system to system

<table>
<thead>
<tr>
<th>Hostname</th>
<th>Telegraph</th>
<th>Nob</th>
<th>Toadfax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Telegraph</td>
<td>Own</td>
<td>ftp</td>
<td>ftp</td>
</tr>
<tr>
<td>Nob</td>
<td>ftp,nfs,mail,own</td>
<td>ftp,nfs,mail</td>
<td></td>
</tr>
<tr>
<td>Toadfax</td>
<td>ftp,mail</td>
<td></td>
<td>ftp,nfs, own</td>
</tr>
</tbody>
</table>

- Interpretation of rights varies from system to system. For example, a process “reading” a process.
Generality

- Model is general. For example, programming language accesses:

<table>
<thead>
<tr>
<th></th>
<th>counter</th>
<th>Inc_ctr</th>
<th>Dec_ctr</th>
<th>manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inc_ctr</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dec_ctr</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>manager</td>
<td>call</td>
<td>call</td>
<td>call</td>
<td>call</td>
</tr>
</tbody>
</table>

- In fact, entries in ACM can even be functions that determine rights based on state, history, time of day etc. For example, a locking function that ensures no access while write.

- Your text has even more elaborate examples.
Various access control mechanisms have been proposed:

- Access Control Matrix
- Access Control List
- Capability based access control
- Lock and Key based access control.
Discretionary and Mandatory Controls

- A *discretionary access control* mechanism is defined as those procedures and mechanisms that enforce specified mediation at the discretion of individual users.

- A *mandatory access control* mechanism is defined as those procedures and mechanisms that enforce specified mediation not at the discretion of individual users but rather at the discretion of a centralized system administration facility.
Access Control Matrix (ACM)

An **Access Control Matrix** is a table in which

- each row represents a subject,
- each column represents an object, and
- each entry is the set of access rights for that subject to that object.
ACM - Example

- Consider system with two files and two processes. Set of rights is - r,w,x,a,o (read, write, execute, append, own).

<table>
<thead>
<tr>
<th></th>
<th>File 1</th>
<th>File 2</th>
<th>Process 1</th>
<th>Process 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Process 1</strong></td>
<td>r,w,o</td>
<td>r</td>
<td>r,w,x,o</td>
<td>w</td>
</tr>
<tr>
<td><strong>Process 2</strong></td>
<td>a</td>
<td>r,o</td>
<td>r</td>
<td>r,w,x,o</td>
</tr>
</tbody>
</table>

- Can get very large and hence inefficient in general purpose scenarios - seldom used.
Access Control Lists

- Instead of using ACM, **Access Control List (ACL)**. Essentially store each column of ACM with the object it represents.

- **Definition:** Let $S$ be set of subjects and $R$ the set of rights of a system. An *access control list* $l$ is a set of pairs $l = \{(s, r) : s \in S, r \subseteq R\}$. Let $acl$ be a function that determines the access control list associated with a particular object $o$. $acl(o) = \{(s_i, r_i) : 1 \leq i \leq n\}$ means that subject $s_i$ may access $o$ using any right in $r_i$.
ACL - Example

For ACM shown earlier, corresponding ACL’s are:

\[
\begin{align*}
\text{acl(file 1)} &= \{ (\text{proc.1}, \{r,w,o\}) \text{ (proc. 2, } \{a\}) \}\} \\
\text{acl(file 2)} &= \{ (\text{proc.1}, \{r\}) \text{ (proc. 2, } \{r,o\}) \}\} \\
\text{acl(proc.1)} &= \{ (\text{proc.1}, \{r,w,x,o\}) \text{ (proc.2, } \{r\}) \}\} \\
\text{acl(proc.2)} &= \{ (\text{proc.1}, \{rw\}) \text{ (proc.2, } \{r,w,x,o\}) \}\}
\end{align*}
\]
Abbreviated ACL’s

- Although same amount of storage, it is now distributed.
- To further reduce storage, one can abbreviate ACL’s as in UNIX.
- One can also assign default access to groups of subjects as well as specific rights to individual subjects.
  - Two ways of doing this: 1) *What is not prohibited is permitted* 2) *What is not permitted is prohibited.* Latter always better!!
  - Example: Unix *hosts.allow* and *hosts.deny* files.
Example - File Protection in Unix

- UNIX - allow *read, write, execute, delete* to each of the individual groups - *owner, group, world.*
  - Difficult for users in different groups to share files, since each user may belong to exactly one group.

- NT - Combination of groups, Access Control Lists and capability based control.

- The Unix *set userid (suid)* scheme allows another user to *temporarily* acquire the protection level of a files owner.
  - While executing the program to change their own password, Unix users actually acquire temporary modify access to the systems password file, but in a controlled way using suid.
Unix special users

- Special user with extra privileges – *root.*
  - UID is 0.
  - Can do (almost) anything!!
  - Holy grail of hackers!

- Other special users
  - *daemon* or *sys* – handles some network services
  - *ftp* – used for anonymous FTP access.
  - *uucp* – manages UUCP system.
  - *guest* – used for site visitors.
  - *lp* - used by printer system
  - Etc.
Unix Groups

- Every user belongs to one or more groups.
- The GID of primary group the user belongs to is stored in passwd file.
- Groups useful for access control features.
- /etc/groups contains a list of all groups in the system along with GID’s.
- Some special groups –
  - wheel - group of administrators
  - uucp, lp, etc. - groups corresponding to special users.
Unix file access control

- Each file entry in a directory is a pointer to a data structure called **inode**.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>Type of file and access rights</td>
</tr>
<tr>
<td>uid</td>
<td>User who owns the file</td>
</tr>
<tr>
<td>gid</td>
<td>Group which owns the file</td>
</tr>
<tr>
<td>atime</td>
<td>Access time</td>
</tr>
<tr>
<td>mtime</td>
<td>Modification time</td>
</tr>
<tr>
<td>itime</td>
<td>Inode alteration</td>
</tr>
<tr>
<td>Block count</td>
<td>Size of file</td>
</tr>
<tr>
<td></td>
<td>Pointer to physical location</td>
</tr>
</tbody>
</table>
Unix file permission bits

- Two examples of file permissions obtained by `ls -l` command
  - `-rw-------`
  - `drwxr-xr-x`
- First character indicates type of file
  - `-` plain file
  - `d` directory
  - `c` character device (tty or printer)
  - `b` block device
  - `l` symbolic link
  - Etc.
File permission bits (contd.)

- Next nine characters taken in groups of three indicate who can do what with the file
  - R – Permission to read
  - W – Permission to write
  - X – Permission to execute

- The three classes of permission correspond respectively to
  - Owner
  - Group
  - Other
File permission bits - special cases

- File permission bits do not apply to symbolic links.
- If you have x access but no r access you can execute the program without reading it (not on linux).
- Execute permission in a directory means you can change to the directory. Secret Files!
- File permission bits also commonly specified in octal notation. 0777 mean -rwxrwxrwx, 0600 means -r_x-------, etc.
Umask and default permissions

- **umask** (User file creation mode mask) is a four digit octal number used to determine file permissions for newly created files.
- It defines permission you do *not* want to be given (the bit-wise complement of the permission you want a file to have by default).
- 0666 – default mode means 0222 umask.
- 0077 umask means … 0022 means …
- Set up at log in time in environment variables.
The suid bit

- Sometimes unprivileged users must perform tasks that are privileged.
  - Change password thereby modify /etc/passwd.
- UNIX allows certain programs to change UID to their owner when executed.
  - SUID programs – change UID to owner.
  - SGID programs – change GID to owners group.
- ls -l command indicates if SUID or SGID
  - -rwsr-xr-x indicates SUID
  - -rwxr-sr-x indicates SGID
Limitations of UNIX file permission system

- Abbreviated ACL’s in general and UNIX in particular may not be flexible enough for many circumstances.

- Consider the following example:
  - 5 users, Anne, Beth, Caroline, Della and Elizabeth.
  - Anne wants Beth to read her file and nothing else.
  - She wants Caroline to write
  - Della to only read and write
  - Elizabeth to only execute

Above not possible with Unix file permission bits!!
Augmenting abbreviated ACL’s

- AIX uses extended permissions to augment base permissions.

  attributes:
  base permissions: owner (bishop): rw-
      group (sys): r—
      others: ---

  extended permissions enabled
  specify   rw-     u:heberlei
  permit    -w-     u:nelson, g=sys
  permit    rw-     u:levitt
  deny      -w-     u:heberlei, g=faculty
NT Access Control Lists (ACL)

- Each object contains a security descriptor, which has
  - Security Identifier of the person who owns the object,
  - The regular ACL for access permissions,
  - The system ACL (SACL) which is used for auditing,
  - A group security identifier.

- ACL may be composed of Access Control Entries (ACE) which are composed of
  - Basic permissions (six individual permissions),
  - Standard permissions which are combinations derived from the basic permissions.
Basic Permissions

- Read (R)
- Write (W)
- Execute (X)
- Delete (D)
- Change Access Permissions (P)
- Take Ownership (O)
### NTFS ACL Standard Permissions

<table>
<thead>
<tr>
<th>Permission Name</th>
<th>Directory Permission</th>
<th>File Permission</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Access</td>
<td>None</td>
<td>None</td>
<td>No access to files and directories.</td>
</tr>
<tr>
<td>List</td>
<td>RX</td>
<td>Not Specified</td>
<td>List Directory Contents Change to subdirectories No access to files unless granted explicitly</td>
</tr>
<tr>
<td>Read</td>
<td>RX</td>
<td>RX</td>
<td>List Directory Contents Change to subdirectories Read data from files Execute programs</td>
</tr>
<tr>
<td>Add</td>
<td>WX</td>
<td>Not Specified</td>
<td>Create subdirectories Create files No access to existing files unless granted explicitly</td>
</tr>
</tbody>
</table>
### NTFS ACL Standard Permissions

<table>
<thead>
<tr>
<th>Permission Name</th>
<th>Directory Permission</th>
<th>File Permission</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add &amp; Read</td>
<td>RWX</td>
<td>RX</td>
<td>List Directory Contents, Change to subdirectories. Create subdirectories, Read data from files. Execute programs.</td>
</tr>
<tr>
<td>Full Control</td>
<td>All</td>
<td>All</td>
<td>All directory permissions. All file permissions, Change permissions. Take ownership.</td>
</tr>
</tbody>
</table>
NT Domains

- A domain is a set of computers with a central security authority, the primary domain controller (PDC), that grants access to a domain.
- PDC and the BDC (Backup) must be Windows NT.
- A domain can be set up to
  - Ease viewing and access to resources,
  - Share a common user account database and security policy,
  - Enforce a common security stance across physical, divisional, or corporate boundaries.
  - Elimination of the need for every machine to provide its own authentication service.
- Users authenticated to the domain, can gain access to resources, such as printing, file sharing or applications, across all of the servers.
Issues to consider while designing an ACL based system

- Which subject can modify an object's ACL?
- Does ACL apply to privileged user (root), if any?
- Does ACL support groups or wildcards?
- How are contradictory permissions handled?
- If default permissions allowed, do ACL's modify it? Or is default used only when subject not explicitly mentioned in ACL?

Read text for examples of how above are handled in different systems.
Revoking Rights

- Revoking involves deletion of subject’s rights from object’s ACL.
- Typically owner of object has ability to provide or delete rights.
- If ownership does not control the giving of rights, then revocation is more complex.
Capability based access control.

- Conceptually, *capability* is row of ACM i.e. list of rights for a subject.
- **Definition**: Let $O$ be set of objects, and $R$ the set of rights of a system. A *capability list* $c$ is a set of pairs $c = \{(o, r) : o \in O, r \subseteq R\}$. Let $\text{cap}$ be function that determines capability list $c$ associated with subject $s$. Then $\text{cap}(s) = \{(o_i, r_i) : 1 \leq i \leq n\}$ is that subject $s$ may access $o_i$ using any right in $r_i$. 
Example

For the ACM we saw earlier, capability lists are:

\[
\text{cap(proc. 1)} = \{(\text{file1}, \{r,w,o\}), (\text{file2}, \{r\}), (\text{proc 1}, \{r,w,x,o\}), (\text{proc 2}, \{w\})\}.
\]

\[
\text{cap(proc. 1)} = \{(\text{file1}, \{r,w,o\}), (\text{file2}, \{r\}), (\text{proc 1}, \{r\}), (\text{proc 2}, \{r,w,x,o\})\}.
\]
Capability Based Access Control – Implementation.

- A capability is an unforgeable “token” giving the possessor certain rights to an object.
- Analogous to movie ticket or ID card.
- Could be made transferable with appropriate entry in ticket.
- Used by Kerberos (WIN 2K).
- To make sure capability cannot be forged:
  - Maintained by OS.
  - Stored in a region not accessible to users.
  - Use cryptography (example, digital signatures) to prevent forgery.
Example - Amoeba

- On creation of an object, capability corresponding to object is returned to owner.
- To later use object, owner presents capability.
- Capability encoded name of object (24 bits), the server that created it (48 bits), rights (8 bits, initially all set), and 48 bit random “check” field.
- Random number stored in table of server that created object. When capability presented, number checked.
- Attacker who does not know random number cannot forge capability.
- If capability disclosed, system becomes vulnerable.
Copying Capability

- Copying capability means giving rights. How do you allow copying?
- Amoeba: X wants Y to read object O which X owns. X asks server for copy of capability to access O, but restricted to reading.
- Server sets only read bit in rights field, XOR's with random check and result is hashed. Output of hash is used as random check for this new capability.
- On receiving capability with at least one bit set to zero, server takes rights field and XOR’s with original random check and hashes. If hash matches that presented in the capability, access is allowed.
- Different capability cannot be forged.
Revoking rights in capability based system

- Check each process and delete capability? Too inefficient. How to do this efficiently?
- One method: Use *indirection*. Capability does not name object but contains a pointer to object in global table. To revoke entry, just invalidate entry in global table.
- Amoeba: Change random check and issue new capability. This validates all existing capabilities.
Comparison of ACL and capability

- Two questions arise in access control systems:
  - Given a subject, what objects can access it and how?
  - Given an object, what subjects can access it and how?
- Former easier with capabilities and latter with ACL. Why?
- Latter more often asked, hence ACL’s used more often.
- With more distributed processing and agent based systems, perhaps the former question will be asked more in the future.
Example: NT Access Tokens and Security Identifiers (SID)

- Created by the Local Security Authority after SAM validation, as part of a successful logon process.
- Stays with that particular user's session for as long as they stay logged on.
- Whenever a user initiates a process during the course of the session, a copy of the token is attached to that process.
- Once the user logs off, the token is destroyed and will never be used again.
NT Tokens

- Each token contains the following information:
  - User’s Security Identifier (SID)
  - Group Security Identifiers
  - User privileges
  - Owner (SID assigned to any objects created during the session)
  - Primary Group SID
  - Default ACL (assigned to any object created by the user)
NT User Rights

- 27 specific 'user rights' that can be assigned (or restricted) to users or groups in NT. These include:
  - the ability to access a computer from the network,
  - to change the system time,
  - to log onto the system locally,
  - the ability to take ownership of objects, and even
  - to shut down the system.
  - password restrictions,
  - logon times,
  - remote access capabilities,
  - group memberships etc.
NT Built-in Groups

- Built-in users and groups have pre-defined rights and permissions
  - Global built-in groups – Domain Admins, Domain Users, Domain Guests
  - Local built-in groups – Administrators, Backup Operators, Users, Guests, Etc.

- Special built-in groups exist that can be used to define appropriate access permissions
  - Everyone
  - Interactive
  - Network
  - Creator Owner
  - System
  - Etc.
Access control with Locks and Keys

- Combines features of ACL’s and capabilities.
- A piece of information (*lock*) associated with the object.
- Another piece of information (*key*) associated with subjects authorized to access the object.
- Example implementation: Encrypt object and provide key to subject.
- To have n subjects needed to access object, encrypt with n keys and give one to each subject. All keys needed to access.
Locks and Keys in IBM 370

- Each process assigned *access key* and each page a *storage key* and *fetch bit*.
  - If fetch bit is cleared, only read access allowed.
  - Process with access key 0 can write any page with fetch bit set.
  - If storage key matches access key of process then process allowed to write to page.
  - If no match and access key not 0, then no access allowed.
Ring based access control

- Used by MULTICS. Generalizes notion of user and supervisor modes.
- Segments in memory and disk are of two types - data and procedure.
- Segments have r, w, x, and a rights associated.
- In addition, notion of protection ring associated with segments. Ring is a number r, 0 <= r <= 63.
- A process is said to reside in ring r if it is currently executing a segment in ring r.
- Kernel resides in ring 0.
- Higher the ring number, lower the privileges.
- Crossing ring boundaries causes traps to the kernel which invokes ring gatekeeper.
Access brackets for data segments

- Each data segment as *access bracket*, which is a ring \((a_1, a_2)\) with \(a_1 \leq a_2\).

- Assume data segments permission allow desired access to a process in ring \(r\). The access bracket add additional constraints:
  - If \(r \leq a_1\); access permitted;
  - If \(a_1 < r \leq a_2\), \(r\) (read) and \(x\) permitted, \(w\) and \(a\) denied.
  - If \(a_2 < r\); all access denied.
Call brackets for procedures.

- In addition to access brackets, procedure segments may also have call bracket \((c_1, c_2)\).

- This leads to protection \((a_1, a_2, a_3)\) where \((a_1, a_2)\) - access bracket & \((a_2, a_3)\) - call bracket.

Access rules are:

- If \(r \leq a_1\) access permitted; but ring crossing fault occurs.
- If \(a_1 \leq r \leq a_2\) all access permitted and no fault occurs.
- If \(a_2 < r \leq a_3\) access permitted if made through valid gate.
- If \(a_3 < r\) all access denied.
Why call brackets?

- Consider a service routine in ring a.
  - We need to allow user set A to invoke this routine.
  - We allow user set B to invoke only in specific ways
  - We do not allow user set C to invoke at all.

- Access brackets can take care of A and C.

- To handle set B we need call brackets so that we can regulate their access.