

CSc 466/566

Computer Security

7 : Access Control

Version: 2019/09/18 13:51:08

Department of Computer Science
University of Arizona

collberg@gmail.com
Copyright © 2019 Christian Collberg

Christian Collberg

1/54

Outline

- 1 Access Control Models
 - Role-Based Access Control
 - Over-entitlement
- 2 Review of Unix File System and Permissions
- 3 SetUID
- 4 Summary

Access Control Models

2/54

Mechanisms — Access Control

Definition (Access Control)

Rules and policies that restrict access to confidential information.

- Information can be accessed by those with a need to know.
- Can be
 - identity based — person's name or computer's serial number.
 - role based — what position (manager, security expert) the user has in the organization.

Access Control Models

3/54

Access Control Models

- We should determine who has the right to access to a piece of information.
- If we can control access to information, we can prevent attacks against confidentiality, anonymity, and integrity.
- Someone (eg. system administrators) should restrict access to those who should have access: they should apply the principle of least privilege.

Access Control Models

4/54

Least privilege

Definition (Least privilege)

Users and processes should operate with no more privileges than they need to function properly.

- Limits the damage if an application or account is compromised.
- Examples:

① The military:



② Windows NT 3.0:



③ Web browsers:



Subjects, Objects, and Rights

Definition (subject)

User, group, or system that can perform actions.

Definition (object)

File, directory, document, device, resource for which we want to define access rights.

Definition (rights)

Ways in which the subject can access the object: read, write, copy, execute, delete, annotate, ...

Access Control Matrices

	object 1	object 2
subject 1	rights	rights
subject 2	rights	rights

- Each table cell holds the rights of the subject to access the object.

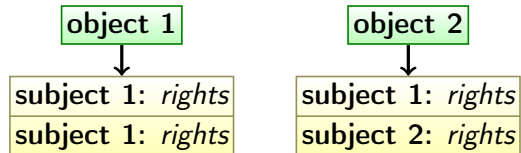
Access Control Matrices: Example

	/etc/passwd	/usr/bob/	/admin/
root	read, write	read, write, execute	read, write, execute
alice	read		
bob	read	read, write, execute	
backup	read	read, execute	read, execute

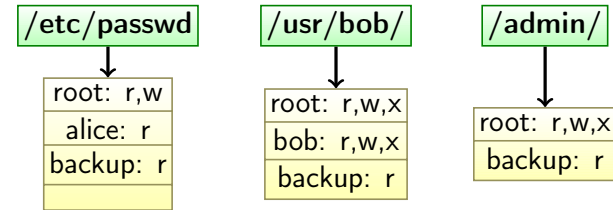
- **Advantages**: fast access
- **Disadvantages**: large size= $\#subjects \cdot \#objects$

Access Control Lists (ACLs)

- Is **object-centered**: for every object o list (only) the subjects s that have access to o , and s ' access rights.



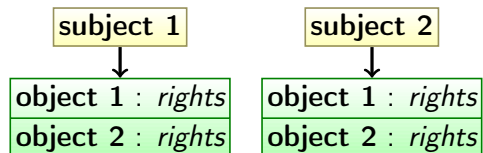
Access Control Lists (ACLs)...



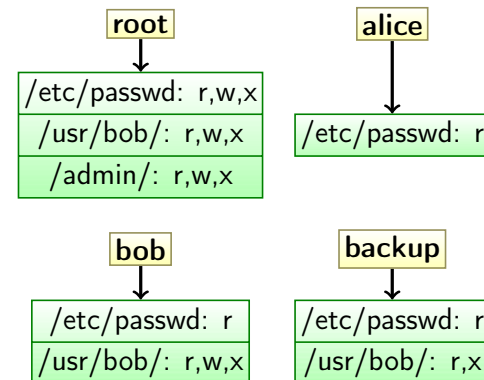
- Advantages**: size, o 's ACL can be stored directly as o 's metadata
- Disadvantages**: can't enumerate a subject's rights (for example when a subject is removed from the system).

Capabilities

- Is **subject-centered**: for every subject s list (only) the objects o that s has non-empty access to, and o 's access rights.



Capabilities...



Capabilities. . .

- **Advantages:**
 - 1 size
 - 2 easy to enumerate a subject's rights (for example when a subject is removed from the system)
 - 3 easy to check if subject s can access object o .
- **Disadvantages:** can't enumerate who has access to an object o .

Exercise: Access Control Matrix

- Users: Alice, Bob, Cyndy.
- Alice owns `a.txt`, Bob and Cyndy can read it.
- Bob owns `b.txt`, Cyndy can read and write it, Alice can read it.
- Cyndy owns `c.txt`, only she can read and write it.
- 1 Create the access control matrix.
- 2 Cindy lets Alice read `c.txt`. Alice no longer allows Bob to read `a.txt`. Show the new matrix.

Source: Bishop, *Introduction to Computer Security*.

Exercise: Access Control Matrix

- 1 Create the access control matrix.

	a.txt	b.txt	c.txt
Alice			
Bob			
Cyndy			

Exercise: Access Control Matrix

- 2 Cindy lets Alice read `c.txt`. Alice no longer allows Bob to read `a.txt`. Show the new matrix.

	a.txt	b.txt	c.txt
Alice			
Bob			
Cyndy			

Exercise: Show the ACL for b.txt!



Exercise: Show the Capabilities for Alice!

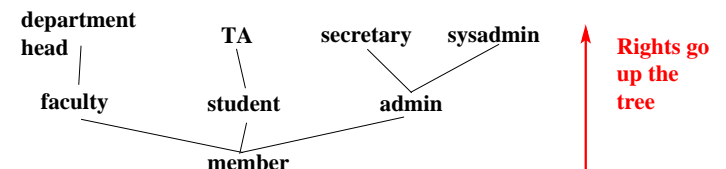


Role-Based Access Control (RBAC)

- In Role-Based Access Control we replace **subjects** by **roles** in any of the access control data structures.
- Each role gets the appropriate access rights.
- Subjects are assigned to roles.
- Examples:
 - 1 CS department roles: faculty, student, sysadmins, department head, TA, ...
 - 2 CS department subjects: bob={student,TA}, alice={faculty}, wendy={faculty,department head}
- A subject's access rights is the union of the rights of its various roles.

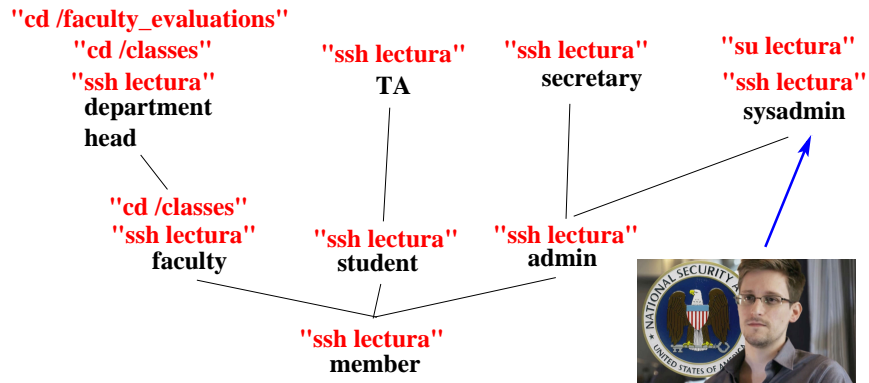
Role Hierarchies

- In a **role hierarchy** a node n inherits all the rights of its children.
- Computer Science department example:

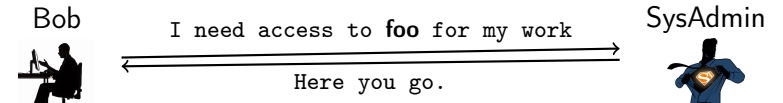


- **Advantages**: fewer rules since there are fewer roles than subjects.
- **Disadvantages**: not implemented in current operating systems.

Role Hierarchies...



Problems: Over-entitlement



- Most important is to get work done.
- 50-90% of employees are **over-entitled**.
- Rights granted are seldom revoked.
- You can predict how long someone has worked in an organization by how much access they have!

Problems: Abuse for Personal Gain

- Leak celebrities' medical/tax/passport/police records.
- NSA employees spy on their girlfriends.
- Employees leave with corporate secrets.



Snowden used web crawlers to access and copy about 1.7 million documents, according to the New York Times. Snowden had broad access to the NSA's complete files because he was working as a technology contractor ..., helping to manage the agency's computer systems.

Problems: Insider Attacks. . .

... the NSA had built enormously high electronic barriers to keep out foreign invaders, it had rudimentary protections against insiders.

<http://www.computerweekly.com/news/2240214065/NSA-failed-to-detect-Snowdens-unsophisticated-insider-attack>

Problems: Insider Attacks. . .

[A] survey of more than 700 IT security decision-makers found that less than a third of respondents said they block privileged user access to data to mitigate insider attacks. However, the study also showed attitudes changing, with 45% saying that Snowden's revelations about US internet surveillance has caused them to be more aware of insider threats. Some . . . 70% said they were using or planning to use data access controls.

Cherax Snowden



- Crawfish named by German scientists in his honor.

http://www.nola.com/environment/index.ssf/2015/08/new_crawfish_species_named_aft.html

Outline

- 1 Access Control Models
 - Role-Based Access Control
 - Over-entitlement
- 2 Review of Unix File System and Permissions
- 3 SetUID
- 4 Summary

Unix file permissions

- Every file has Owner, Group, and Other permission.

- **Owner:**



- **Group:**



- **Other:**



File Access Modes

- What does read/write/execute mean for a regular (non-directory) file?

- **Read:**



- **Write:**



- **Execute:**



Directory Access Modes

- What does read/write/execute mean for a directory (folder)?

- **Read:**



- **Write:**



- **Execute:**



Operations

- What do these unix commands do?

- **chmod:**



- **chown:**



- **chgrp:**




Unix File System Tree

```
/ (root)
|
+ -- bin
|
+ -- usr
|   |
|   + -- local
|   |
|   + -- bin
|
+ -- home
|   |
|   + -- alice
|   |
|   + -- bob
|
+ -- etc
```


Exercises

- What are the file permissions for / (root)?

```
> ls -dl /
d????????? 48 root wheel 1536 21 May 16:18 /
```


- What are the file permissions for /etc/passwd?

```
> ls -ld /etc/passwd
-????????? 1 root wheel 6774 15 Feb 2018 /etc/passwd
```



Outline

- 1 Access Control Models
 - Role-Based Access Control
 - Over-entitlement
- 2 Review of Unix File System and Permissions
- 3 SetUID
- 4 Summary

How to change password?

- On Unix, you invoke the **passwd** program to change your password.
- The **passwd** program updates the **/etc/passwd** file.
- But, only **root** can change **/etc/passwd**:

```
> ls -l /etc/passwd
-rw-r--r-- 1 root wheel 5253 Nov 5 2013 /etc/passwd
```
- So, how can a normal user change their own password?

How to play a CD?

- On Unix, you invoke the **mount** program to load a CD.
- **mount** changes the file system tree.

```
/ (root)
|
+ -- mnt
|   |
|   + -- cdrom
+ -- bin
```

How to play a CD...

- But, only **root** can add a mount point:

```
> ls -ld /
drwxr-xr-x 54 root wheel 1904 Sep 17 01:04 /
```
- So, how can a normal user play



???

What's the problem?

- Processes inherit the permissions of their parents!
- When Bob invokes **passwd** it runs with his privileges!
- But **/etc/passwd** is owned by **root**!
- Thus, Bob can't change **/etc/passwd**!

The setuid bit

- Unix permissions include a **setuid** bit.
- If prog has **setuid=1** ⇒
 - prog runs with **effective uid** (euid) of its owner.

```
> ls -l /usr/bin/sudo /usr/bin/passwd /bin/mount
-rwsr-xr-x 1 root wheel 94792 Jun 17 2014 /bin/mount*
-rwsr-xr-x 1 root wheel 42824 Sep 12 2012 /usr/bin/passwd*
-rwsr-xr-x 2 root wheel 71280 Mar 12 2015 /usr/bin/sudo*
```

Process Privileges

- Each process has a **uid** (user ID) and **gid** (group ID) that identifies the user/group for the process.
- **Effective User ID** (euid) — used when deciding a process' access privileges.

Change Password — Take 2

- passwd is owned by root.
- **But, passwd has setuid=1!**
- Bob runs passwd.
- Bob's passwd process runs with root permissions.
- \Rightarrow Bob can change /etc/passwd!

Play Audio CD — Take 2

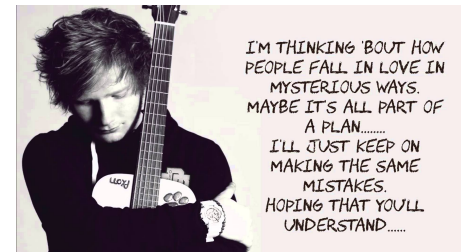
- mount is owned by root.
- **But, mount has setuid=1!**
- Bob runs mount /cdrom.

```
/ (root)
|
+ -- mnt
|   |
|   + -- cdrom
|
+ -- bin
```

- Bob's mount process runs with root permissions!

Play Audio CD — Take 2

- \Rightarrow We can now play this dreamy song!!!
- <https://www.youtube.com/watch?v=lp-E05I60KA>



SetUID issues

- What if a setuid program prog has a bug in it?
- Bob exploits the bug to make prog run arbitrary code!
- \Rightarrow Bob gets privileges of the program's owner!
- **Privilege escalation** scenario.
- setuid programs must be safe!

Privilege escalation

Definition (Privilege escalation)

Privilege escalation is the act of exploiting a bug, design flaw or configuration oversight in an operating system or software application to gain elevated access to resources that are normally protected from an application or user. The result is that an application with more privileges than intended by the application developer or system administrator can perform unauthorized actions.

Wikipedia: https://en.wikipedia.org/wiki/Privilege_escalation

SetUID: Safe example

```
static uid_t euid, uid;

int main(int argc, char* argv[]) {
    uid = getuid();
    euid = geteuid();
    seteuid(uid);           // Drop privileges!
    // Do something...
    seteuid(euid);          // Raise privileges!
    FILE *file = fopen("/var/log", "a");
    seteuid(uid);           // Drop privileges!
    fprintf(file, "...");    // Print using
                             // permissions of owner!
    fclose(file);
    return 0;
}
```

SetUID: Safe example. . .

- The example program runs with the user's permissions, most of the time.
- It raises permissions to the owner's in order to write to the log file.
- It drops privileges when it's done.

SetUID: Vulnerable example

- Consider this C program:

```
int main() {  
    system("ls");  
    return 0;  
}
```

- system() invokes /bin/sh.
- system("ls") therefore invokes /bin/sh which, in turn, invokes /bin/ls from inside your program!

SetUID: Vulnerable example...

```
int main() {  
    system("ls");  
    return 0;  
}
```

- But, how does the shell know which ls-program to invoke?
- It searches the PATH:

```
> echo $PATH  
.: /bin : /usr/local/bin : /home/collberg/bin
```

SetUID: Vulnerable example...

```
int main() {  
    system("ls");  
    return 0;  
}
```

- Now, assume this program has setuid=1.
- ⇒ The user can manipulate PATH to make system execute the wrong program:
 > PATH=./home/collberg/bin:/bin:/usr/local/bin
- How do you rewrite this program to make it secure?



Outline

- 1 Access Control Models
 - Role-Based Access Control
 - Over-entitlement
- 2 Review of Unix File System and Permissions
- 3 SetUID
- 4 Summary

Readings

- Chapter 1 in *Introduction to Computer Security*, by Goodrich and Tamassia.
- Chapter 3 in *Introduction to Computer Security*, by Goodrich and Tamassia.

Acknowledgments

Material and exercises have also been collected from these sources:

- 1 Bishop, *Introduction to Computer Security*.
- 2 Bhushan Jain, Chia-Che Tsai, Jitin John, and Donald E. Porter, *Practical Techniques to Obviate Setuid-to-Root Binaries*, <http://www3.cs.stonybrook.edu/~porter/pubs/jain-setuid.pdf>.
- 3 Hao Chen, David Wagner, Drew Dean, *Setuid Demystified*, <http://www.cs.berkeley.edu/~daw/papers/setuid-usenix02.pdf>