Security in Operating Systems: Basics
CS 111
Operating Systems
Peter Reiher

Security Goals

- Confidentiality
  - If it's supposed to be secret, be careful who hears it
- Integrity
  - Don't let someone change something they shouldn't
- Availability
  - Don't let someone stop others from using services
- Exclusivity
  - Don't let someone use something he shouldn't
- Note that we didn't mention "computers" here
  - This classification of security goals is very general

Access Control

- Security could be easy
  - If we didn't want anyone to get access to anything
- The trick is giving access to only the right people
- How do we ensure that a given resource can only be accessed by the proper people?
- The OS plays a major role in enforcing access control

Common Mechanisms for Access Control in Operating Systems

- Access control lists
  - Like a list of who gets to do something
- Capabilities
  - Like a ring of keys that open different doors
- They have different properties
- And are used by the OS in different ways

The Language of Access Control

- *Subjects* are active entities that want to gain access to something
  - E.g., users or programs
- *Objects* represent things that can be accessed
  - E.g., files, devices, database records
- *Access* is any form of interaction with an object
- An entity can be both subject and object

Access Control Lists

- ACLs
- For each protected object, maintain a single list
- Each list entry specifies a subject who can access the object
  - And the allowable modes of access
- When a subject requests access to a object, check the access control list

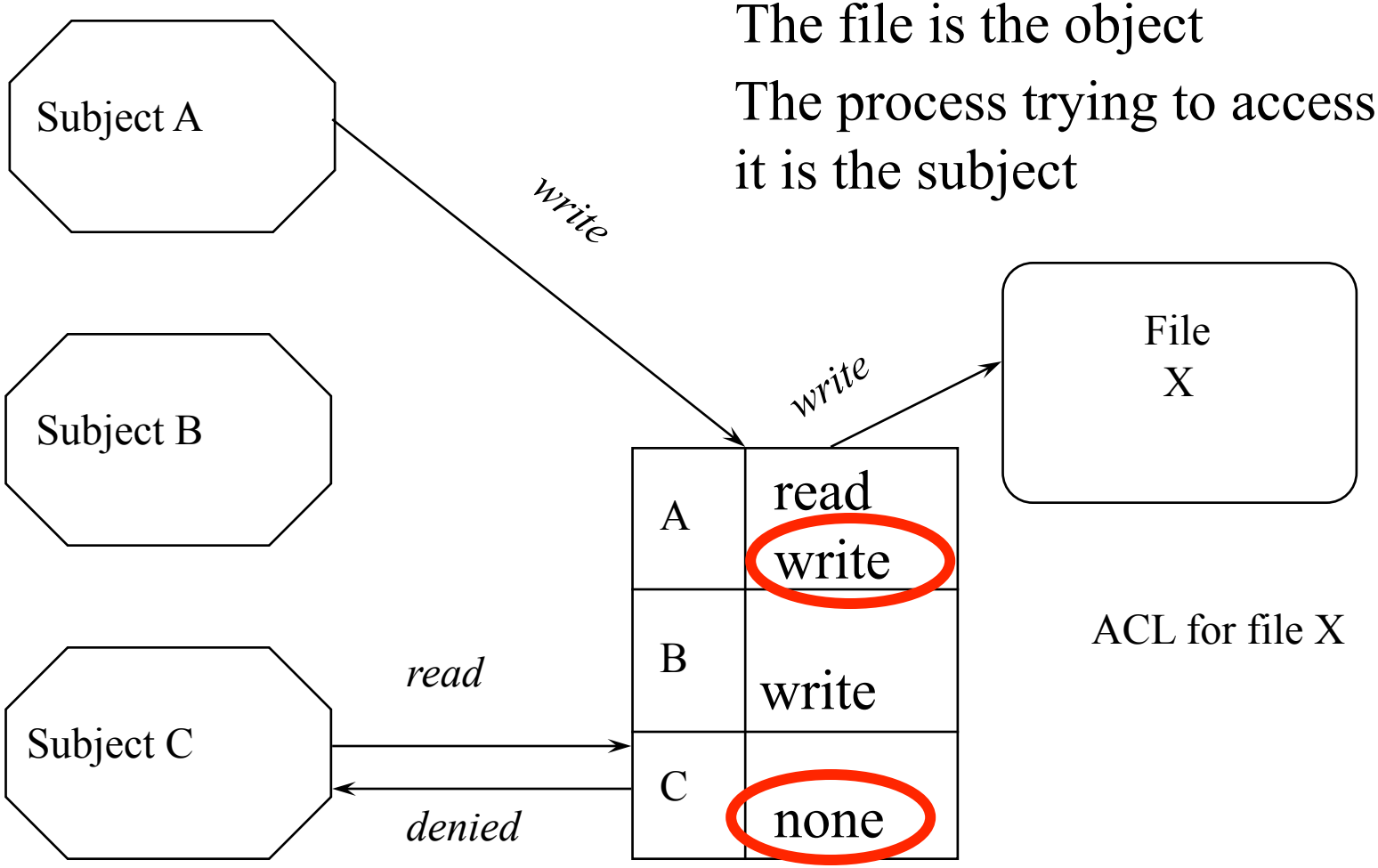
An Analogy

This is an access control list

An ACL Protecting a File

The file is the object

The process trying to access it is the subject



Issues For Access Control Lists

- How do you know the requestor is who he says he is?
- How do you protect the access control list from modification?
- How do you determine what resources a user can access?

An Example Use of ACLs: the Unix File System

- An ACL-based method for protecting files
  - Developed in the 1970s
- Still in very wide use today
  - With relatively few modifications
- Per-file ACLs (files are the objects)
- Three subjects on list for each file
    - Owner, group, other
- And three modes
  - Read, write, execute
  - Sometimes these have special meanings

Storing the ACLs

- They can be very small
  - Since there are only three entries
  - Basic ACL is only 9 bits
- Therefore, kept inside the file descriptor
- Makes it easy to find them
  - Since trying to open the file requires the file descriptor, anyway
- Checking this ACL is not much more than a logical AND with the requested access mode

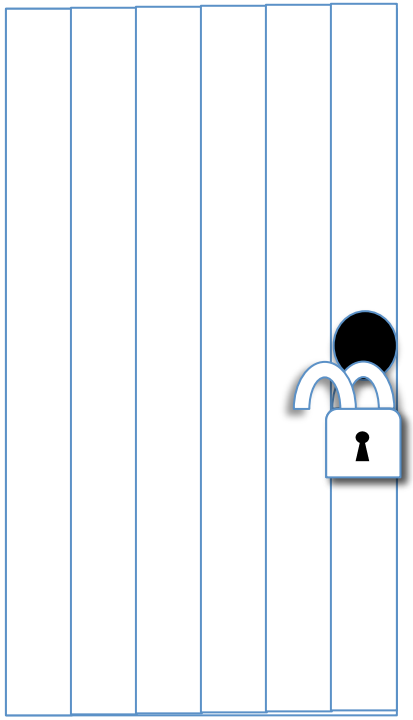
Pros and Cons of ACLs

+ Easy to figure out who can access a resource
+ Easy to revoke or change access permissions
– Hard to figure out what a subject can access
– Changing access rights requires getting to the object

Capabilities

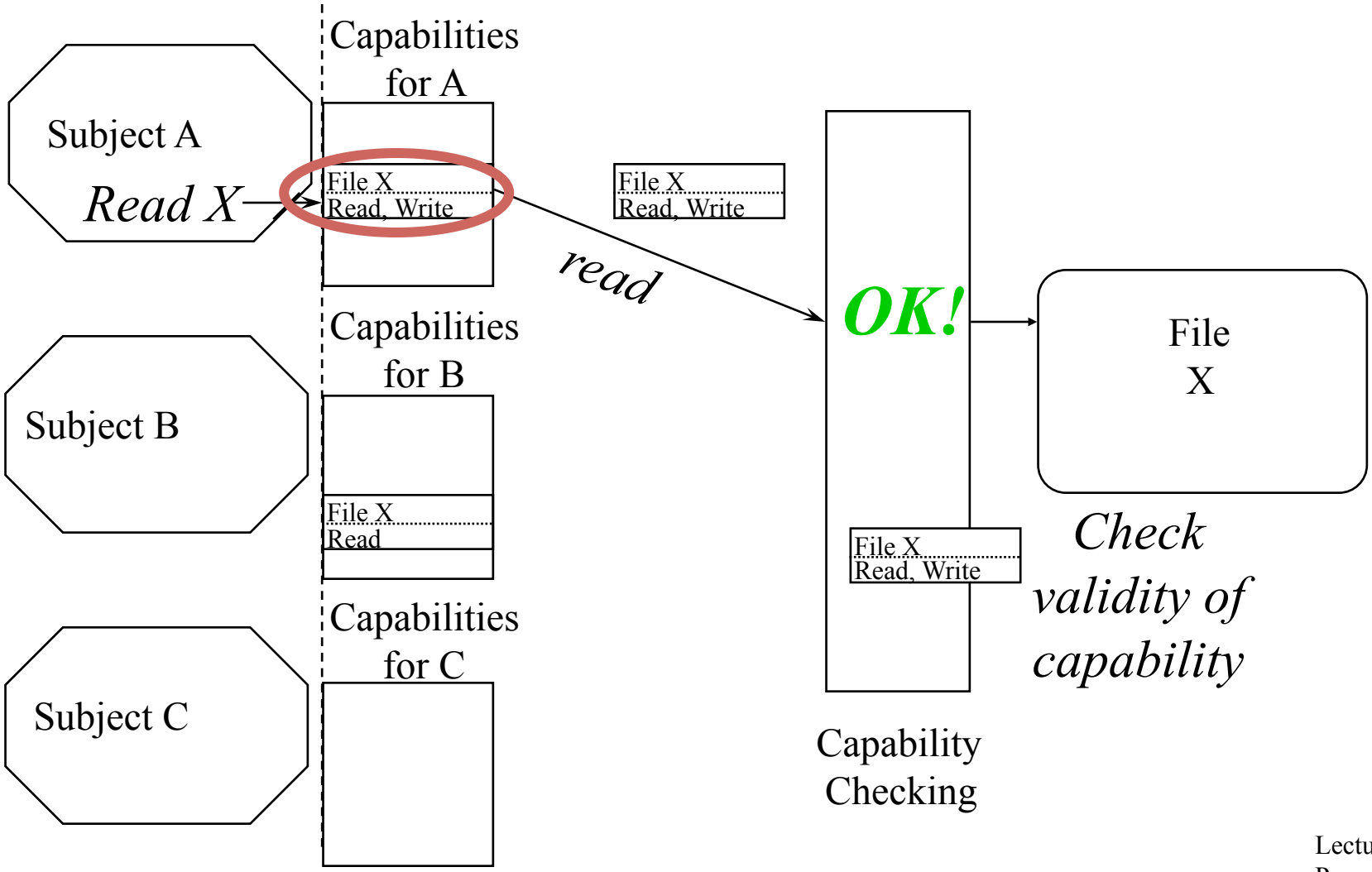
- Each subject keeps a set of data items that specify his allowable accesses
- Essentially, a set of tickets
- To access an object, present the proper capability
- Possession of the capability for an object implies that access is allowed

An Analogy

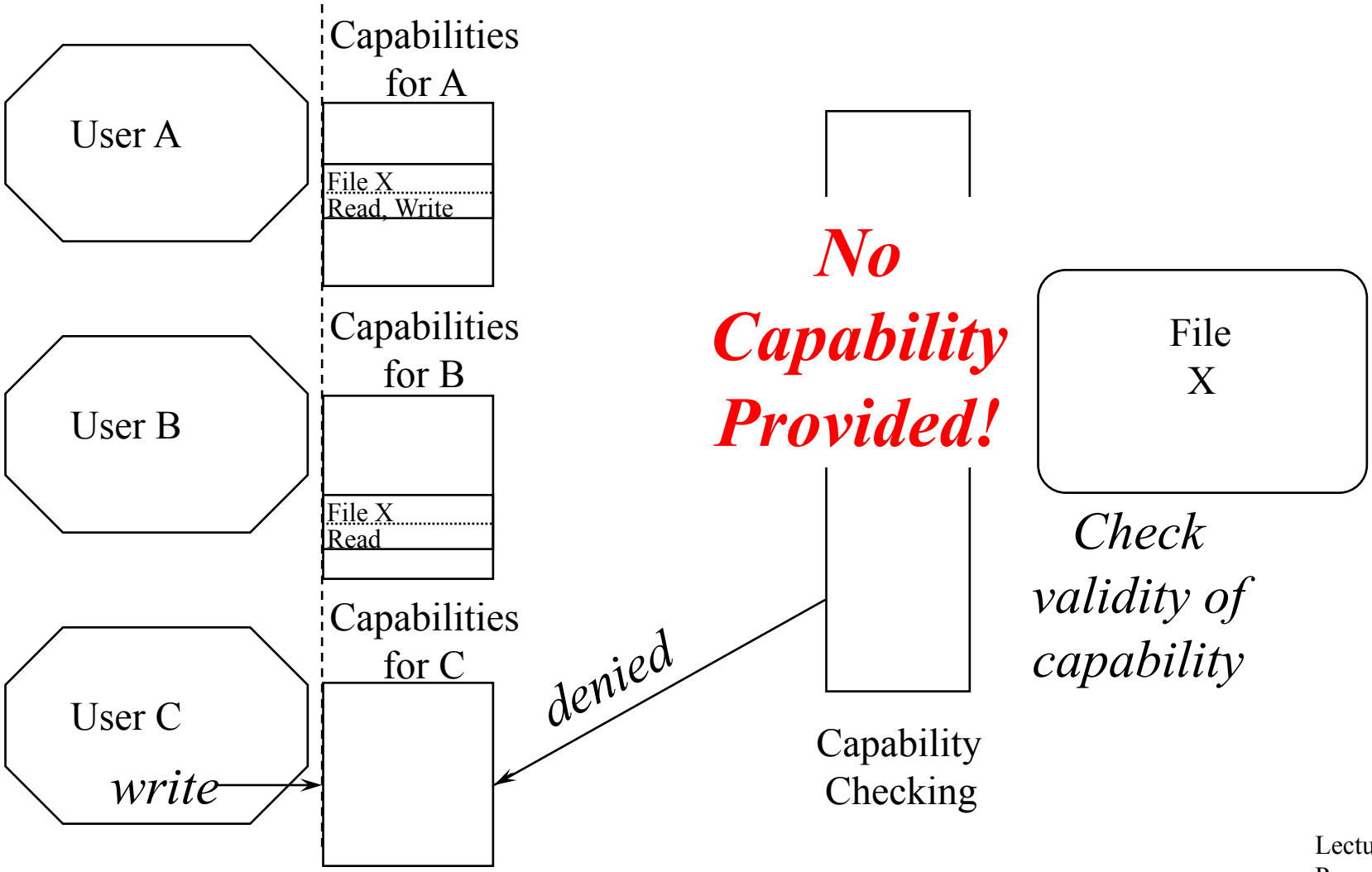


The key is a capability

Capabilities Protecting a File



Capabilities Denying Access



Properties of Capabilities

- Capabilities are essentially a data structure
  - Ultimately, just a collection of bits
- Merely possessing the capability grants access
  - So they must not be forgeable
- How do we ensure unforgeability for a collection of bits?
- One solution:
  - Don't let the user/process have them
  - Store them in the operating system

Revoking Capabilities

- A simple problem for capabilities stored in the operating system
  - Just have the OS get rid of it
- Much harder if it's not in the operating system
  - E.g., in a network context
- How do we make the bundle of bits change from valid to invalid?
- Consider the real world problem of a door lock
- If several people have the key, how do we keep one of them out?

Pros and Cons of Capabilities

+ Easy to determine what objects a subject can access
+ Potentially faster than ACLs (in some circumstances)
+ Easy model for transfer of privileges
– Hard to determine who can access an object
– Requires extra mechanism to allow revocation
– In network environment, need cryptographic methods to prevent forgery

OS Use of Access Control

- Operating systems often use both ACLs and capabilities
  - Sometimes for the same resource
- E.g., Unix/Linux uses ACLs for file opens
- That creates a file descriptor with a particular set of access rights
  - E.g., read-only
- The descriptor is essentially a capability

Enforcing Access in an OS

- Protected resources must be inaccessible
  - Hardware protection must be used to ensure this
  - So only the OS can make them accessible to a process
- To get access, issue request to resource manager
  - Resource manager consults access control policy data
- Access may be granted directly
  - Resource manager maps resource into process
- Access may be granted indirectly
  - Resource manager returns a "capability" to process

Direct Access To Resources

- OS checks access control on initial request
- If OK, OS maps it into a process' address space
  - The process manipulates resource with normal instructions
  - Examples: shared data segment or video frame buffer
- Advantages:
  - Access check is performed only once, at grant time
  - Very efficient, process can access resource directly
- Disadvantages:
  - Process may be able to corrupt the resource
  - Access revocation may be awkward
    - You've pulled part of a process' address space out from under it

Indirect Access To Resources

- Resource is not directly mapped into process
  - Process must issue service requests to use resource
  - Access control can be checked on each request
  - Examples: network and IPC connections
- Advantages:
  - Only resource manager actually touches resource
  - Resource manager can ensure integrity of resource
  - Access can be checked, blocked, revoked at any time
    - If revoked, system call can just return error code
- Disadvantages:
  - Overhead of system call every time resource is used

Protecting Operating Systems Resources

- How do we use these various tools to protect actual OS resources?
- Memory?
- Files?
- Devices?
- IPC?
- Secure booting

Protecting Memory

- Most modern operating systems provide strong memory protection
- Usually hardware-based
- Most commonly through use of page tables and paging hardware
- To remind you, addresses issued by programs translated by hardware to physical addresses
  - If page tables handled right, process can't even name other processes' memory

Protecting Files

- We've already discussed this
- Most file systems have a built-in access control model
- The OS must enforce it
- All file access done through system calls
- Which gives the OS a chance to enforce the access control policy
- Typically checked on open

A File Protection Vulnerability

- Unix/Linux systems only check access permissions on open
- The open file descriptor limits access to what was checked for
- But if the access permissions change while the file is open, access is NOT revoked
- Sometimes possible to keep files open for a long, long time
- So if user once had access to a file, may be hard to ever push him out again

Another File Data Vulnerability

- What if someone bypasses the operating system?
- Directly accessing the disk as a device
- The OS typically won't allow that to happen
  - If it's still in control . . .
- But there can be flaws or misconfigurations
- Or the disk can be moved to another machine
  - Which may not enforce the access permissions it specifies

Full Disk Encryption

- FDE
- A solution to this problem
- Encrypt everything you put on the disk
- Decrypt data moved from the disk to memory
- Can be done in hardware
  - Typically in the disk drive or controller
- Or software
  - Typically by the operating system
- Various options for storing the key

Protecting Devices

- Most devices are treated as files
- So the file protection model applies
- In some cases, some parts of the devices are memory mapped into processes
  - Memory protections apply, here
  - But potential issues if you map them into more than one process
- Non-OS controlled bus interfaces can also cause problems (e.g., Firewire)

Protecting IPC

- IPC channels are often also treated like files
- So the same protection model and mechanisms apply
- Even shared memory is handled this way
  - But especially important to remember that you don't get complete mediation here
  - And granularity of protection is the segment, not the word or page or block

Secure Boot

- Our OS-based protection mechanisms rely on one fundamental assumption
  - We are running an OS that properly implements them
- What if we aren't running the OS that we think we are?
- Then all bets are off
- The false OS can do whatever it wants
- So we need to be sure we've booted what we wanted to boot

The Bootstrap Process

- When a computer is powered on, the OS is not usually resident in memory
- It gets put there by a *bootstrap loader*
- The bootstrap program is usually very short
- Located in an easily defined place
- Hardware finds it, loads it, runs it
- Bootstrap then takes care of initializing the OS

Booting and Security

- Most systems make it hard to change bootstrap loader
  - But it must have enough flexibility to load different OSes
  - From different places on machine
- Malware likes to corrupt the bootstrap
- Trusted computing platforms can help secure bootstrapping

Approaches to Bootstrap Security

- TPM – an industry standard
- A hardware-assisted method to guarantee that the right bootstrap was loaded
  - And, from that, guarantee that the right OS was booted
  - And possibly build up further security from that
- SecureBoot – a Microsoft technology
- Built into the boot hardware and SW
- Essentially, only allows booting of particular OS versions