Security Principles, Policies, and Tools

CS 136

Computer Security

Peter Reiher

October 7, 2014

Outline

- Security design principles
- Security policies
  - Basic concepts
  - Security policies for real systems
- Classes of security tools
  - Access control

Design Principles for Secure Systems

- Economy
- Complete mediation
- Open design
- Separation of privileges
- Least privilege
- Least common mechanism
- Acceptability
- Fail-safe defaults

Economy in Security Design

- Economical to develop
  - And to use
  - And to verify
- Should add little or no overhead
- Should do only what needs to be done
- Generally, try to keep it simple and small

Complete Mediation

- Apply security on every access to a protected object
  - E.g., each read of a file, not just the open
- Also involves checking access on everything that could be attacked

Open Design

- Don’t rely on “security through obscurity”
- Assume all potential attackers know everything about the design
  – And completely understand it
- This doesn’t necessarily mean publishing everything important about your security system
  – Though sometimes that’s a good idea
- Obscurity can provide *some* security, but it’s brittle
  – When the fog is cleared, the security disappears
  – And modern attackers have good fog blowers

Separation of Privileges

- Provide mechanisms that separate the privileges used for one purpose from those used for another
- To allow flexibility in security systems
- E.g., separate access control on each file

Least Privilege

- Give bare minimum access rights required to complete a task
- Require another request to perform another type of access
- E.g., don't give write permission to a file if the program only asked for read

Least Common Mechanism

- Avoid sharing parts of the security mechanism
  - Among different users
  - Among different parts of the system
- Coupling leads to possible security breaches

Acceptability

- Mechanism must be simple to use
- Simple enough that people will use it without thinking about it
- Must rarely or never prevent permissible accesses

Fail-Safe Designs

- Default to lack of access
- So if something goes wrong or is forgotten or isn't done, no security lost
- If important mistakes are made, you'll find out about them
  - Without loss of security
  - But if it happens too often . . .

Security Policies

- Security policies describe how a secure system should behave
- Policy says what should happen, not how you achieve that
- Generally, if you don't have a clear policy, you don't have a secure system
  - Since you don't really know what you're trying to do

Informal Security Policies

- "Users should only be able to access their own files, in most cases."
- "Only authorized users should be able to log in."
- "System executables should only be altered by system administrators."
- The general idea is pretty clear
- But it can be hard to determine if a system meets these goals

Formal Security Policies

- Typically expressed in a mathematical security policy language
- Tending towards precision
  - Allowing formal reasoning about the system and policy
- Often matched to a particular policy model
  - E.g., Bell-La Padula model
- Hard to express many sensible policies in formal ways
  - And hard to reason about them usefully

Some Important Security Policies

- Bell-La Padula
- Biba integrity policy

Bell-La Padula Model

- Probably best-known computer security model
- Corresponds to military classifications
- Combines mandatory and discretionary access control
- Two parts:
  - Clearances
  - Classifications

Clearances

- Subjects (people, programs, etc.) have a *clearance*
- Clearance describes how trusted the subject is
- E.g., *unclassified*, *confidential*, *secret*, *top secret*

Classifications

- Each object (file, database entry, etc.) has a *classification*
- The classification describes how sensitive the object is
- Using same categories as clearances
- Informally, only people with the same (or higher) clearance should be able to access objects of a particular classification

Goal of Bell-La Padula Model

- Prevent any subject from ever getting read access to data at higher classification levels than subject's clearance
  - I.e., don't let untrusted people see your secrets
- Concerned not just with objects
- Also concerned with the objects' contents
- Includes discretionary access control
  - Which we won't cover in lecture

Bell-La Padula Simple Security Condition

- *Subject S can read object O iff* $l_O \leq l_S$
- Simple enough:
  - If S isn't granted top secret clearance, S can't read top secret objects
- Are we done?

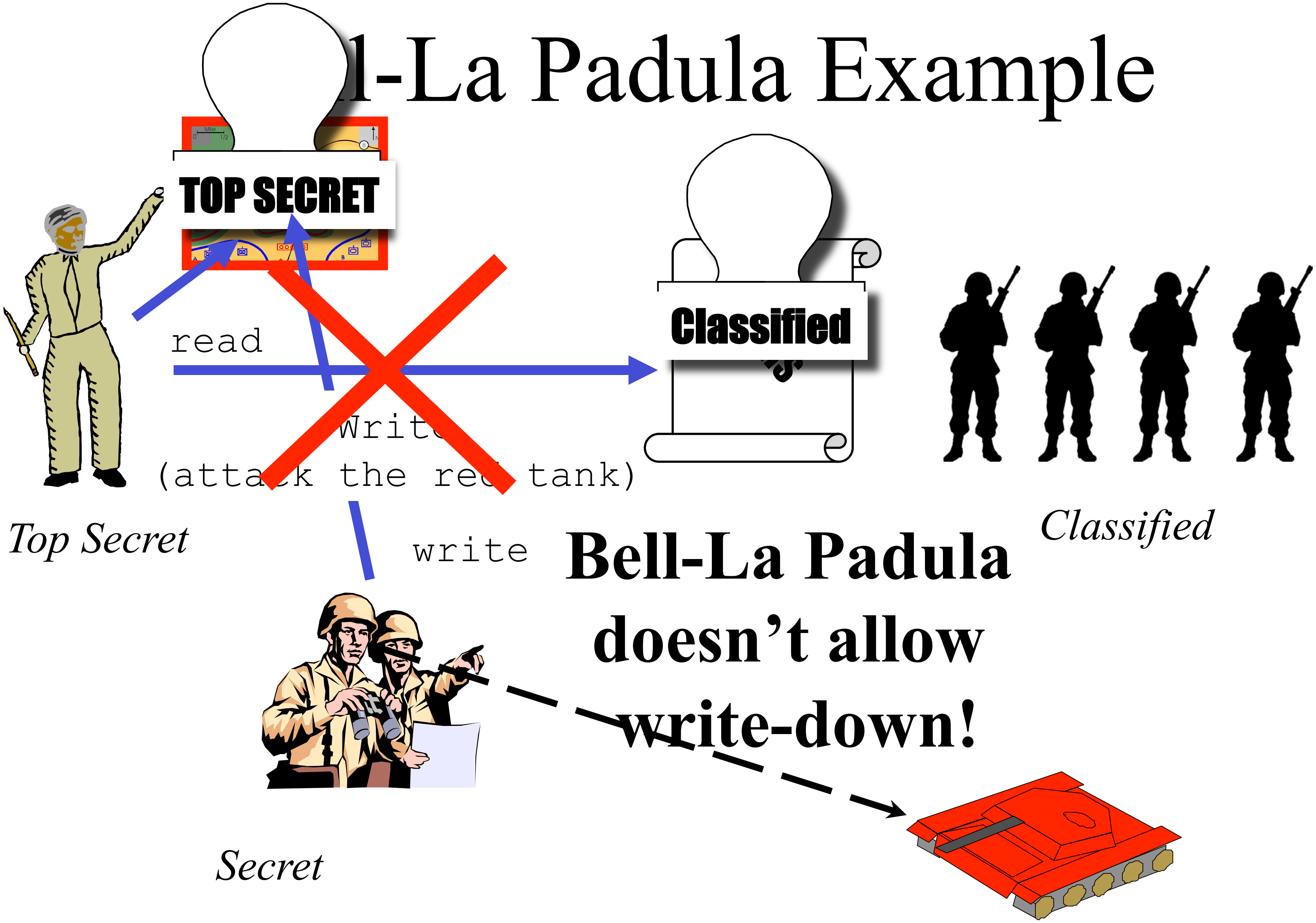
Why Aren't We Done?

- Remember, we really care about the information in an object
- A subject with top secret clearance can read a top secret object
- If careless, he could write that information to a confidential object
- Then someone with confidential clearance can read top secret information

The Bell-La Padula *-Property

- *S can write O iff* $l_S \leq l_O$
- Prevents *write-down*
  – Privileged subjects writing high-classification information to low-classification objects
  – E.g., a top secret user can't write to a confidential data file
- Can be proven that a system meeting these properties is "secure"

l-La Padula Example



Bell-La Padula doesn't allow write-down!

So How Do You Really Use The System?

- There have to be mechanisms for reclassification
  - Usually requiring explicit operation
- Danger that reclassification process will be done incautiously
- Real systems also use classes of information

Integrity Security Policies

- Designed to ensure that information is not improperly changed
- Often the key issue for commercial systems
- Secrecy is nice, but not losing track of your inventory is crucial

Example: Biba Integrity Policy

- Subject set S, object set O
- Set of ordered integrity levels I
- Subjects and objects have integrity levels
- Subjects at high integrity levels are less likely to screw up data
  - E.g., trusted users or carefully audited programs
- Data at a high integrity level is less likely to be screwed up
  - Probably because it badly needs not to be screwed up

Biba Integrity Policy Rules

- s can write to o iff $i(o) \leq i(s)$
- s_1 can execute s_2 iff $i(s_2) \leq i(s_1)$
- A subject s can read object o iff $i(s) \leq i(o)$
- Why do we need the read rule?

Hybrid Models

- Sometimes the issue is keeping things carefully separated
- E.g., a brokerage that handles accounts for several competing businesses
- Microsoft might not like the same analyst working on their account and IBM's
- There are issues of both confidentiality and integrity here
- Example – Chinese Wall model

The Realities of Discretionary Access Control

- Most users never change the defaults on anything
  - Unless the defaults prevent them from doing something they want to do
- Most users don't think about or understand access control
- Probably not wise to rely on it to protect information you care about
  - Unless you're the one setting it
  - And you know what you're doing

The Problems With Security Policies

- Hard to define properly
  - How do you determine what to allow and disallow?
- Hard to go from policy to the mechanisms that actually implement it
- Hard to understand implications of policy
- Defining and implementing policies is a lot of work

Tools for Security

- Physical security
- Access control
- Encryption
- Authentication
- Encapsulation
- Intrusion detection
- Common sense

Physical Security

- Lock up your computer
  - Actually, sometimes a good answer
- But what about networking?
  - Networks poke a hole in the locked door
- Hard to prevent legitimate holder of a computer from using it as he wants
  - E.g., smart phone jailbreaks
- In any case, lack of physical security often makes other measures pointless

Access Controls

- 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
- One of the most important tools in computer security
  - But not a panacea
- Covered in more detail later in class

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

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

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 it's unrealistic to believe any prevention is certain
- Must be automatic to be really useful

Common Sense

- A lot of problems arise because people don't like to think
- The best security tools generally fail if people use them badly
- If the easiest way in is to fool people, that's what attackers will do

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
  - And at the right time and circumstances
- How do we ensure that a given resource can only be accessed when it should be?

Goals for Access Control

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

Access Control Mechanisms

- Access control lists
- Capabilities
- Access control matrices
  - Theoretical concept we won't discuss in detail
- Role based access control

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

Mandatory vs. Discretionary Access Control

- Mandatory access control is dictated by the underlying system
  - Individual users can’t override it
  - Even for their own data
- Discretionary access control is under command of the user
  - System enforces what they choose
  - More common than mandatory

Access Control Lists

- For each protected resource, maintain a single list
- Each list entry specifies a user who can access the resource
  - And the allowable modes of access
- When a user requests access to a resource, check the access control list (ACL)

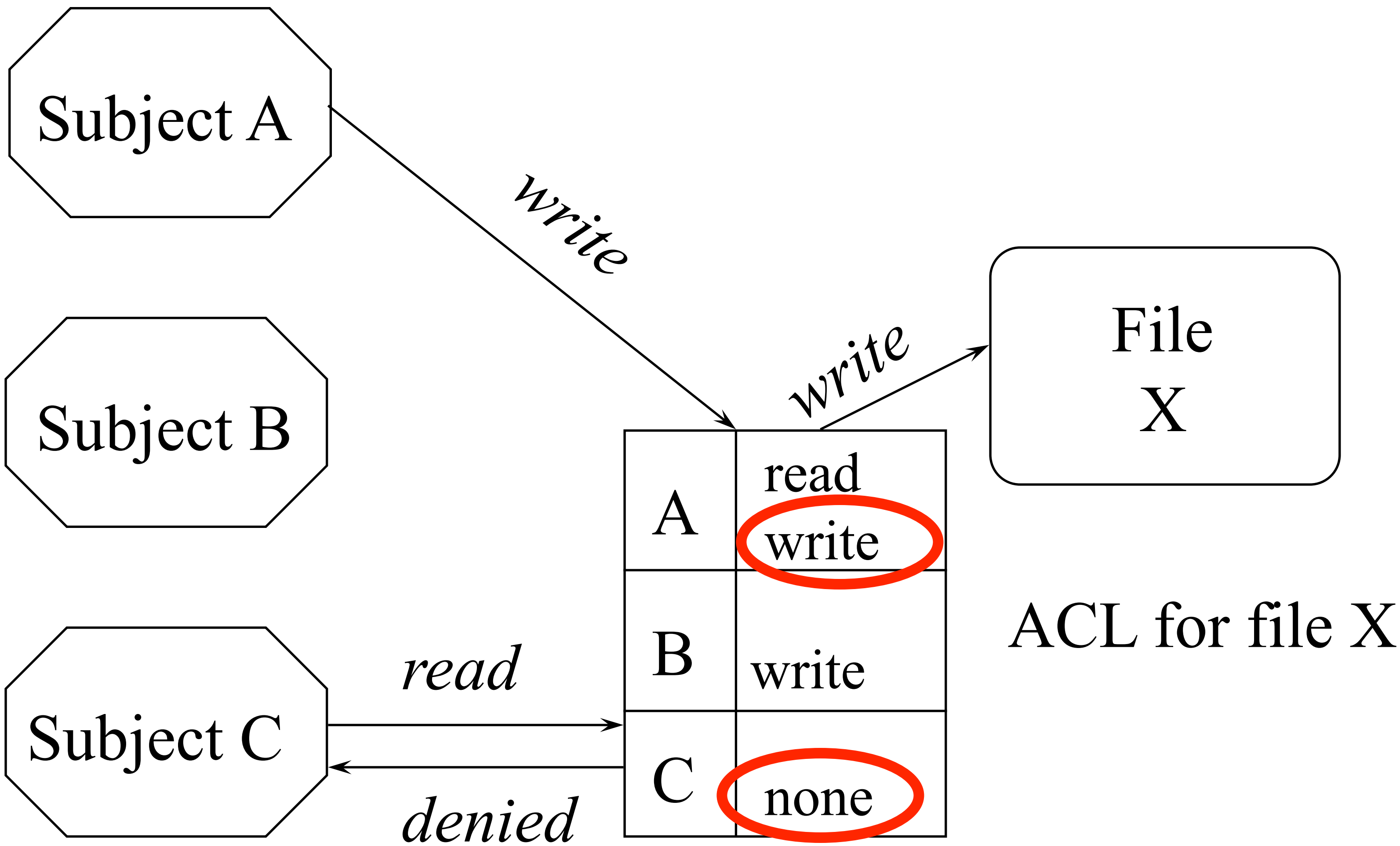
ACL Objects and Subjects

- In ACL terminology, the resources being protected are *objects*
- The entities attempting to access them are *subjects*
  - Allowing finer granularity of control than per-user

ACL Example

- An operating system example:
  - Using ACLs to protect a file
- User (Subject) A is allowed to read and write to the file
- User (Subject) B may only read from it
- User (Subject) C may not access it

An ACL Protecting a File



Issues for Access Control Lists

- How do you know that 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?
- Generally issues for OS design

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

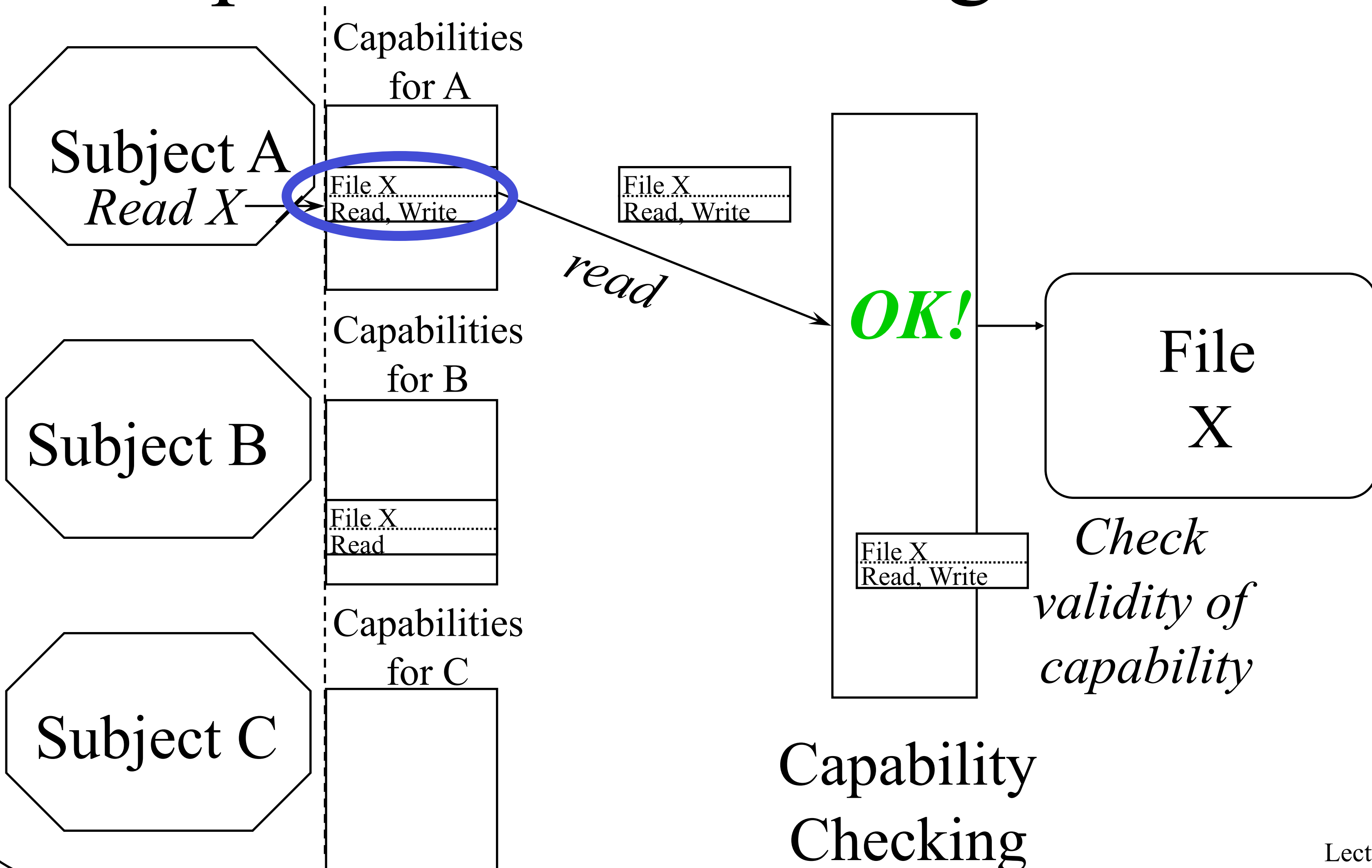
Capabilities

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

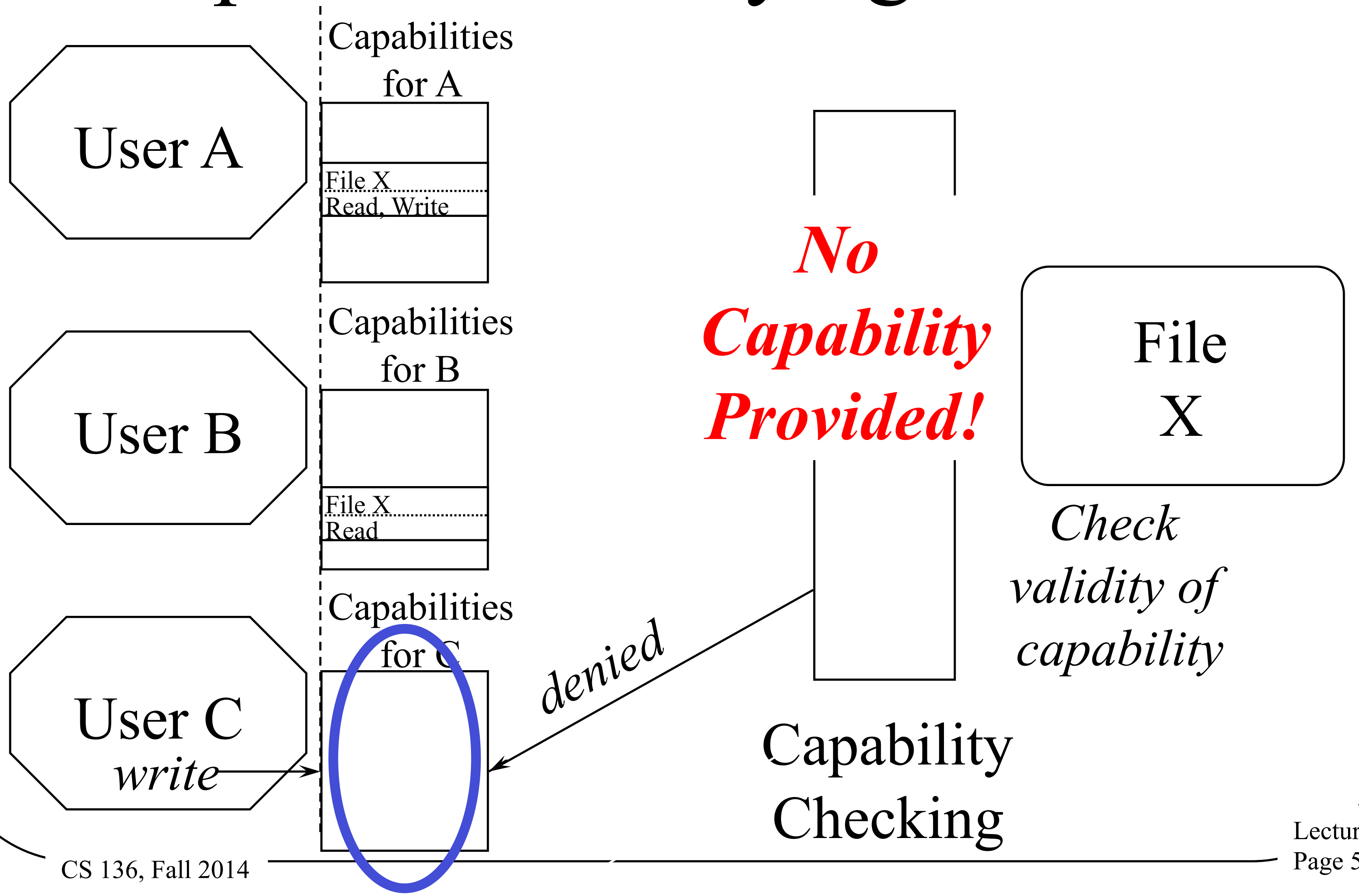
Properties of Capabilities

- Must be unforgeable
  - In single machine, keep capabilities under control of OS
  - What about in a networked system?
- In most systems, some capabilities allow creation of other capabilities
  - Process can pass a restricted set of capabilities to a subprocess

Capabilities Protecting a File

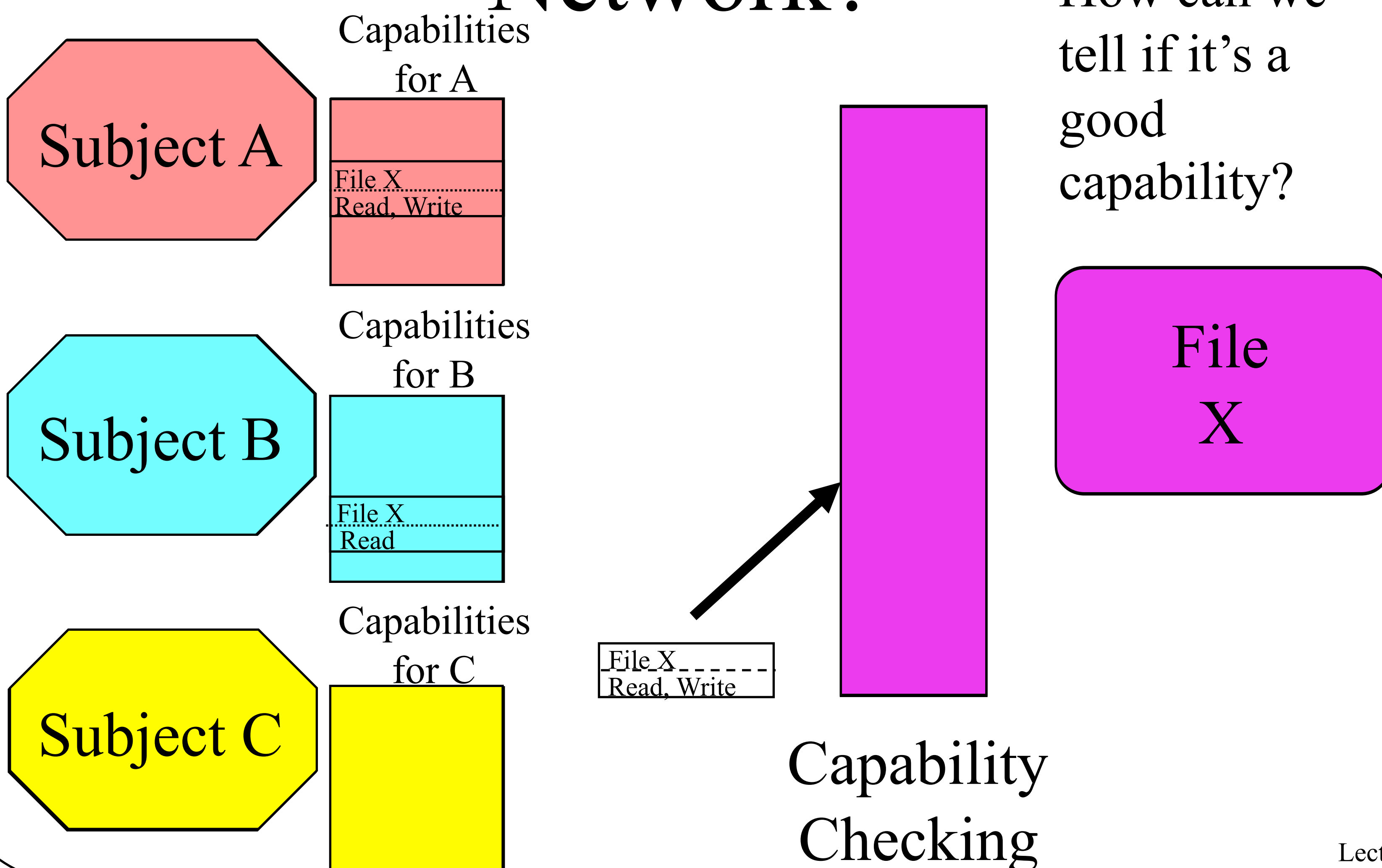


Capabilities Denying Access



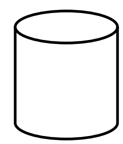
How Will This Work in a Network?

How can we tell if it's a good capability?



Revoking Capabilities

Fred



`Accounts receivable`

How do we take away Fred's capability?

Nancy

Without taking away Nancy's?

Options for Revoking Capabilities

- Destroy the capability
  - How do you find it?
- Revoke on use
  - Requires checking on use
- Generation numbers
  - Requires updating non-revoked capabilities

Pros and Cons of Capabilities

+ Easy to determine what 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

Distributed Access Control

- ACLs still work OK
  - Provided you have a global namespace for subjects
  - And no one can masquerade
- Capabilities are more problematic
  - Security relies on unforgeability
  - Provided by cryptographic methods
  - Prevents forging, not copying

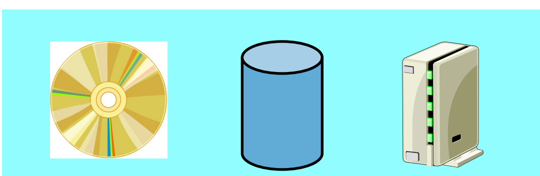
Role Based Access Control

- An enhancement to ACLs or capabilities
- Each user has certain roles he can take while using the system
- At any given time, the user is performing a certain role
- Give the user access to only those things that are required to fulfill that role
- Available in some form in most modern operating systems

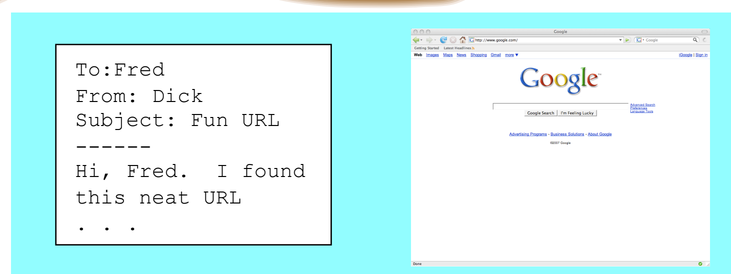
A Simple Example

Fred is a system administrator

But Fred is a also a normal user



Fred should operate under one role while doing system administration



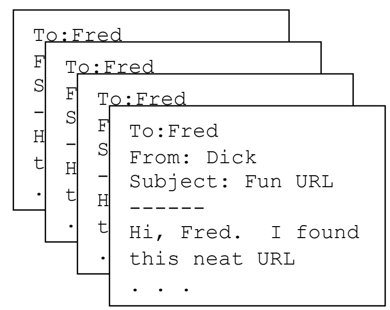
And another role while doing normal stuff

Continuing With Our Example

He decides to upgrade the C++ compiler

Fred logs on as “fred”

He reads his email



So he changes his role to “sysadmin”

Then he has the privileges to upgrade the compiler

But may have lost the privileges to read “fred’s” email

Result: Evil malware in fred’s email can’t “upgrade” the compiler

Changing Roles

- Role based access control only helps if changing roles isn't trivial
  – Otherwise, the malicious code merely changes roles before doing anything else
- Typically requires providing some secure form of authentication
  – Which proves you have the right to change roles
  – Usually passwords, but other methods possible

Practical Limitations on Role Based Access Control

- Number of roles per user
- Problems of disjoint role privileges
- System administration overheads
- Generally, these cause usability and management problems

Android Access Control

- Android is a software development environment for mobile devices
  - Especially phones
- An open platform that allows adding arbitrary applications
  - Written by many different parties
- What's the appropriate access control model?

The Android Access Control Model

- Linux ACLs at the bottom
  - If that were all, apps would run with permissions of user who ran them
- Above that, access control specific for Android
- Each application runs as its own Linux user
  - But how to handle interactions between apps?
- Access to other apps' components handled by Intercomponent Communications (ICC) controls

ICC Access Control

- Built into Android stack
  - So Android apps use it, but no regular app does
- ICC reference monitor provides a form of mandatory access control
- Android apps are built of components
  - Each app component has an access label
- Developer assigns a set of access labels to an app
  - Some for components in their own app
  - Some for components of other apps
  - Set defines an application's access domain

What Does This Mean?

- Application developer limits what his application can do
  - Even if compromised, it can't do more
  - Permissions settable only at app installation
- Developer can also limit who else can use his components
  - Preventing data leakage, for example

Some Advantages of This Approach

- Limits power of applications
- Allows those installing applications to know what they can access
- Centralizes information about access permissions
  - Extensions limit that somewhat

Reference Monitors

- Whatever form it takes, access control must be instantiated in actual code
  - Which checks if a given attempt to reference an object should be allowed
- That code is called a *reference monitor*
- Obviously, good reference monitors are critical for system security

Desirable Properties of Reference Monitors

- Correctness
- Proper placement
- Efficiency
- Simplicity
- Flexibility