Chapter 14 ACCESS CONTROL MECHANISMS

- There are several problems with a straightforward implementation of the access control matrix.
  - On a typical system, the number of subjects and objects will be sufficiently large, and the size of the matrix will require a significant amount of storage.
  - Most entries in the matrix will either be blank (indicating no access) or the same (implementations often use a default setting).
  - The creation and deletion of subjects and objects will require the matrix to manage its storage carefully.
- Several optimizations enable systems to use more convenient, and in some cases simpler, versions of the access control matrix.
- The following remedies reduce the complexity of the code:
  - Access control lists
  - Capabilities
  - Locks and keys
  - Ring-based access control

Chapter 14 ACCESS CONTROL MECHANISMS

14.1 Access Control Lists

- A variant of the access control matrix is to store each column with the object it represents.
- Each object is associated with a set of pairs {a subject, a set of rights}.

• S: the set of subjects, R: the set of rights

An access control list (ACL) is a set of pairs $l = \{(s,r): s \in S, r \subseteq R\}$.

acl: a function that determines the ACL associated with a particular object o.

$acl(o) = \{(s_i,r_i): 1 \leq i \leq n\}$ is interpreted as subject s_i may access object o using any right in r_i.

• Consider the following access control matrix.

Each subject and object has an associated ACL. →

	file 1	file 2	process 1	process 2
process 1	read, write, own	read	read, write, execute, own	write
process 2	append	read, own	read	read, write, execute, own

acl(file 1) = {(process 1, {read, write, own}), (process 2, {append})}
acl(file 2) = {(process 1, {read}), (process 2, {read,own})}
acl(process 1) = {(process 1, {read, write, execute, own}), (process 2, {read})}
acl(process 2) = {(process 1, {write}), (process 2, {read, write, execute, own})}

Chapter 14 ACCESS CONTROL MECHANISMS

Another example of ACLs

	***file*1**	***file*2**	***file*3**
Andy	**rx**	**r**	**rwo**
Betty	**rwxo**	**r**	
Charlie	**rx**	**rwo**	**w**

ACLs:

file1: { (Andy, rx) (Betty, rwxo) (Charlie, rx) }
file2: { (Andy, r) (Betty, r) (Charlie, rwo) }
file3: { (Andy, rwo) (Charlie, w) }

Chapter 14 ACCESS CONTROL MECHANISMS

- <u>Default</u> Permissions
  - If a subject is not named in the ACL, its has no rights over the associated object.
  - On a system with many subjects, the ACL may be very large.
  - If many subjects have the same right over the file, one may use groups or wildcards in ACL.
- <u>EXAMPLE</u>: UNICOS 7.0
  - ACLs have entries of the form (*user*, *group*, *rights*).
  - If the user is the named *group*, he/she has those *rights* over the object.
  - For example, the triplet (*holly*, *maceranch*, *r*) gives user *holly* read(*r*) access over the object only when *holly* has *maceranch* as her group.
  - If either *user* or *group* is specified as "*," that character is taken to match all users or all groups.
  - (*holly*, *, *r*) gives *holly* read permission over the object regardless of the group she is in.
  - (*, *maceranch*, *r*) gives any user read permission over the object when that user is in the group *maceranch*.

Chapter 14 ACCESS CONTROL MECHANISMS

<u>Abbreviations of ACLs</u>

- Some systems abbreviate ACLs.
- UNIX systems divide the set of users into 3 classes:
  - The owner of the file
  - The group owner of the file
  - All other users

  Each class has a separate set of rights.

- **<u>EXAMPLE</u>: UNIX systems**
  - Provide read (*r*), write (*w*), and execute (*x*) rights.
  - Permissions are represented as triplets.
    - Owner rights
    - Group rights
    - Other rights
  - Within each triplet, we have (*r*, *w*, *x*) if the right is allowed.
  - Assume user *bishop* is in group *vulner*. He creates a file.
    - Initially, he requests that
      - He be able to read and write to the file
      - Members of the group be allowed to read from the file
      - No one else has access to the file
    - The permissions would be: *rw* for owner, *r* for group, none for others.
    - In the triplet, the rights are rw_r_ _ _ _ _.

Chapter 14 ACCESS CONTROL MECHANISMS

- Abbreviations of ACLs (e.g., UNIX OS) suffer from a <u>loss of granularity</u>.
  - Suppose a UNIX system has 5 users: Anne, Beth, Caroline, Della, and Liz.
  - Anne wants to allow Beth to read her file.
  - Anne wants to allow Caroline to write to it.
  - Anne wants to allow Della to read and write to it.
  - Anne wants to allow Liz to execute it.

Because there are 3 sets of permissions and 5 desired arrangements, 3 triplets are insufficient to allow all desired modes of access. So, Anne must compromise by giving someone more rights or give someone fewer rights.

- Many systems <u>augment</u> abbreviations of ACLs with full-blown ACLs.
  - This scheme uses the abbreviations of ACLs as the default permission controls.
  - The explicit ACL overrides the defaults as needed.
- <u>EXAMPLE</u>: IBM's version of the UNIX OS (AIX)
  - Uses an ACL ("extended permissions") to augment the traditional UNIX abbreviations of ACL ("base permissions").
  - Unlike traditional ACLs, the AIX ACL allows one to specify permissions to be added or deleted from the user's set.
  - Like UNICOS, AIX bases matches on group and user identity.
  - The specific algorithm ("base permissions" are the UNIX abbreviations of ACLs, and "extended permissions" are unabbreviated ACL entries) is as follows:

Chapter 14 ACCESS CONTROL MECHANISMS

1. Determine what set *S* of permissions the user has from the base permissions.
2. If extended permissions are disabled, stop. The set *S* is the user's set of permissions.
3. Get the next entry in the extended permissions. If there are no more, stop. The set *S* is the user's set of permissions.
4. If the entry has the same user and group as the process requesting access, determine if the entry denies access. If so, stop. Access is denied.
5. Modify *S* as dictated by the permissions in the entry.
6. Go to Step 3.

- **EXAMPLE**:
  - Consider the following representation of an AIX system's access control permissions for the file *xyzzy*.

```
attributes:
base permissions
            owner(bishop):  rw-
            group(sys):     r--
            others:         ---
extended permissions enabled
            specify         rw-     u:holly
            permit          -w-     u:heidi, g=sys
            permit          rw-     u:matt
            deny            -w-     u:holly, g=faculty
```

In the extended permissions:
- the first field determines what the line means.
  "specify": overrides base permissions
  "permit": adds rights
  "deny": deletes rights
- the second field states the rights involved
- the third field defines the user and group.

  - holly can read *xyzzy* because the 1st and 4th lines in the extended permissions section override the base permission denial of access to others.
  - If holly is working in the faculty group, she cannot write to *xyzzy* but can read it.

Chapter 14 ACCESS CONTROL MECHANISMS

- Creation and Maintenance of ACLs
  - Specific implementations of ACLs differ in details.
  - Some of the issues are as follows:
    - Which subjects can modify an object's ACL?
    - If there is a privileged user (e.g., *root* in the UNIX system), do the ACLs apply to that user?
    - Does the ACL support groups or wildcards?
    - How are contradictory access control permissions handled? For example, if one entry grants read privileges only and another grants write privileges only, which right does the subject have over the object?
    - If a default setting is allowed, do the ACL permissions modify it? Is the default used only when the subject is not explicitly mentioned in the ACL?
  - Which subjects can modify an objects ACL?
    - When an ACL is created, rights are instantiated.
    - Possessors of the *own* right can modify the ACL.
    - When an object is created, its creator is initially given all rights (e.g., *own*) over the new object.
    - EXAMPLE: The relational database System R
      - Contains set of *n*-tuples.
      - Each element of each *n*-tuple has attributes.
      - The *n*-tuples are stored as tables.
      - In every table, the records are the rows and the attributes are the columns.
      - Each table defines an relation.
      - The rights for manipulating a table include *read*, *update*, *insert*, *delete*, and *drop*.
      - Each right has a modifier (the *grant* option). If it is set, the possessor is allowed to give the right to another.

Chapter 14 ACCESS CONTROL MECHANISMS

- Do the ACLs apply to a privileged user?
  - Many systems have users with extra privileges.
  - A known example is the *root* superuser on UNIX systems.
  - EXAMPLE: Solaris UNIX system
    - Uses both the abbreviations of ACLs standard to UNIX systems and a full-blown ACL.
    - The abbreviations of ACLs are ignored when *root* is the subject.
    - However, the full ACLs apply even to the *root*.
- Does the ACL support groups and wildcards?
  - In its classic form, ACLs do not support groups or wildcards.
  - In practice, systems support one or both to limit the size of the ACL and to make manipulation of the lists easier.
  - EXAMPLE 1:
    - In the previous AIX example, the extended permissions were:

base permissions			
	owner(bishop):	rw-	
	group(sys):	r--	
	others:	---	
extended permissions enabled			
	specify	rw-	u:holly
	permit	-w-	u:heidi, g=sys
	permit	rw-	u:matt
	deny	-w-	u:holly, g=faculty

    - Initially, the group *sys* had read permission only on the file.
    - The 2nd line adds write permission for processes with UID *heidi* and GID *sys*.
    - The 1st line gives processes with UID *holly* read and write access except when the GID of the process is *faculty*.

Chapter 14 ACCESS CONTROL MECHANISMS

- **EXAMPLE 2:**
  - The UNICOS OS provides ACLs similar to those of AIX, but also allows wildcards.
  - For example, "holly : maceranch : r" means that a process with UID *holly* and GID *maceranch* can read the object with which the ACL is associated.
  - The ACL entry "holly : * : r" means that a process with UID *holly* can access the object regardless of the group that the process is in.
  - The entry "* : maceranch : r" means that any process with GID *maceranch* can read the object.

– **Conflicts**

- **A <u>conflict</u> arises when 2 access control list entries in the same ACL give different permissions to the subject.**
- **The system can**
  - Allow access if any entry would give access
  - Deny access if any entry would deny access
  - Apply the first entry that matches the subject
- **Example 1:**
  - If any entry in an AIX ACL denies access, the subject is denied access regardless of the location of that entry.
  - Otherwise, if any entry has granted access, the subject is granted access.
  - This is an example of denial taking precedence.
- **Example 2:**
  - Cisco routers apply the first access control list entry that matches the incoming packet.
  - If none applies, the incoming packet is discarded.
  - This is an example of the second approach, with a default rule of deny.

Chapter 14 ACCESS CONTROL MECHANISMS

- ACLs and Default Permissions
  - On many UNIX systems, when ACLs and abbreviations of access control lists or default access rights coexist, there are 2 ways to determine access rights:
    - Apply the appropriate ACL entry, if one exists. Apply the default permissions or abbreviations of access control lists, otherwise.
    - Augment the default permissions or abbreviations of access control lists with those in the appropriate ACL entry.
  - EXAMPLE 1:
    - The AIX extended permissions fall into the second category because they modify the base permissions.
  - EXAMPLE 2:
    - If a packet entering a Cisco router is destined for a host on a network behind the router, but the router has no access list entry that allows the packet to be forwarded, the packet is discarded.
    - This is an example of the first method because the default permission is deny.

- **Revocation of Rights**
  - Revocation requires that the subject's rights be deleted from the object's ACL.
  - The owner deletes the subject's entry from the object's ACL.
  - If only specific rights are to be deleted, they are removed from the relevant subject's entry in the ACL.
  - If ownership does not control the giving of rights, revocation is more complex!

Chapter 14 ACCESS CONTROL MECHANISMS

- **EXAMPLE: System R**
  - **Suppose Anna has given Peter *update* rights over a relation *T* but now wishes to revoke them.**
  - **System R restores the protection state to what it was before the right was given.**
  - **If Peter gave Mary *update* rights after Anna's revocation of Peter's *update* rights, Mary's *update* rights should be revoked.**
  - **To implement this, System R defines a relation called *Sysauth*.**
  - **The attributes of this relation are (*User*, *Table*, *Grantor*, *Read*, *Insert*, *Delete*, *Drop*, *Update*).**
  - **The values of the attributes corresponding to the rights are timestamps indicating when the right was given (with one exception: *Update*).**
    - **Suppose Anna gave Peter *read* rights over the relation Reports at time 10, and Peter gave them to Mary at time 20, the table would be:**

User	Table	Grantor	Read
Peter	Reports	Anna	10
Mary	Reports	Peter	20

    - **If Anna revokes Peter's *read* rights, and Mary obtained her *read* rights after Anna gave them to Peter, Mary's *read* rights would also be revoked.**
    - **However, suppose Michelle has also given Mary *read* rights over Reports.**
    - **If the last row in the above table is deleted, we would have (Mary can still read Reports):**

User	Table	Grantor	Read
Peter	Reports	Anna	10
Mary	Reports	Michelle	5

    - **The *update* right has a value of *All*, *Some*, or *None*.**
    - **If the value is *Some*, a 2nd relation called *Syscolauth* records the columns that the subject can update. This table also records times, and revocation proceeds as for the other columns.**

Chapter 14 ACCESS CONTROL MECHANISMS

14.2 Capabilities

- Conceptually, a capability is like the row of an access control matrix.
- Each subject is associated with a set of pairs {an object, a set of rights}.

- *O*: the set of objects, *R*: the set of rights
  A <u>capability list *c*</u> (C-List) is a set of pairs $c=\{(o,r):\ o \in O,\ r \subseteq R\}$.
  cap: a function that determines the capability list associated with a particular subject *s*.
  $cap(s) = \{(o_i,r_i): 1 \leq i \leq n\}$ is interpreted as subject *s* may access object o_i using any right in r_i.
- Consider the following access control matrix.

Each subject has an associated C-List.

	file 1	file 2	process 1	process 2
process 1	read, write, own	read	read, write, execute, own	write
process 2	append	read, own	read	read, write, execute, own

cap(process 1) = {(file 1, {read, write, own}), (file 2, {read})},
(process 1, {read, write, execute, own}), (process 2, {write})}
cap(process 2) = {(file 1, {append}), (file 2, {read, own}),
(process 1, {read}), (process 2, {read, write, execute, own})}

Chapter 14 ACCESS CONTROL MECHANISMS

- Capabilities encapsulate object identity.
- When a process presents a capability on behalf of a user, the OS examines the capability to determine both the object and the access to which the process is entitled.
- The location of the object in memory is encapsulated in the capability.
- Without a capability, the process cannot name the object in a way that will give it the desired access.
- EXAMPLE:
  - To open a UNIX file, a process gives the file name to the kernel.
  - The kernel obtains the file's inode number by resolving the name through the file hierarchy.
  - Once the inode is obtained, the system determines if the requested access should be granted using the access control permissions.
  - If the access is granted, the OS returns a capability called a *file descriptor*.
  - The capability is tightly bound to the file object. Even if the file is deleted, and a new file with the same name is created, the file descriptor still refers to the previous file.

Chapter 14 ACCESS CONTROL MECHANISMS

Another example of C-Lists

	*file*1	*file*2	*file*3
Andy	rx	r	rwo
Betty	rwxo	r	
Charlie	rx	rwo	w

C-Lists:

Andy: { (file1, rx) (file2, r) (file3, rwo) }

Betty: { (file1, rwxo) (file2, r) }

Charlie: { (file1, rx) (file2, rwo) (file3, w) }

Chapter 14 ACCESS CONTROL MECHANISMS

- <u>Implementation of Capabilities</u>
  - 3 mechanisms are used to protect capabilities:
    - Tags
    - Protected memory
    - Cryptography
  - A <u>tagged</u> architecture has a set of bits associated with each hardware word.
    - EXAMPLE:
      - The B5700 used a tagged architecture.
      - The tag field consisted of 3 bits to indicate how the architecture was to treat a word.
  - Protected memory
    - All capabilities are stored in a page (segment) that the process can read but not alter.
    - EXAMPLE:
      - The CAP system did not allow processes to modify the segment in which instructions lay.
      - The system also stored capabilities in this segment.
  - Cryptography
    - The goal of tags and memory protection is to prevent the capabilities from being altered.
    - Cryptographic checksums are another mechanism for checking the integrity of information.
    - Each capability has a cryptographic checksum associated with it.
      - The checksum is enciphered using a cryptosystem whose key is known to the OS.
      - When the process presents a capability to the OS, the system recomputes the cryptographic checksum.
      - The system then enciphers or deciphers the checksum. In either case, if there is a match, the capability is unaltered.

Chapter 14 ACCESS CONTROL MECHANISMS

- **<u>Copying Capabilities</u>**
  - **The ability to copy capabilities implies the ability to give rights.**
  - **To prevent processes from indiscriminately giving away rights, a <u>copy flag</u> is associated with capabilities.**
  - **A process cannot copy a capability to another process unless the copy flag is set.**
  - **If the process does copy the capability, the copy flag may be turned off.**
  - **EXAMPLE:**
    - **Amoeba uses an interesting cryptographic scheme.**
    - **Suppose user *matt* wishes to allow user *holly* to read an object he owns.**
    - **He passes his capability for that object to the server and requests a restricted capability for reading.**
    - **The server creates a new capability for the object with only the read right turned on.**
    - **The rights field is all 0's except for the read bit.**
    - **The field is xored with the random check and input to a cryptographic hash function.**
    - **The output is the new random number for this capability.**
    - **The restricted capability is then passed back to *matt*, who gives it to *holly*.**
    - **When *holly* uses the capability, the server notes that at least one bit in the rights field is 0.**
      - **It takes the rights field, xors it with the random number of the original capability, and hashes the result.**
      - **If the resulting hash matches with the random number in the capability, the capability is valid.**

Chapter 14 ACCESS CONTROL MECHANISMS

- **<u>Amplifying Capabilities</u>**
  - Amplifying allows temporary increase of privileges.
  - The idea of modular programming (especially of abstract data types) requires that the rights a process has over an object be amplified.
  - Consider the following abstract data type for a counter:

```
module counter;
        procedure entry increment (var ctr: integer);
        begin
                ctr:= ctr + 1;
        end;
        function entry getval (ctr: integer);
        begin
                getval := ctr:
        end;
        procedure entry clear (var ctr: integer);
        begin
                ctr:= 0;
        end;
end.
```

  - Suppose *x* is declared to be a <u>counter</u>.
    - The rules of abstract data types allow that objects to be accessed only by the *counter* module.
    - Initially, the capability for *x* would contain the right to invoke the *counter* model only.
    - When the object is passed to the counter model, the process must now be able to read and write to that object.
    - Hence, the capability must be amplified temporarily while the module *counter* is active.

Chapter 14 ACCESS CONTROL MECHANISMS

- **Revocation of Rights**
  - In a capability system, revoking access to an object requires that all the capabilities granting access to that object be revoked.
  - Conceptually, each process could be checked, and the capabilities deleted.
    - The cost of this operation would be unacceptable.
    - Hence, alternative methods are used.
  - The simplest mechanism is indirection.
    - Define one or more *global object tables*.
    - Each object has a corresponding entry in a table.
    - Capabilities do not name the objects directly. They name the entry in the table corresponding to the object.
    - This scheme has several advantages:
      - To revoke capabilities, the entry in the global object table is invalidated.
      - If only some of the capabilities are to be revoked, the object can have multiple entries, each corresponding to a different set of rights or a different group of users.
    - EXAMPLE:
      - Amoeba uses essentially this scheme.
      - To revoke a capability, the owner of the objects requests that the server change the random number and issue a new capability.
      - This invalidates all exiting capabilities.
  - An alternative revocation mechanism uses abstract data type managers.
    - Each abstract data type includes a revocation mechanism.
    - When access is to be revoked, the type manager disallows further accesses by the subject whose rights are being revoked.

Chapter 14 ACCESS CONTROL MECHANISMS

- **<u>Comparison with Access Control Lists</u>**
  - 2 questions underlie the use of access controls:
    - Given a subject, what objects can it access, and how?
    - Given an object, what subjects can access it, and how?
  - In theory, either access control lists or capabilities can answer these questions.
  - For the 1st question, capabilities are the simplest.
    - Just list the elements of the subject's associated C-List.
  - For the 2nd question, ACLs are the simplest.
    - Just list the elements of the object's access control list.
  - In an ACL-based system, answering the first question requires all objects to be scanned.
    - The system extracts all ACL entries associated with the subject in question.
  - In a capability-based system, answering the second question requires all subjects to be scanned.
    - The system extracts all capabilities associated with the object in question.

Chapter 14 ACCESS CONTROL MECHANISMS

14.3 Locks and Keys

- The locks and keys technique combines features of access control lists and capabilities.
- A piece of information (the lock) is associated with the object.
- A second piece of information (the key) is associated with those subjects authorized to access the object and the manner in which they are allowed to access the object.
- When a subject tries to access an object, the subject's set of keys is checked.
  - If the subject has a key corresponding to any of the object's lock, access of the appropriate type is granted.
- The difference between locks and keys and other access control mechanisms is the dynamic nature of locks and keys.
- Gifford suggests a cryptographic implementation of locks and keys.
  - Object o is enciphered with a cryptographic key.
  - The subject has a deciphering key.
  - To access the object, the subject deciphers it.
  - This provides a simple way to allow n subjects to access the data (called *or-access*).
  - Simply encipher n copies of the data using n different keys, one per subject.
  - The object is then represented as $o' = (E_1(o), \ldots, E_n(o))$.
  - The system can easily deny access except on the request of n subjects (called *and-access*).
  - Simply iterate the cipher using n different keys, one per subject: $o' = E_1(\ldots(E_n(o))$.
  - Requires consent of all n subjects to access the object.

Chapter 14 ACCESS CONTROL MECHANISMS

14.4 Ring-Based Access Control

- Multics (Multiplexed Information and Computing Service) is a mainframe timesharing operating system that began in 1965 and was used until 2000.
  - Multics began as a research project and was an important influence on operating system development.
  - The system became a commercial product sold by Honeywell to education, government, and industry.
- The Multics system generalizes the notion of a supervisor and user state with a protection mechanism called ring-based access control.
  - Files and memory are treated the same from the protection point of view.
  - A procedure may occupy a segment of the disk.
    - When invoked, the segment is mapped into memory and executed.
  - Data occupies other segments on the disk.
    - When accessed, they are mapped into memory and accessed.
  - There is no conceptual difference between a segment of memory and a segment on a disk.
  - Segments are of 2 kinds: data and procedure.
    - A segment could have the following rights: *r*, *w*, *e*, and *a*.
    - These rights are contained in access control lists.
  - In addition, the Multics system defines a sequence of protection rings numbered from 0 to 63.
    - The kernel resides in ring 0.
    - The higher the ring number, the lower the privileges of the segments in that ring.

Chapter 14 ACCESS CONTROL MECHANISMS

14.5 Propagated Access Control Lists

- The Propagated Access Control List (PACL) mechanism provides the creator of an object with the control over who can access the object.
- PACL is ideal for the ORCON policy.
- The creator is kept with the PACL.
  - Only the creator can change the PACL.
  - When a subject reads an object, the PACL of the object is associated with the subject.
  - When a subject creates an object, the PACL of the subject is associated with the object.
  - $PACL_{subject}$ means that subject is the originator of the PACL.
  - PACL(*entity*) is the PACL associated with *entity*.
- EXAMPLE:
  - Ann creates the file *dates*.
  - Ann wants to control who can read the file.
  - The file's PACL is the PACL associated with Ann: PACL(*dates*)=$PACL_{Ann}$.
- Multiple creators
  - Let the PACL of an object *o* be $PACL_s$.
  - When another subject *s'* reads *o*, PACL(*o*) must augment PACL(*s'*).
  - Otherwise, *s'* could create another subject *o'* and copy the data from *o* to *o'*.
  - An object can have PACLs associated with 2 creators.
  - Both creators control access to the object.