Operating System Security
CS 111
Operating Systems
Peter Reiher

Outline

- Basic concepts in computer security
- Design principles for security
- Important security tools for operating systems
- Access control
- Cryptography and operating systems
- Authentication and operating systems
- Protecting operating system resources

Security: Basic Concepts

- What do we mean by security?
- What is trust?
- Why is security a problem?
  - In particular, a problem with a different nature than, say, performance
  - Or even reliability

What Is Security?

- *Security* is a policy
  - E.g., "no unauthorized user may access this file"
- *Protection* is a mechanism
  - E.g., "the system checks user identity against access permissions"
- Protection mechanisms implement security policies
- We need to understand our goals to properly set our policies
  - And threats to achieving our goals
  - These factors drive which mechanisms we must use

Security Goals

- **C**onfidentiality
  - If it's supposed to be secret, be careful who hears it
- **I**ntegrity
  - Don't let someone change something they shouldn't
- **A**vailability
  - Don't let someone stop others from using services
- Note that we didn't mention "computers" here
  - This classification of security goals is very general

What Makes Security Hard?

- The "universe" we work in is more hostile
- Human opponents seek to outwit us
- Fundamentally, we want to share secrets in a controlled way
- You have to get everything right
  - Any mistake is an opportunity for your opponent
- Security costs, both performance and money

Tools For Securing Systems

- Physical security
- Access control
- Encryption
- Authentication
- Encapsulation
- Intrusion detection
- Filtering technologies

Physical Security

- Lock up your computer
  - Usually not sufficient, but . . .
  - Necessary (when possible)
- Networking means that attackers can get to it, anyway
- But lack of physical security often makes other measures pointless
  - A challenging issue for mobile computing

Access Control

- Only let authorized parties access the system
- A lot trickier than it sounds
- Particularly in a network environment
- Once data is outside your system, how can you continue to control it?
  - Again, of concern in network environments

Encryption

- Algorithms to hide the content of data or communications
- Only those knowing a secret can decrypt the protection
- Obvious value in maintaining secrecy
- But clever use can provide other important security properties
- One of the most important tools in computer security
  - But not a panacea

Authentication

- Methods of ensuring that someone is who they say they are
- Vital for access control
- But also vital for many other purposes
- Often (but not always) based on encryption
- Especially difficult in distributed environments

Encapsulation

- Methods of allowing outsiders limited access to your resources
- Let them use or access some things
  - But not everything
- Simple, in concept
- Extremely challenging, in practice
- Operating system often plays a large role, here

Intrusion Detection

- All security methods sometimes fail
- When they do, notice that something is wrong
- And take steps to correct the problem
- Reactive, not preventative
  - But unrealistic to believe any prevention is certain
- Must be automatic to be really useful

Filtering Technologies

- Detect that there's something bad:
  - In a data stream
  - In a file
  - Wherever
- Filter it out and only deliver "safe" stuff
- The basic idea behind firewalls
  - And many other approaches
- Serious issues with detecting the bad stuff and not dropping the good stuff

Operating Systems and Security Tools

- Physical security is usually assumed by OS
- Access control is key to OS technologies
- Encapsulation in various forms is widely provided by operating systems
- Some form of authentication required by OS
- Encryption is increasingly used by OS
- Intrusion detection and filtering not common parts of the OS

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

Goals for Access Control

- Complete mediation
- Least privilege
- Useful in a networked environment
- Scalability
- Cost and usability

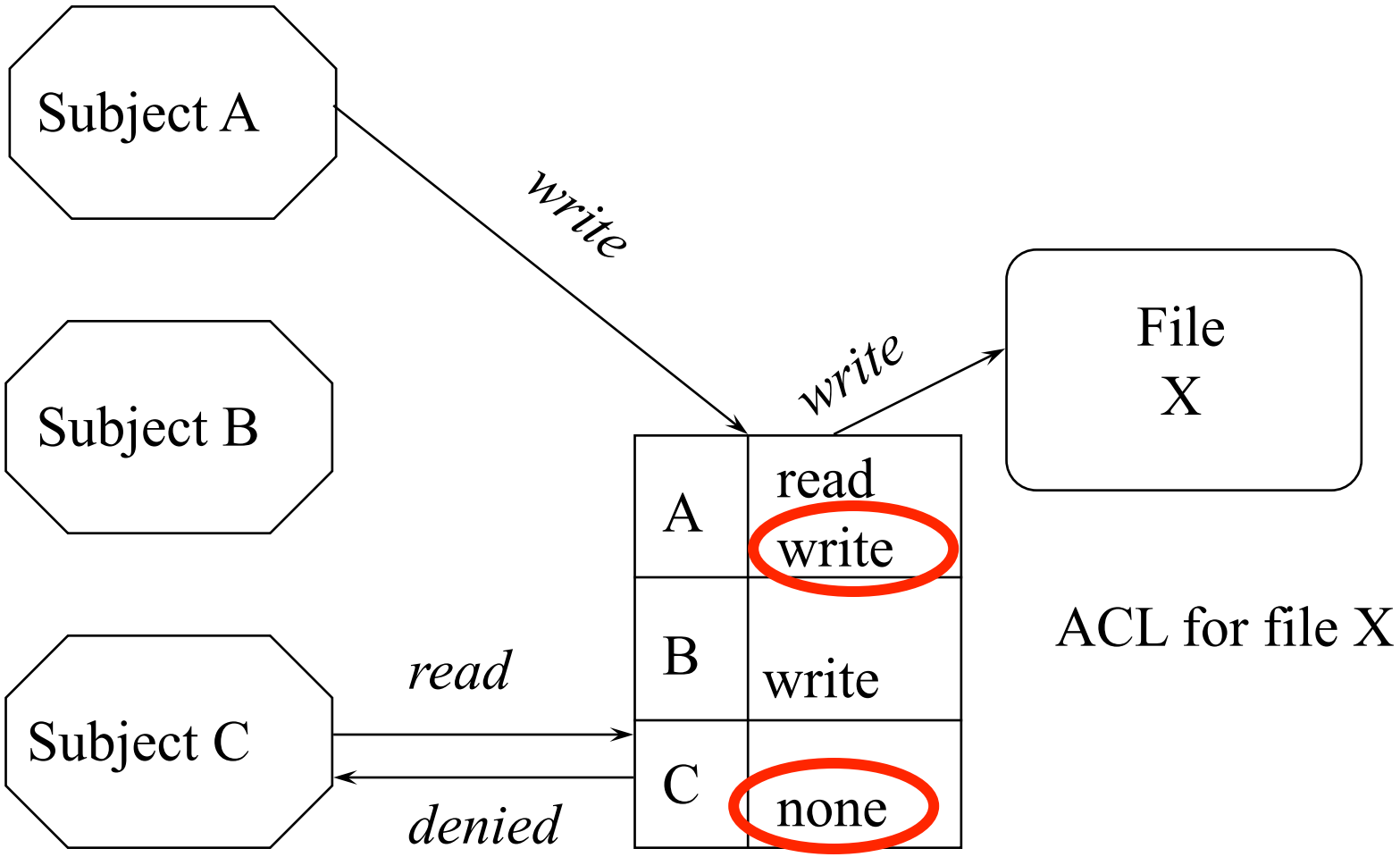
Common Mechanisms for Access Control in Operating Systems

- 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

Access Control Lists

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

An ACL Protecting a File



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?
- Costs associated with complete mediation

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

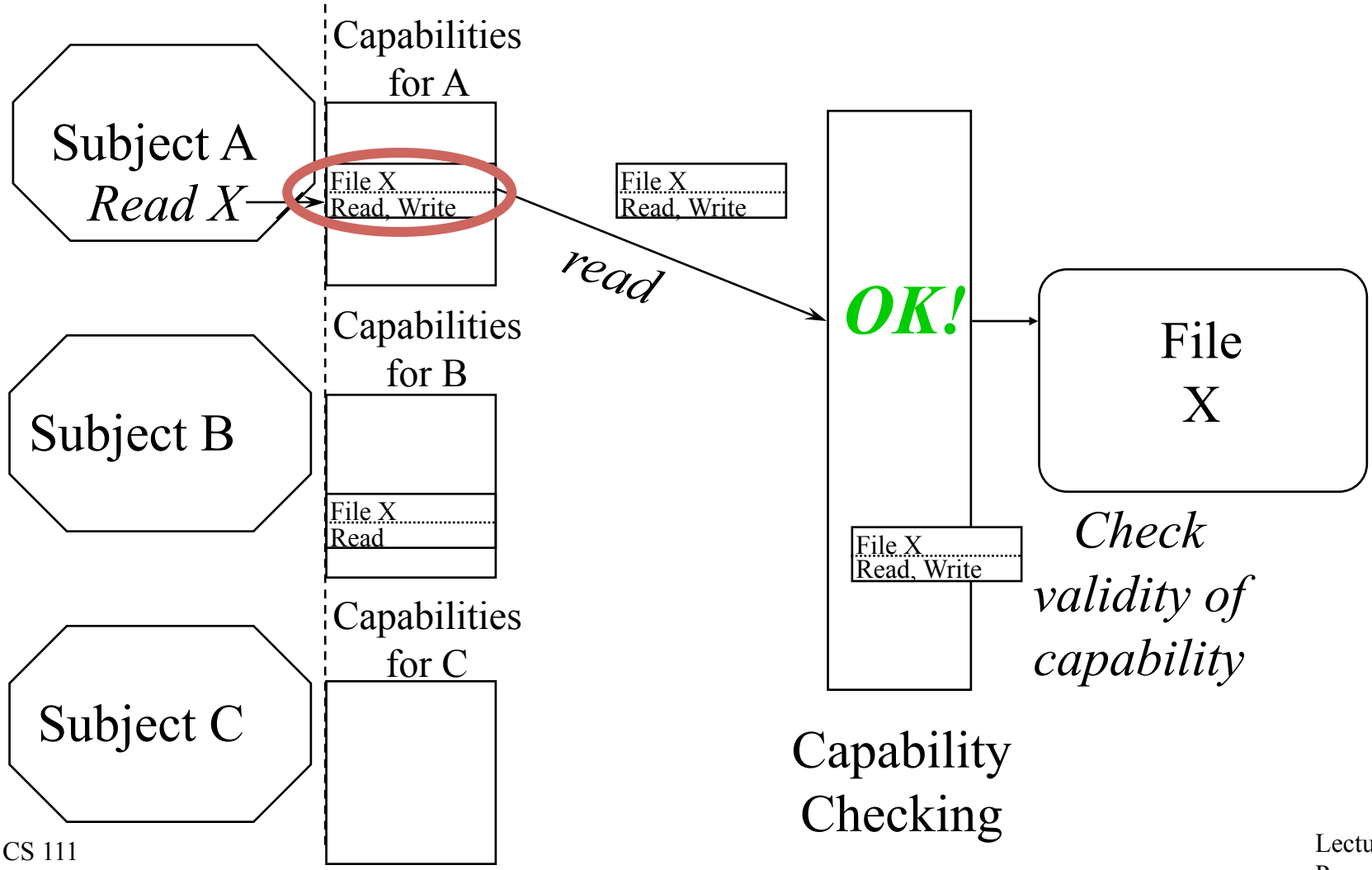
Pros and Cons of ACLs

+ 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

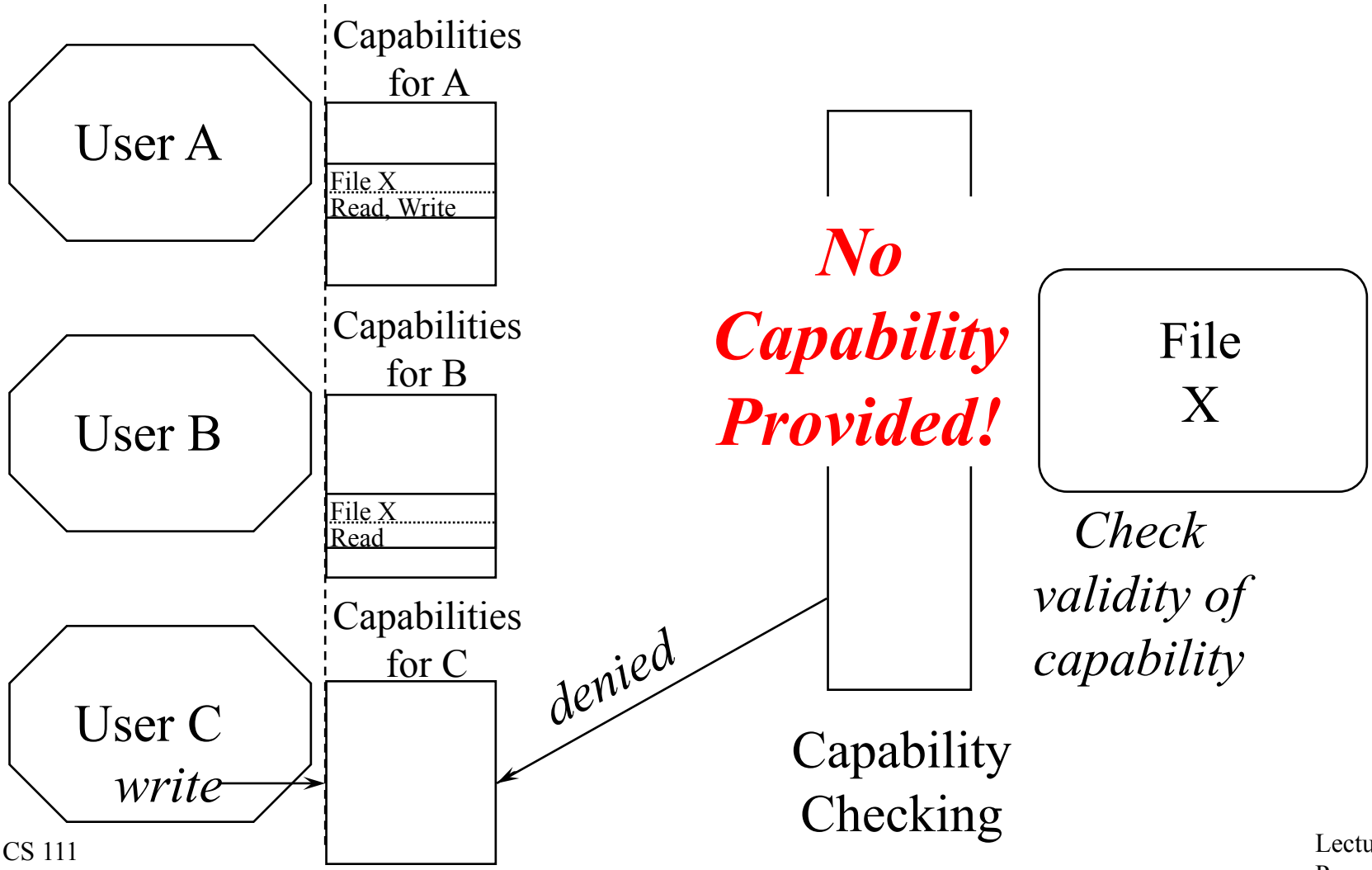
Capabilities

- Each entity 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

Capabilities Protecting a File



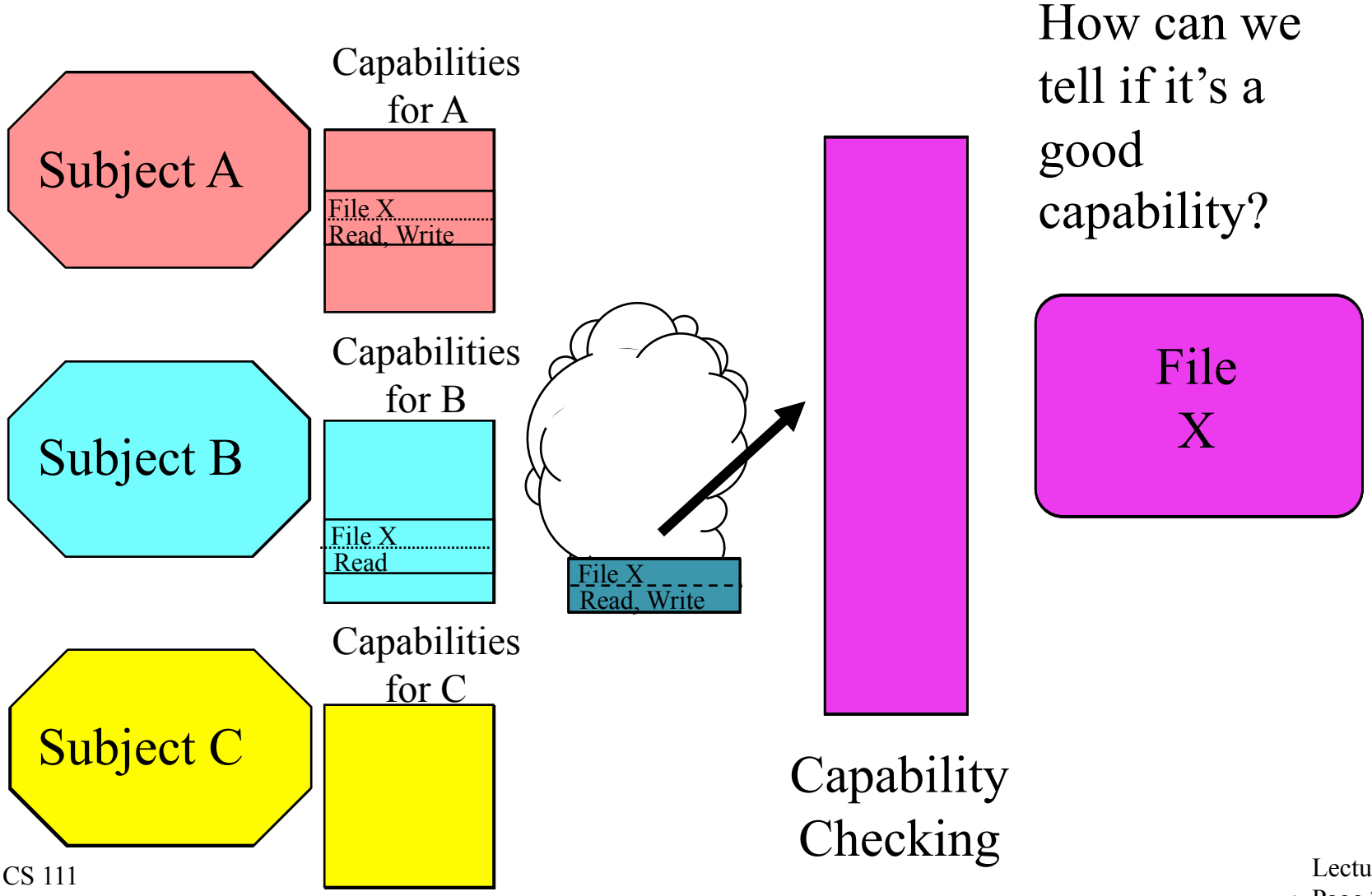
Capabilities Denying Access



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

Capabilities and Networks



Cryptographic Capabilities

- Create unforgeable capabilities by using cryptography
  - We'll discuss cryptography in detail in the next lecture
- Essentially, a user CANNOT create this capability for himself
- The examining entity can check the validity
- Prevents creation of capabilities from nothing
  - But doesn't prevent copying them

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
  - Making sure you catch every access

Cryptography

- Much of computer security is about keeping secrets
- One method of doing so is to make it hard for others to read the secrets
- While (usually) making it simple for authorized parties to read them
- That's what cryptography is all about

What Is Encryption?

- Encryption is the process of hiding information in plain sight
- Transform the secret data into something else
- Even if the attacker can see the transformed data, he can't understand the underlying secret
- Usually, someone you want to understand it can

Cryptography Terminology

- Typically described in terms of sending a message
  - Though it's used for many other purposes
- The sender is *S*
- The receiver is *R*
- *Encryption* is the process of making message unreadable/unalterable by anyone but *R*
- *Decryption* is the process of making the encrypted message readable by *R*
- A system performing these transformations is a *cryptosystem*
  - Rules for transformation sometimes called a *cipher*

Plaintext and Ciphertext

- *Plaintext* is the original form of the message (often referred to as *P*)

Transfer $100 to my savings account

- *Ciphertext* is the encrypted form of the message (often referred to as *C*)

Sqzmredq #099 sn lx rzuhmfr zbbntms

Cryptographic Keys

- Most cryptographic algorithms use a *key* to perform encryption and decryption
  - Referred to as *K*
- The key is a secret
- Without the key, decryption is hard
- With the key, decryption is easy
- Reduces the secrecy problem from your (long) message to the (short) key
  - But there's still a secret

More Terminology

- The encryption algorithm is referred to as *E()*
- $C = E(K,P)$
- The decryption algorithm is referred to as *D()*
- The decryption algorithm also has a key
- The combination of the two algorithms are often called a *cryptosystem*

Symmetric and Asymmetric Cryptosystems

- Symmetric cryptosystems use the same keys for E and D :

  $P = D(K, C)$

  – Expanding, $P = D(K, E(K,P))$

- Asymmetric cryptosystems use different keys for E and D:

  $C = E(K_E,P)$

  $P = D(K_D,C)$

  – Expanding, $P = D(K_D , E(K_E ,P))$

Symmetric Cryptosystems

- $C = E(K,P)$
- $P = D(K,C)$
- *E()* and *D()* are not necessarily the same operations
- Symmetric cryptosystems are relatively fast

Some Popular Symmetric Ciphers

- The Data Encryption Standard (DES)
  - The old US encryption standard
  - Still fairly widely used, due to legacy
  - Weak by modern standards
- The Advanced Encryption Standard (AES)
  - The current US encryption standard
  - Probably the most widely used cipher
- Blowfish
- There are many, many others

Symmetric Ciphers and Brute Force Attacks

- *Brute force attacks* – try every possible key until one works
- Cost depends on key length
  - Assuming random choice of key
  - For N possible keys, attack must try N/2 keys, on average, before finding the right one
- DES: 56 bit keys, too short for today
- AES: 128 or 256 bit keys, long enough

Asymmetric Cryptosystems

- Often called *public key cryptography*
  - Or PK, for short
- The encrypter and decrypter have different keys
  - $C = E(K_E, P)$
  - $P = D(K_D, C)$
- Often works the other way, too
  - $C' = E(K_D, P)$
  - $P = D(K_E, C')$

Using Public Key Cryptography

- Keys are created in pairs
- One key is kept secret by the owner
  - Authenticate with this one
  - Only owner could create the message
- The other is made public to the world
  - Protect messages with that one
  - Only owner has private key to decrypt
- Need both? Use two keys on same message

PK Key Management

- To communicate via shared key cryptography, key must be distributed
  - In trusted fashion
  - Either a *key distribution infrastructure*
  - Or use of *certificates*
- Both are problematic, at high scale and in the real world
- Bad PK key management == insecure systems

Example Public Key Ciphers

- RSA
  - The most popular public key algorithm
  - Used on pretty much everyone's computer, nowadays
- Elliptic curve cryptography
  - An alternative to RSA
  - Tends to have better performance
  - Not as widely used or studied

Combined Use of Symmetric and Asymmetric Cryptography

- Very common to use both in a single session
- Asymmetric cryptography essentially used to "bootstrap" symmetric crypto
- Use RSA (or another PK algorithm) to authenticate and establish a *session key*
- Use DES or AES with session key for the rest of the transmission

For Example,

Alice wants to share K_S only with Bob

Bob wants to be sure it's Alice's key

Only Bob can decrypt it

Only Alice could have created it

Alice

K_{EA} K_{DA}

K_{DB}

K_S

$C=E(K_S,K_{DB})$

$M=E(C,K_{EA})$

Bob

K_{EB} K_{DB}

K_{DA}

$K_S=D(C,K_{EB})$ $C=D(M,K_{DA})$

Authentication for Operating Systems

- Authentication is determining the identity of some entity
  - Process
  - Human user
- Requires notion of identity
  - One implication is we need some defined name space
- And some degree of proof of identity

Where Do We Use Authentication in the OS?

- Typically users authenticate themselves to the system
- Their identity tends to be tied to the processes they create
  - OS can keep track of this easily
- Once authenticated, users (and their processes) typically need not authenticate again
  - One authentication per session, usually

Authentication Mechanisms

- Something you know
  - E.g., passwords
- Something you have
  - E.g., smart cards or tokens
- Something you are
  - Biometrics

Passwords

- Authentication by what you know
- One of the oldest and most commonly used security mechanisms
- Authenticate the user by requiring him to produce a secret
  - Usually known only to him and to the authenticator

Problems With Passwords

- They have to be unguessable
  - Yet easy for people to remember
- If sent over the network, susceptible to password sniffers
- Unless fairly long, brute force attacks often work on them

Handling Passwords

- The OS must be able to check passwords when users log in
- So must the OS store passwords?
- Not really
  - It can store an encrypted version
- Encrypt the offered password
  - Using a *one-way function*
  - E.g., a secure hash algorithm like SHA1
- And compare it to the stored version

Authentication Devices

- Authentication by what you have
- A smart card or other hardware device that is readable by the computer
  - Safest if device has some computing capability
  - Rather than just data storage
- Authenticate by providing the device to the computer
- More challenging when done remotely, of course

Biometric Authentication

- Authentication based on who you are
- Things like fingerprints, voice patterns, retinal patterns, etc.
- To authenticate, allow the system to measure the appropriate physical characteristics
- Biometric measurement converted to binary and compared to stored values
  - With some level of match required

Problems With Biometrics

- Requires very special hardware
- May not be as foolproof as you think
- Many physical characteristics vary too much for practical use
  - Day to day or over long periods of time
- Generally not helpful for authenticating programs or roles
- What happens when it's cracked?
  - You only have two retinas, after all

Protecting Memory

- Most modern operating systems provide strong memory protection
- Usually hardware-based
- Most commonly through use of page tables and paging hardware
- Each process can only access page frames mapped in its own page table
- Reduces issue to OS' proper use of page tables for processes

Protecting Files

- 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 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