CY2550 Foundations of Cybersecurity

Access Control

Alina Oprea
Associate Professor, Khoury College
Northeastern University
March 12 2020

Authentication

- Verification of identity claim made by a subject on behalf of a principal
- Three classes of secrets:
  1. Something you know
     - Example: a password
  2. Something you have
     - Examples: a smart card or smart phone
  3. Something you are
     - Examples: fingerprint, voice scan, iris scan
- Desirable properties include being *unforgeable*, *unguessable*, and *revocable*

Authorization

- Authorization follows authentication
  - If asking what someone can do, you must know who they are
- Usually represented as a policy specification
  - What resources can be accessed by a given subject?
  - Can also include the nature of the access

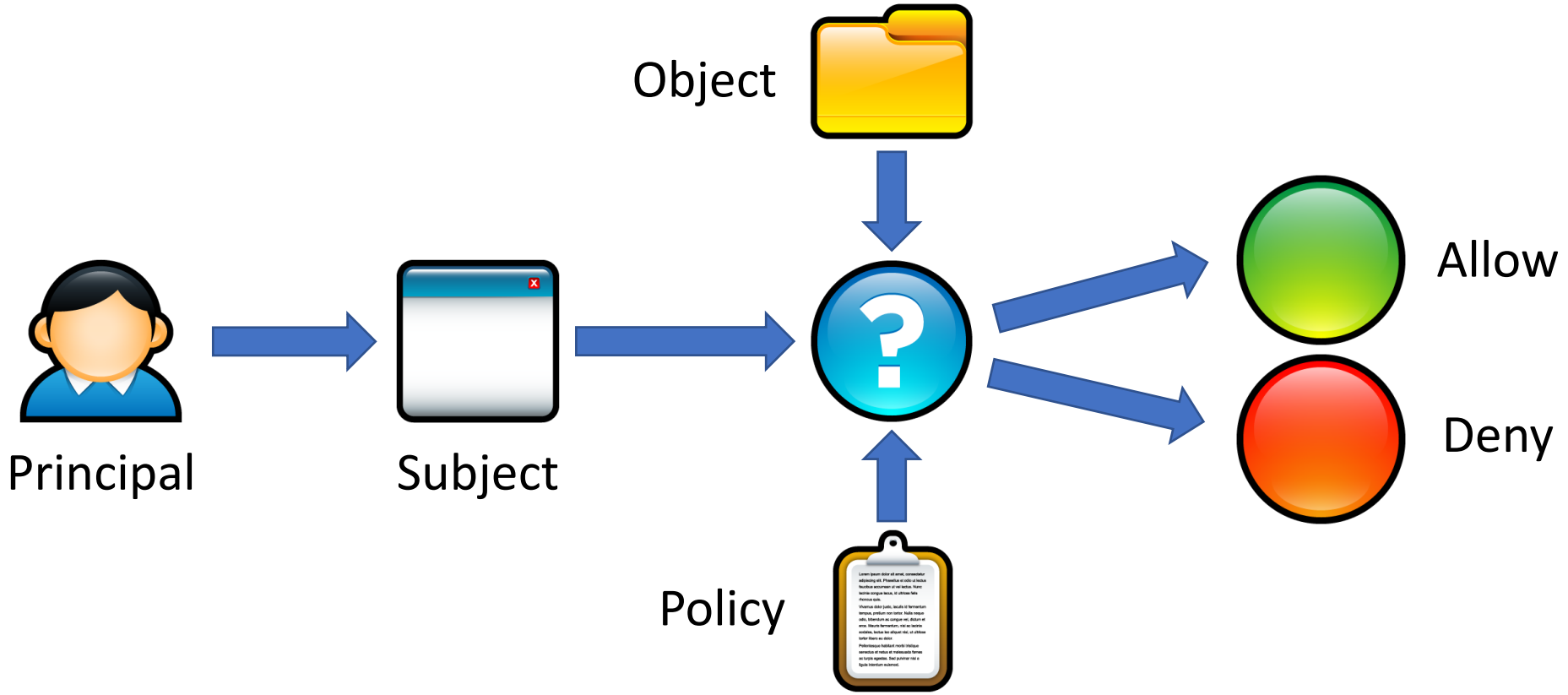
Access Control

- Policy specifying how entities can interact with resources
  - i.e., Who can access what?
  - Requires authentication and authorization
- Access control primitives

Principal	User of a system	
Subject	Entity that acts on behalf of principals	Software program
Object	Resource acted upon by subjects	Files Sockets Devices OS APIs

Access Control Check

- Given an access request from a subject, on behalf of a principal, for an object, return an access control decision based on the policy



Access Control Models

- Discretionary Access Control (DAC)
  - The kind of access control you are familiar with
  - Access rights propagate and may be changed at subject's discretion
- Mandatory Access Control (MAC)
  - Access of subjects to objects is based on a system-wide policy
  - Denies users full control over resources they create

Discretionary Access Control

Access Control Matrices

Access Control Lists

Unix Access Control

Discretionary Access Control

- According to Trusted Computer System Evaluation Criteria (TCSEC)

"A means of restricting access to objects based on the identity and need-to-know of users and/or groups to which they belong. Controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (directly or indirectly) to any other subject."

Access Control Matrices

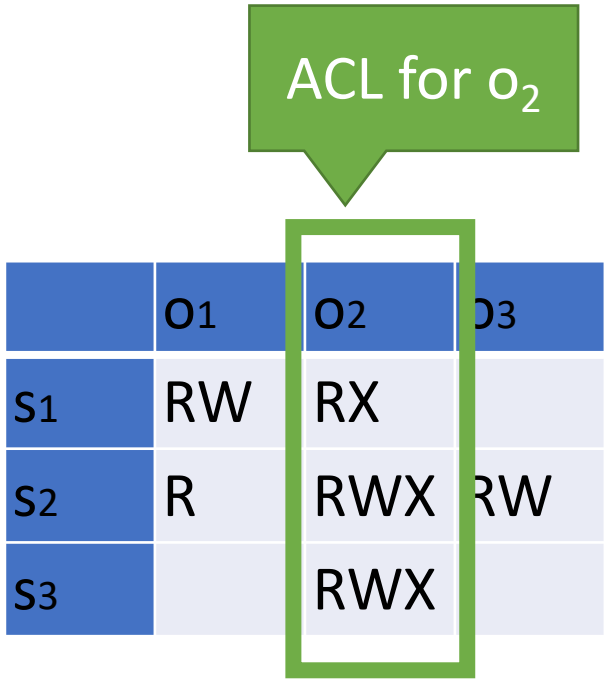
Given subjects $s_i \in S$, objects $o_j \in O$, rights {**R**ead, **W**rite, e**X**ecute},

	o_1	o_2	o_3
s_1	RW	RX	
s_2	R	RWX	RW
s_3		RWX	

- Introduced by Lampson in 1971
- Static description of system protection state
- Abstract model of concrete systems

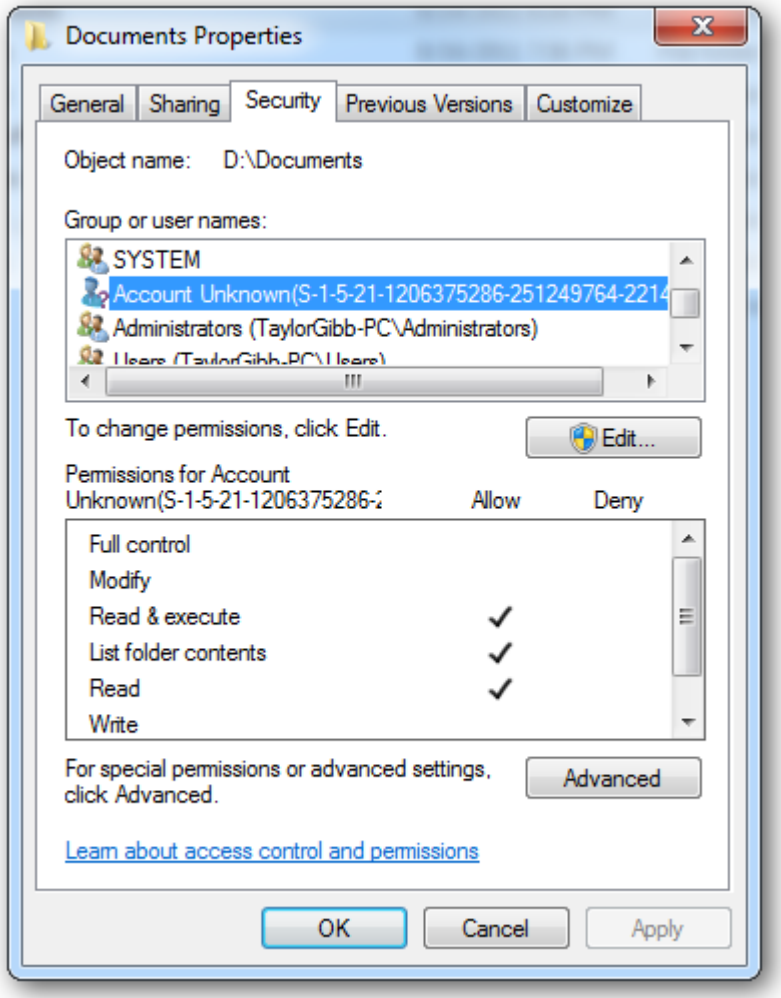
Access Control List (ACL)

- Each object has an associated list of subject → operation pairs
- Authorization verified for each request by checking list of tuples
- Used pervasively in filesystems and networks
  - "Users a, b, and c can read file x."
  - "Hosts a and b can listen on port x."



Windows ACLs


	D:\Music	D:\Images	D:\Documents
System	RWX	RWX	RWX
Administrators	RW	RW	RW
Users:alice	RWX	RW	
Users:bob		RW	R



ACL Review

The Good

- Very flexible
  - Can express any possible access control matrix
  - Any principal can be configured to have any rights on any object

The Bad

- Complicated to manage
  - Every object can have wildly different policies
  - Infinite permutations of subjects, objects, and rights

Unix-style Permissions

- Based around the concept of owners and groups
  - All objects have an owner and a group
  - Permissions assigned to owner, group, and everyone else
- Authorization verified for each request by mapping the subject to owner, group, or other and checking the associated permissions

Unix Permissions

Directory

Permission to list the contents of a directory

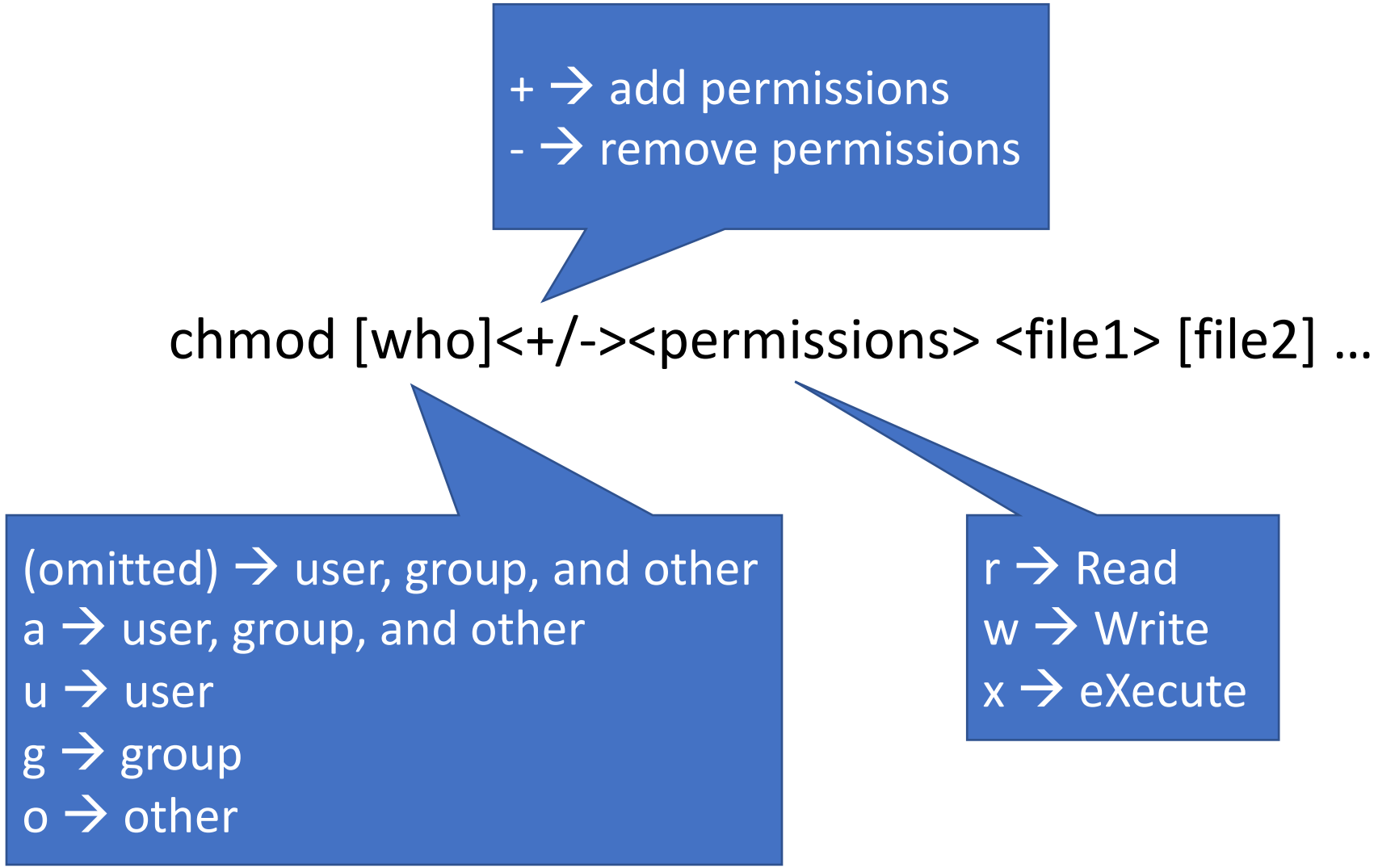
```
alice@DESKTO    $ ls -l
drwxrwxrwx 0 alice  alice    512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 alice  alice     17 Jan 29 22:46 my_file
-rwxrwxrwx 1 alice  faculty  313 Jan 29 22:47 my_program.py
-rw------- 1 root   root     896 Jan 29 22:47 sensitive_data.csv
```

Owner Group Other

Owner Group

d → Directory **r → Read** **w → Write** **x → eXecute**

Setting Permissions



```
alice@DESKTOP:~$ ls -l
drwxrwxrwx 0 alice alice    512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 alice alice     17 Jan 29 22:46 my_file
-rwxrwxrwx 1 alice  faculty 313 Jan 29 22:47 my_program.py
alice@DESKTOP:~$ chmod ugo-rwx my_dir
alice@DESKTOP:~$ chmod go-rwx my_program.py
alice@DESKTOP:~$ chmod u-rw my_program.py
alice@DESKTOP:~$ chmod +x my_file
alice@DESKTOP:~$ ls -l
d--------- 0 alice alice     512 Jan 29 22:46 my_dir
-rwxrwxrwx 1 alice alice      17 Jan 29 22:46 my_file
---x------ 1 alice faculty 313 Jan 29 22:47 my_program.py
```

Alternate Form of Setting Permissions

chmod ### <file1> [file2] …

- #s correspond to owner, group, and other
- Each value encodes read, write, and execute permissions
  - 1 → execute
  - 2 → write
  - 4 → read
- What if you want to set something as read, write, and execute?
  - 1 + 2 + 4 = 7

```
alice@DESKTOP:~$ ls -l
drwxrwxrwx 0 alice  alice     512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 alice  alice      17 Jan 29 22:46 my_file
-rwxrwxrwx 1 alice  faculty 313 Jan 29 22:47 my_program.py
alice@DESKTOP:~$ chmod 000 my_dir
alice@DESKTOP:~$ chmod 100 my_program.py
alice@DESKTOP:~$ chmod 777 my_file
alice@DESKTOP:~$ ls -l
d--------- 0 alice alice     512 Jan 29 22:46 my_dir
-rwxrwxrwx 1 alice alice      17 Jan 29 22:46 my_file
---x------ 1 alice  faculty 313 Jan 29 22:47 my_program.py
```

Who May Change Permissions?

```
alice@DESKTOP:~$ groups
alice faculty
alice@DESKTOP:~$ ls -l
-rw-rw-rw- 1 alice alice      17 Jan 29 22:46 my_file
-rw-rw-rw- 1 alice faculty  17 Jan 29 22:46 my_other_file
-rw------- 1 root  root    896 Jan 29 22:47 sensitive_data.csv
-rwxrwx--- 1 root  faculty 313 Jan 29 22:47 program.py
```

- Which files is user *alice* permitted to *chmod*?
  - Only owners can *chmod* files
  - *alice* can *chmod my_file* and *my_other_file*
  - Group membership doesn't grant *chmod* ability (cannot *chmod program.py*)

Setting Ownership

- Unix uses discretionary access control
  - New objects are owned by the subject that created them
- How can you modify the owner or group of an object?

chown <owner>:<group> <file1> [file2] …

Who May Change Ownership?

```
alice@DESKTOP:~$ groups
alice faculty
alice@DESKTOP:~$ ls -l
-rw-rw-rw- 1 alice alice    17 Jan 29 22:46 my_file
-rw-rw-rw- 1 alice faculty  17 Jan 29 22:46 my_other_file
-rw------- 1 root root    896 Jan 29 22:47 sensitive_data.csv
-rwxrwx--- 1 root faculty 313 Jan 29 22:47 program.py
```

- Which operations are permitted?

chown alice:faculty my_file

chown alice:alice sensitive_date.csv

chown alice:faculty program.py

Unix Access Control Exercise (1)

- What Unix group and permission assignments satisfy this access control matrix?

Desired Permissions

	file1	file2
user1	r--	rwx
user2	r--	rw-
user3	r--	rw-
user4	rwx	rw-

User	Groups
user1	user1
user2	user2
user3	user3
user4	user4

```
~$ ls -l
-rwxrwxr-- 1 user4  user4  0 file1
-rwxrwxrw- 1 user1  user1  0 file2
```

Unix Access Control Exercise (2)

- What Unix group and permission assignments satisfy this access control matrix?

Desired Permissions

	file1	file2
user1	r--	--x
user2	r-x	rwx
user3	r-x	r--
user4	rwx	r--

User	Groups
user1	user1
user2	user2, group1
user3	user3, group1, group2
user4	user4, group2

```
~$ ls -l
-rwxr-xr-- 1 user4  group1  0 file1
-rwxr----x 1 user2  group2  0 file2
```

Unix Access Control Exercise (3)

- What Unix group and permission assignments satisfy this access control matrix?

Desired Permissions

	file 1	file 2
user 1	---	rw-
user 2	r--	r--
user 3	rwx	rwx
user 4	rwx	---

- Trick question! This matrix **cannot** be represented
- *file2*: four distinct privilege levels
  - Maximum of three levels (user, group, other)
- *file1*: two users have high privileges
  - Owner has highest privilege
  - If select one of user 3 or user 4 as owner, need to support 4 privilege levels

Unix Access Control Review

The Good

- Very simple model
  - Owners, groups, and other
  - Read, write, execute
- Relatively simple to manage and understand

The Bad

- Not all policies can be encoded!
  - Contrast to ACL
- Not quite as simple as it seems
  - setuid