

Access Control

CSC 348-648



Spring 2013

Attempted Topics

- Describe language/models for talking about security systems
 - ACM, Capabilities and ACLs
 - How do we talk about existing systems
 - What to consider in a new design
- Discussion of how some existing systems work
 - Unix access controls

System Security

- Authentication (password/crypto/etc.)
 - Who are you?
- Authorization (Access control)
 - Now that you are authenticated, *what are you allowed to do*
 - Focus is policy and enforcement
- Enforcement Mechanisms
 - Ensures that the policy is followed
- N.B. *It is often difficult to divide policy and mechanism*

Concert Example

- Authentication, collect ticket
- Access policy (*list of rules*)
 - Over 18, allowed in
 - Over 21, allowed to purchase expensive drinks
 - VIP, given expensive drinks
- Enforcement mechanisms (how do you enforce the policy)
 - Building, locks, security staff, and velvet ropes
- Some interesting points
 - Are tickets anonymous or specific to a person? Date?
 - What happens if you want to leave and return?

Basic Ideas

- All security architectures combine isolation and controlled sharing
- Examples isolation include
 - *Unplug the computer from the network, I guess...*
 - Segment the network (physically or virtually), required to some degree for HIPPA compliance...
 - Military and national labs use this approach (enclaves) to separate networks based on secrecy
- Sharing is necessary, but there are some challenges

Why Have Access Control

- Prevent authorized users from having unlimited access to resources
- Limit access of unauthorized users that manage to break in

Are the previous two statements the same?
- Two fundamental principles of access control
 1. Least privilege (Saltzer & Schroeder 1975) a user should only be given sufficient access to perform duties
 2. Fail-safe default, the default response is to deny access
- *Applying these principals and providing a useful system is challenging*

Operating Systems

- Most heavily studied area for access control
But what is another area that uses access control?
- First computers where single user/no network
 - Timesharing systems introduced need for access control
- After 40+ years, still an open area of research
 - First 20 years: restrict what a user can do with data, focus on military problems, thought problem was malicious users
 - Last 10: Malicious applications the primary problem
What is the difference in terms of the solution?
- What we really want is access control policy dictated by usage models, threat model, and applications... *simple right?*

Difference Between Access Control and IDS

- Consider soundness (can I make a definite allow/deny decision)
 - An IDS generates a probabilistic response, *I think this is bad*, covers a larger range of behavior
 - An IDS that is 100% accurate is actually performing enforcement, and really is an IPS
- Consider completeness (do I catch every bad action)
 - Access control systems should enforce some property

Computer System Operations

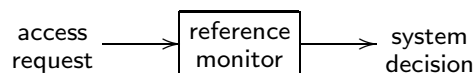
- Consider the operation of modern computers
 - Computers update data and mediate access to shared resources
 - Access control is provided, but primarily for integrity

What are we interested in?

- Operating systems designed to provide generic operations
 - Applicable to a variety of resources, regardless of meaning
- As a result, it is difficult to map high level security requirements using these low level operations

Overview

- User requests access to a resource
- The **reference monitor** determines if allowed
 - Establishes the validity of the request
 - Returns a decision, granting or denying



- Decision is based on the access control policy

System States and Security

- The **state** of a system is a collection of *values*
 - System components such as memory, secondary, storage, etc...
 - Subset that deals with protection is the **protection state**
- Consider a the set of possible protection states P
 - Some subset Q of P consists of exactly those states in which the system is authorized to reside
 - Therefore, if the system state is in Q it is secure, if it is in $P - Q$ it is **not** secure
- Characterizing Q is a function of the *security policy*
- Preventing the system from $P - Q$ is the security mechanism

Access Control Matrix

- **Access control matrix** is a precise protection state model
 - Characterizes the right of subjects WRT other entities
 - The description can be used to compare the current state
- The set of all protected objects is O
 - Objects are entities relevant to the protection of the system
- Let S be the set of subjects or active objects
 - This includes users and processes

- The relationship between S and O is given by the matrix A
 - Relationship is drawn from a set R of rights
- Let $s \in S$ and $o \in O$, then a subject s has the right over an object o defined by $a[s, o]$, where $a[s, o] \in R$

	f_1	f_2	p_1	p_2
p_1	read, write, own	read	read, write, execute, own	write
p_2	append	read, own	read	read, write, execute, own
Legend: f is a file and p is a process				

What is the set of protection states, (S, O, A) ?

- Request r is granted if it belongs to the matrix entry $a[s, o]$
 - Request (p_1 write f_2) would be denied

Another Example

	f_1	f_2	f_3
u_1	r, w	r, w, x	r, w
u_2	\emptyset	r, x	r
Legend: u is a user and f is a file			

- Access Control List (ACL) is a column in the matrix
 - Focus is on the objects (Windows NT and Unix use ACLs)

How does Unix use ACLs?

- For example $f_2 : (u_1, [r, w, x]), (u_2, [r, x])$

Things Change

- As processes execute the state of the protection system changes
 - Files created, deleted, etc...
 - Access matrix must be updated to reflect changes
- *Primitive commands* that update the matrix
 - **Create subject s**
Precondition: $s \notin S, O$
Postcondition: $S' = S \cup s, O' = O \cup s$
 $\forall y \in O' \quad a'[s, y] = \emptyset$
 $\forall x \in S' \quad a'[x, s] = \emptyset$
 $\forall x \in S \quad \forall y \in O \quad a'[x, y] = a[x, y]$
Does this add any rights?

- **Create object o**
Precondition: $o \notin O$
Postcondition: $S' = S, O' = O \cup o$
 $\forall x \in S' \quad a'[x, o] = \emptyset$
 $\forall x \in S \quad \forall y \in O \quad a'[x, y] = a[x, y]$
- Other primitive commands include: adding/remove a right and add/remove subject or object
 - These can be combined to create more complex commands

These procedures describe how to modify rights in an ACL. How do you determine if a right should be modified?

Rights

- **Copy right** and **own right** need more discussion
 - Involve the *attenuation of privilege*
 - *A subject cannot give away rights it does not possess*
- Copy right is also called *grant right*
 - Allows the possessor to copy rights to another
 - Principle of attenuation, only rights of the grantor are allowed
 - Whether the grantor loses those rights depends on the system

What?

- Own right enables possessor to add or delete rights
 - Unix system the owner can change using `chown` to change permissions granted to others

Attenuation of Privilege

- If a subject does not possess a right over an object, it should not be able to give that right to another subject
 - This is the **principle of attenuation**
- On most systems, the owner of an object can give others rights over the object whether the owner has those rights enabled or not

Unix follows this convention, does it violate the principle?

- For example, assume Pluf owns a file called `example.txt`
 - Even if he cannot read it, he can give this permission to others

```
chmod go+r /home/pluf/example.txt
```

What if another user tries the command on the file?

Access Control Matrix Use

- Primary abstraction for computer security
 - Can represent any expressible security policy
- Not used in practice, *yes Chukar I just wasted your time...*

Why?

- Number of subjects and objects are typically too large
 - Most entries in the matrix will be blank or the same
 - Creation/deletion of objects/subjects is difficult in a matrix
- Several variations of the ACM exist
 - *More convenient, but won't be as comprehensive...*

Access Control Lists

- A variation of the ACM is the Access Control List (ACL)
 - Each object has a set of pairs
 - Each pair defines a subject and the rights
- Let S be the set of subjects and R the set of rights
 - An ACL l is a set of pairs, $l = \{(s, r) : s \in S, r \subseteq R\}$
 - For example
$$l(f_1) = \{(p_1, \{r, w, o\}), (p_2, \{r\})\}$$
$$l(f_2) = \{(p_1, \{r\}), (p_2, \{r, w, o\})\}$$

Have we gained any advantage over a matrix?

Abbreviated ACL

- Unix uses an abbreviated ACL, users are grouped into three classes
 - *Owner* of the file
 - *Group owner* of the file
 - All others
- Each class has a separate set of rights
 - Read (r), write (w), and execute (x)
 - Assume you did a `ls -al` command with the results

```
-rw-r--r-- 1 pluf faculty 17k Sep 24 16:08 example.cpp
```

owner group other owner group size date file name
perm perm perm

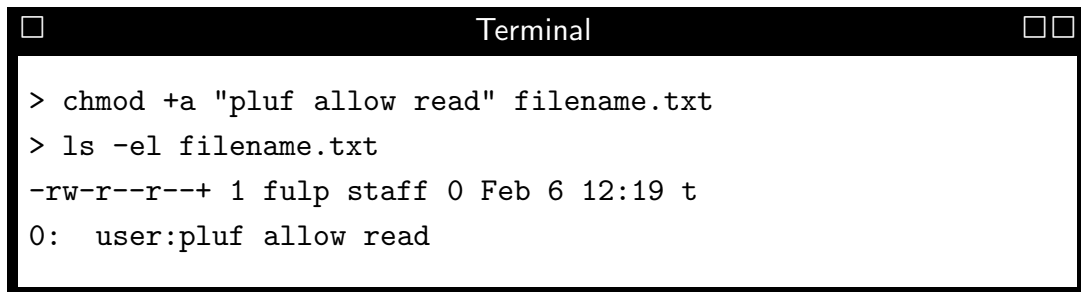
What are the advantages and disadvantages of this approach?

- Assume you have a list of users with the same permissions
 - Use (create) a Unix *group* and add the users

```
Terminal
> groups
pluf adm cdrom sudo
> groupadd awesomegroup
> useradd -g awesomegroup pluf
> useradd -g awesomegroup gort
> groups awesomegroup
pluf gort
```

- There is a loss of granularity with the abbreviated approach
 - Assume you wanted *LeAchim* to read a file, *Ylime* to write to the file, *Ttosc* to read and write, and *Turd* to execute it?

- OSX also uses the concept of Access Control Lists per file
 - List of rules that indicate the [*file, actor, permission, if-action-allowed*]
 - For example, permissions are applicable to non-directory filesystem objects include, read, write, append, and execute
 - Evaluate the rules until a first match is found (*conflicting permissions? ACLs only have positive permission, not negative... I think*)
 - If no ACL rules match, then evaluate the Unix permissions on the file
 - Newer Linux installs have the same ACL functionality (`setfacl`)



```
> chmod +a "pluf allow read" filename.txt
> ls -el filename.txt
-rw-r--r--+ 1 fulp staff 0 Feb 6 12:19 t
0: user:pluf allow read
```

Creation and Maintenance of ACL

1. *Which subjects can modify an object's ACL?*
2. *Is there a privileged user, how are ACLs applied?*
3. *Does the ACL support groups or wildcards?*
4. *How are contradictory access control permissions handled?*
 - If one entry grants read privileges only and another grants write only, *which is applied?*
5. *Is there a default setting, do the ACL permissions modify it?*

Capability Lists

- C-list is like a row in the access control matrix
 - Each subject has a set of pairs associated with it
 - Each pair consists of an object and a set of rights
- Let O be the set of objects and R the set of rights
 - A C-list c is a set of pairs, $c = \{(o, r) : o \in O, r \subseteq R\}$
 - For example
$$c(p_1) = \{(f_1, \{r, w, x, o\}), (f_2, \{r\})\}$$
$$c(p_2) = \{(f_1, \{r\}), (f_2, \{r, w, x, o\})\}$$
- Consider opening a file in Unix
 - The process gives the filename to kernel
 - Kernel obtains the file's *inode number*

- Kernel determines if the request should be granted

Does the ACL help in this?

- If granted, kernel returns a capability called a *file descriptor*

- The architecture of capabilities is more *interesting* than ACL
 - For ACL, the OS controls the ACL and process identity; therefore, a user process can only change access via the OS
 - In contrast, a process must identify a capability in order to use it; as a result, process **must** have some control over capabilities

What? Why is a C-list more dangerous than ACL?

Analogy of Capabilities-Based Access

- Suppose you want a safety-deposit box to keep valuable items
 - You also want trusted friends to make deposits and withdraws
- ACL approach, bank maintains a list of authorized people
 - Authentication: Bank must authenticate
 - Banks involvement: (i) Store the list, and (ii) verify friends
 - Forging access right: Bank safeguards the list
 - Add a new person: You must visit the bank, update list
 - Delegation: Friend cannot extend their privilege to others
 - Revocation: You can remove name(s) friends (*like Chukar*)

- Capabilities-list approach, bank issues you key(s) to the box
 - Authentication: Bank does not need to authenticate
 - Banks involvement: The bank need not be involved in any transactions
 - Forging access right: The key cannot be forged
 - Add a new person: You can let friends borrow the key
 - Delegation: Friend cannot extend their privilege to others
 - Revocation: You can ask for the key back, but it may not be possible to know whether or not the friend has made a copy (*it's Chukar after all...*)

Implementation of Capabilities

- Capabilities can be considered an *unforgeable token*
 - The token identifies the right of the subject
 - Two mechanisms are used to make tokens unforgeable: *protected memory* and *cryptography*
- Protected memory, c-list is not directly given to the process/users
- Cryptography used to maintain the integrity of the c-list

Unix File Access

- Consider the following C program for reading and writing

```
1 /* /etc/passwd permission allows anyone open and read */
2 fd = open("/etc/passwd", "r");
3 read(fd, buffer, 10);
4 write(fd, buffer, 10);
5 /* before next statement executed, assume root modifies
6    permission on /etc/passwd to 600, users cannot read */
7 read(fd, buffer, 10);
```

- Second and third lines will succeed since open and read permitted
- Fourth line will fail, *due to permissions (ACL)?*
- Seven line will succeed

How? Permissions have changed to not allow reads?

- The file descriptor (fd) is actually a capability
 - When a file is opened descriptor created and stored in `flip` (stored in kernel space)
 - User is given fd which is an index to the table
 - `flip` is a capability list describing what the user/process can do with the file

Capabilities, Rights, and Domain Space

- One access right to an object is **transfer** or **propagate**
 - Subject can pass copies of capabilities to other subjects
 - Capability has a list of access rights that can be transferred
 - A can pass copy capability to B , who can pass copy to C ; however, B can prevent C from passing this capability by omitting the transfer right
- As a process executes, it operates in a **domain space**
 - Domain is the collection of objects to which process has rights
 - This can include, programs, files, I/O devices, etc...

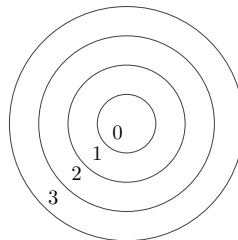
What is this like in Unix?

ACL and C-List

- Two important questions for any access control
 1. *Given a subject, what objects can it access and how?*
 2. *Given an object, what subjects can access it and how?*
- First question C-list is easier than ACL, vice versa for the second *Why?*
- The second question is *supposedly* asked more often
 - As a result, more systems implement ACL*When is the first question more important?*

Ring-Based Access Control

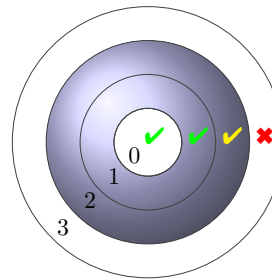
- A system of protection rings are defined
 - Numbered from 0 to n , where 0 has the most privileges
- Processes, data, and procedures assigned a ring, r
 - For example the kernel is a ring 0 process, OS is ring 1, utilities are ring 2, and user processes are ring 3



- Processes can cross ring boundaries, under certain circumstances
 - Invokes the *gatekeeper* to determine if cross is legal*Why would a process cross a ring?*

Data Segments

- Data segments have a *access bracket*
 - Access bracket is defined as (a_1, a_2) , where $a_1 \leq a_2$
 - Gatekeeper checks the ring numbers to constrain ring crossing
- Assume procedure executing in ring r wants to access data
 - If $r \leq a_1$, then access is permitted
 - If $a_1 < r \leq a_2$, then r and x are permitted only
 - If $r > a_2$, then all access denied

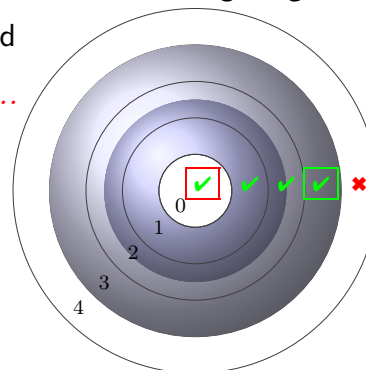


assume $a_1 = 1$ and $a_2 = 2$

Procedure Segments

- Procedure segments have a triple of r values (a_1, a_2, a_3)
 - (a_1, a_2) is the access bracket and (a_2, a_3) is the call bracket
- Assume ring r procedure wants to access a procedure segment
 - If $r \leq a_1$, then access is permitted, but *ring fault*
 - If $a_1 \leq r \leq a_2$, access permitted
 - If $a_2 < r \leq a_3$, access permitted if called through a *gate*
 - If $r > a_3$, then all access denied

Are we providing secrecy, integrity, ...



assume $a_1 = 1$, $a_2 = 2$, and $a_3 = 3$

Ring Examples

- Assume a data segment has an access bracket of (2, 4)

Can a ring 1 process read the data? Can a ring 3 or 5?

- Assume a procedure segment has a bracket of (2, 4, 6)

What is the access bracket? What is the call bracket?

What occurs if ring 1 process calls? What about ring 3, 5, or 7?

Why have brackets? Should data and procedures be in one ring?

What does ring access control provide that the other methods (ACL, capabilities list, etc..) do not?

Rings and Brackets

- Ideally an access bracket should only contain one ring
 - Processes from other rings would cause excessive crossings
- Consider an *access file* routine
 - If the routing covers multiple rings, want less crossings
 - Fewer crossings requires less OS intervention
- Consider a procedure segment
 - Suppose certain users should have access, another group should have specific access, while a third group should have no access
 - First group is in the access bracket, second group in a call bracket, and the last group is beyond the call bracket
- The Multics and VAX systems utilized a *ring-based access control*

Multics Rings

- Multics stores a *segment descriptor*
 - **Access bracket**, a pair of integers, b_1 and b_2 , where $b_1 \leq b_2$
 - **Limit**, an integer b_3 , such that $b_3 \geq b_2$
 - **List of gates**, entry points where segments may be called
- If a process executing in ring i tried to execute a segment (b_1, b_2)
 - Allowed if $b_1 \leq i \leq b_2$ and the current ring number of the process remains i
 - If $i \leq b_1$ then the call was allowed to occur and the *current ring number* of the process was changes to b_1

Why temporarily change the ring number of the process?

- If $i > b_2$ then the call allowed only if $i \leq b_3$, and the call had been directed to one of the designated entry points in the list-of-gates. If the call was successful, the current ring number of the process was changed to b_2 .

Does this make sense?

- Regarding read/write, the rules imply that a process with ring i that tries to read/write a segment with access bracket (b_1, b_2) , is allowed to do so if $i \leq b_2$.

The Multics scheme has the disadvantage that the ring (hierarchical) structure does not allow enforcement of the need-to-know principle. So what?

Unix and Rings

- Unix provides a simplified version of the Multics access control
 - Can change the access rights of a process dynamically
 - Specifically, a process can switch between user and kernel mode
- For example, a process executing with the access rights of one user
 - If it executes a file owned by another user, it gets the access rights of the second user
 - Assumes the SETUID bit is set in the object file
- When the command interpreter subprocess forked to process the *mail command* executes the mail program it acquires the access rights of root and can create/modify a file (owned by the receiver) in the directory `/usr/spool/mail`

This can be a problem...