Security Principles, Policies, and Tools
CS 136
Computer Security
Peter Reiher
March 31, 2016

Some of the different designs to ensure good security

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 — the amt of effort you have to put in for your system; don't want there to be a high cost of entry
- Complete mediation — you always want to check on user for every action/access
- Open design — assume the attack knows every single detail of how your security works; don't focus on obscurity, focus on good security design
- Separation of privileges — separate the privilege needed to do action x from the privilege to do action y; each privilege correspond to one specific action
- Least privilege — give the least amt of privilege needed to get something done
- Least common mechanism — spread out mechanism for a user; if too many users have the same mechanism, it can allow attackers to exploit weakness
- Acceptability — UI/UX bullshit
- Fail-safe defaults — when given no condition, default to no access

Economy in Security Design

cheap to develop; cheap to use (no high cost of any time at runtime, or high cost of barrier of entry).

- Economical to develop
  - And to use
  - And to verify

cheap to verify what we build
essentially invisible when protecting us

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

Complete Mediation

definition: some actions should be able to be performed, some should not; we make a complete independent check every time a user tries to do something (read file, access remote server)

- 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

closed design: not ideal, since attackers can still hack into the system

Open Design

- Don't rely on "security through obscurity"
- Assume all potential attackers know everything about the design
  - more secure that attackers know the design that we make in terms of security
  - And completely understand it
- This doesn't mean publish 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

ex. common cryptographic algorithms, like AES, has their algorithm online; everyone agreed that it achieves a high level of security, even though everyone knows how it works

ex of NOT doing this: DOS - pure access to the system
step up: old version of Windows - login with password, but once you access, you have complete access to the resources

KEY: separation of the privileges

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

Don't give complete privilege to do everything;
ex. Linux, you have permission to open the file for read, but not necessarily for write privilege

Least Privilege

what access rights, what security privileges are there for a given file

- 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

What should be done
ex. User A has access rights 'x'
User B has access rights 'y'

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

Coupling: sharing access rights among different users

users take the path of least resistance; clearly, users will take the simple action, which is ignoring the problem

Acceptability

Security system should not get in the way of people's work

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

Security needs to be accepted by users: ex. someone ppl have bad experiences with AVG, so they never use it again

Fail-Safe Designs

This is an important aspect of Security Design

- 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

  we want to be alarmed when there are mistakes that are made

  – Without loss of security
  – But if it happens too often . . .

    firewall doesn't allow anything to be let in by default
    you only specify what to be allowed in

Thinking About Security

When considering the security of any system, ask these questions:

if you are regularly getting denied of access, then you run into acceptability when users don't want to use your systems

1. What assets are you trying to protect?
2. What are the risks to those assets?

some of these principles work against each other, so be careful of design

3. How well does the security solution mitigate those risks?
4. What other security problems does the security solution cause?
5. What tradeoffs does the security solution require?

(This set of questions was developed by Bruce Schneier, for his book *Beyond Fear*)

ex. strong encryption is not enough to address all the problems

end goal, but not the actual algorithm

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

problem with informal policies: did that mechanism achieve the security policy; hard to say YES or NO

Informal Security Policies

just examples of informal security policies
seem relatively obvious

- "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

Bell-La Padula model - ensuring secrecy of data in a particular class of systems

- Hard to express many sensible policies in formal ways
  - And hard to reason about them usefully

Some Important Security Policies

- Bell-La Padula — read down; write up; confidentiality
- Biba integrity policy — read up; write down; integrity

Biba ex. company manager has more integrity trust than a new hire
manager doesn't trust new hire, so manager should not be reading anything from new hire
manager, however, can read from CEO
manager can write to new hire, because new hire can read manager's writing

Bell-La Padula Model

from military use: some information more sensitive than other information
people are trusted to different degrees
very very sensitive information are only seen by trusted people

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

Clearances

programs, remote systems, each of them have a clearance, how much information do they have

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

Classifications

a unclassified clearance means someone who can't view some resource

- 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

  care about content of object, not just object themselves

  – 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: classification of object (l_o) <= clearance of subject (l_s)
  - If S isn't granted top secret clearance, S can't read top secret objects
- Are we done? no!

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 confidential object - lower clearance
- Then someone with confidential clearance can read top secret information

someone with higher clearance can accidently release information to others

The Bell-La Padula *-Property

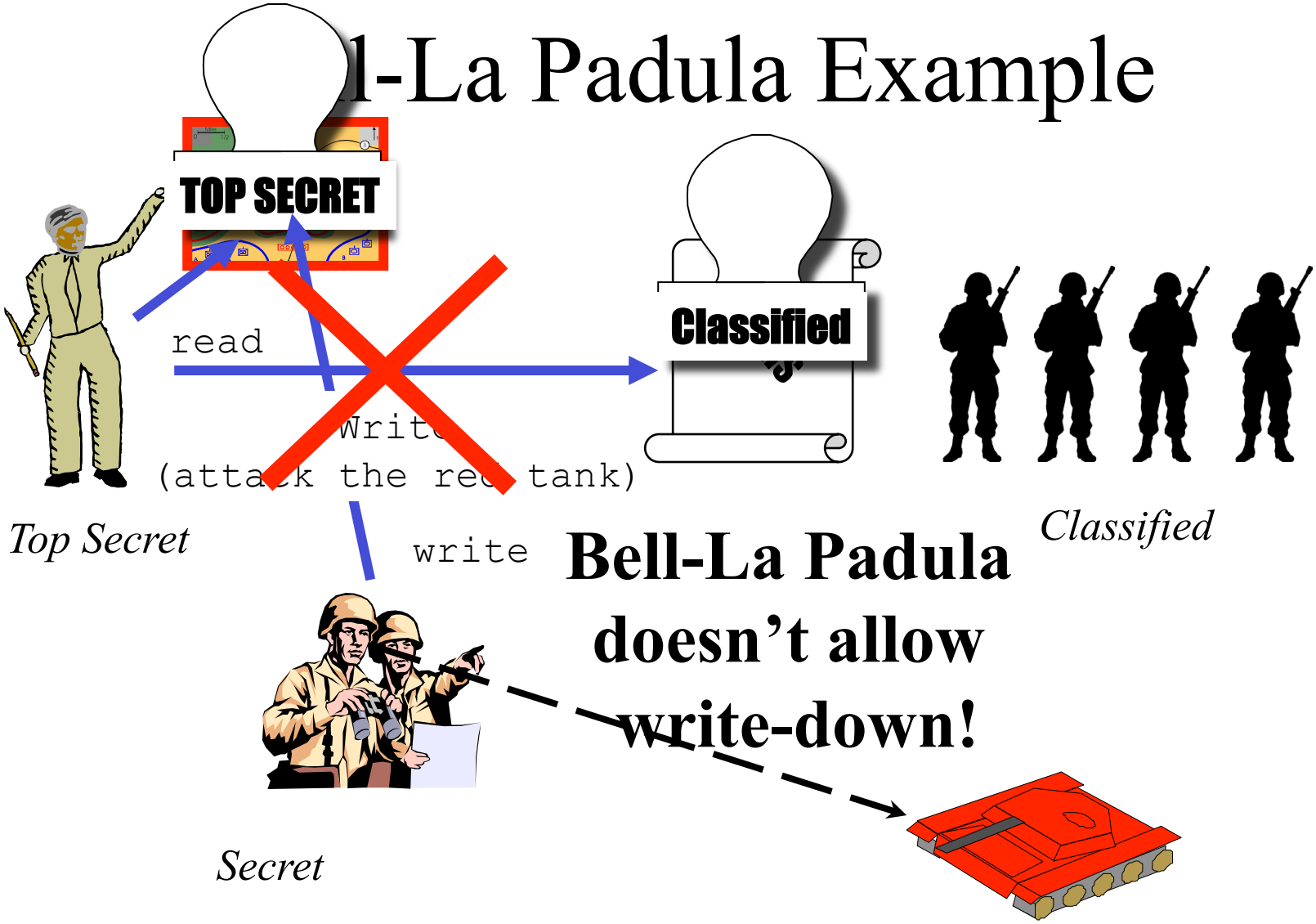
subject write to object
if clearance of subject <= classification of object

- *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"

ex. general can't write to lower classified data because general is top secret
problem: general then cannot give any orders to any group

Bell - read down + write up

l-La Padula Example



So How Do You Really Use The System?

- There have to be mechanisms for reclassification

  general can require a 'explicit operation' to change the classification, but you lost the bella padula guarantee

  - 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

integrity is also important for database

Bella Padula doesn't work for integrity

read up + write down

Example: Biba Integrity Policy

trusted users or programs that don't screw up

- 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 write to o if integrity level of o <= integrity level of subject

- 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?

  read rule allows someone of integrity i to only read objects that have a higher integrity

Chinese Wall Model: if you work for consulting firms, you have companies of conflicting interest. Follow a set of rules to guarantee that information from one subset is not leaked to another subset.

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

individual users decide what to be done

- 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

users don't really think about access control

The Problems With Security Policies

problems that exist in the real world (in industry)

- 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 — physically lock up a computer; problem: most computers have network connected
- Access control — ACL - only let authorized people access certain things
- Encryption — algorithms that hide data or communications - confidentiality
- Authentication — verifying that the user is who they say they are
  username and password, 2 factor authentication
- Encapsulation — allowing outsiders limited access to resources. ex. virtual machine
  probably exercise least privilege
- Intrusion detection — detect if there's any attacks; problem: doesn't fix error
- Common sense — smell bullshit and don't do stupid stuff

in the past 20 or 30 years, most computers are attached to networks, so attackers can still get through to the computer by using the network

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

a mobile device has low degree of physical security

- 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

important tool in computer & network security

- 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 integrity and authentication applications as well
  - But not a panacea
- Covered in more detail later in class

Authentication

authentication implements access control

- 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

limited access to some resources, but not all
ex. virtual machines

- 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

ex. a firewall or anti-virus
virus-scanning computer

- All security methods sometimes fail
- When they do, notice that something is wrong
  reactive mechanism - it doesn't take care of the problem, but it only detects it & allows you to detect
- 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

ex. running AVG in the background

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

social engineering…

Access Control

allow some people to get some access under some circumstances

- 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 only what is required to perform that action
- 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

a more theoretical concept

- Role based access control

a different way of looking at the problem

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

ex. a program is running, it is a subject
ex. a program responding to users, it is an object

Mandatory vs. Discretionary Access Control

- Mandatory access control is dictated by the underlying system ex. sudo, root user
  - Individual users can't override it
  - Even for their own data
- Discretionary access control is under command of the user user can use sudo, ex. mac osx you are the only user on the mac
  - System enforces what they choose
  - More common than mandatory

Access Control Lists

keep a list that describe everyone who can access that resource

- 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)

when some entity access obj, figure out his mode of access, goto ACL, and check if he is allowed to do this

ACL Objects and Subjects

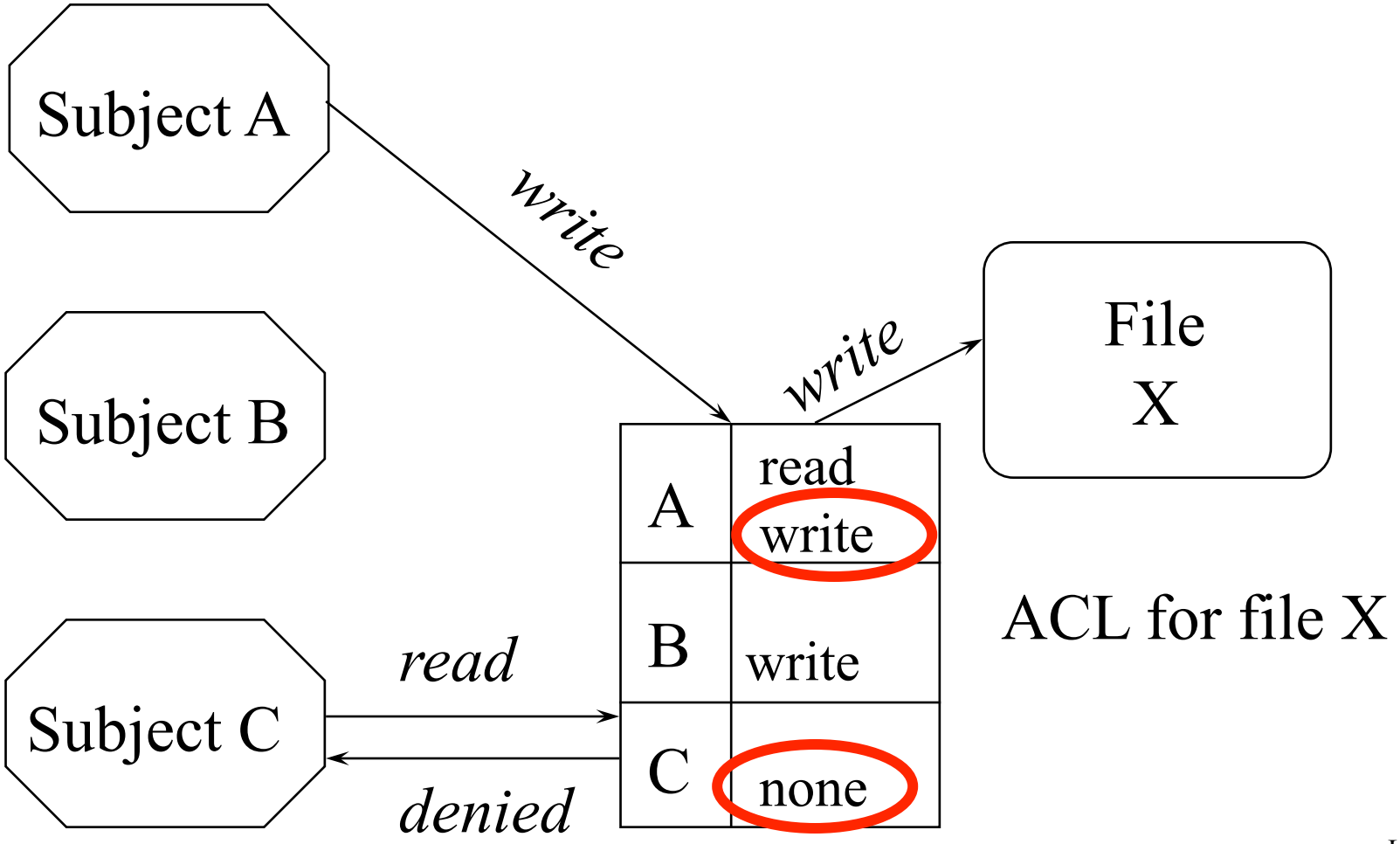
- In ACL terminology, the resources being protected are *objects*
- The entities attempting to access them are *subjects*
  - Allowing finer granularity of control than per-user

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

ex. Linux chmod can change your read/write/execute policy

An ACL Protecting a File



Issues for Access Control Lists

- How do you know that the requestor is who he says he is? ACL doesn't check for identity
- How do you protect the access control list from modification? someone could change the ACL
- How do you determine what resources a user can access? we will need to go through every single files and see which ACL has someone's name in it
- 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

don't have these lists associated with every objects

- Each subject keeps a set of data items that specify his allowable accesses

a user has tickets to get into different doors

- Essentially, a set of tickets
- Possession of the capability for an object implies that access is allowed

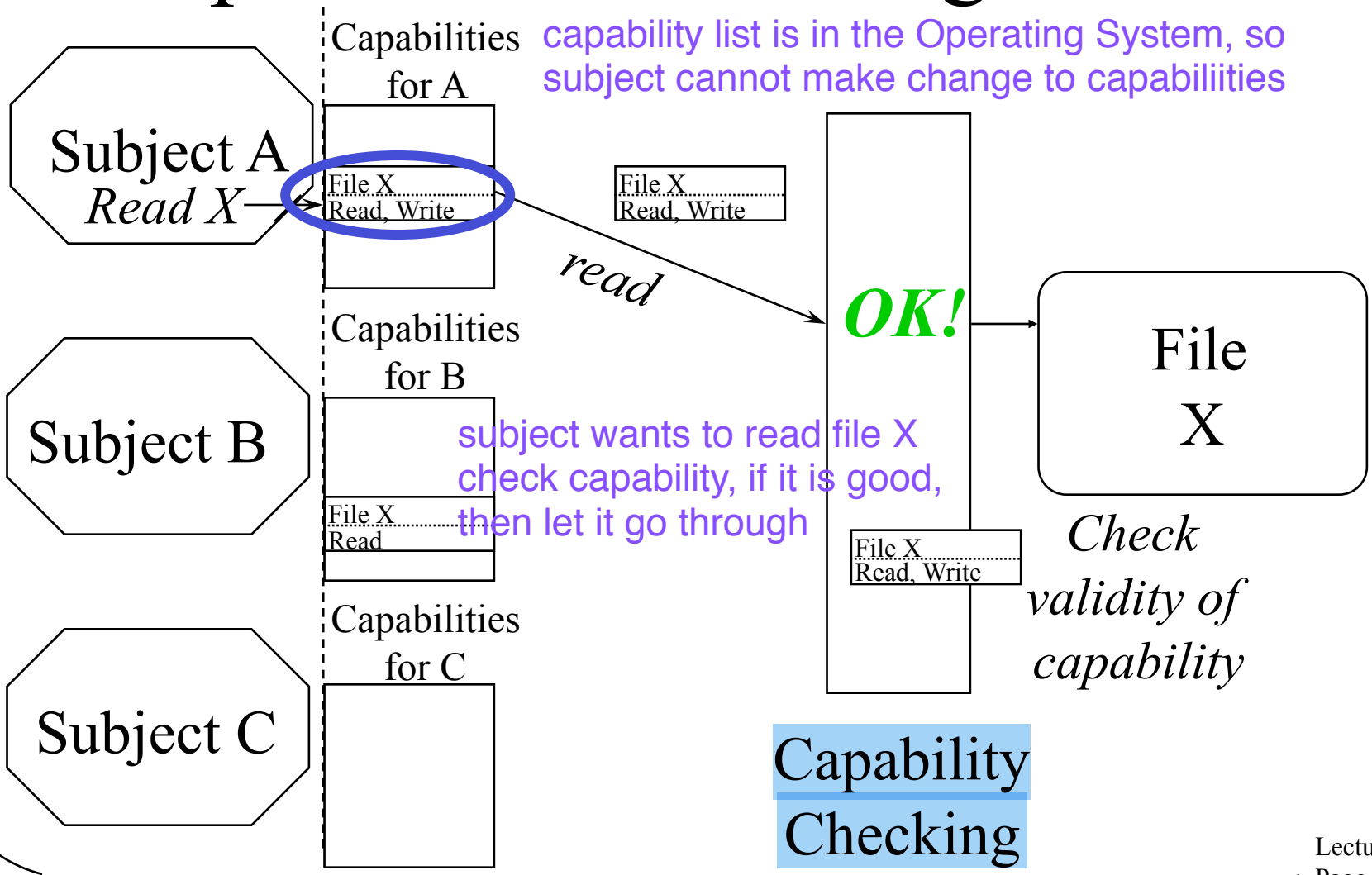
must be unable to create the pattern that makes the capabilities; basically, do not give users or programs the power to create capabilities list

Properties of Capabilities

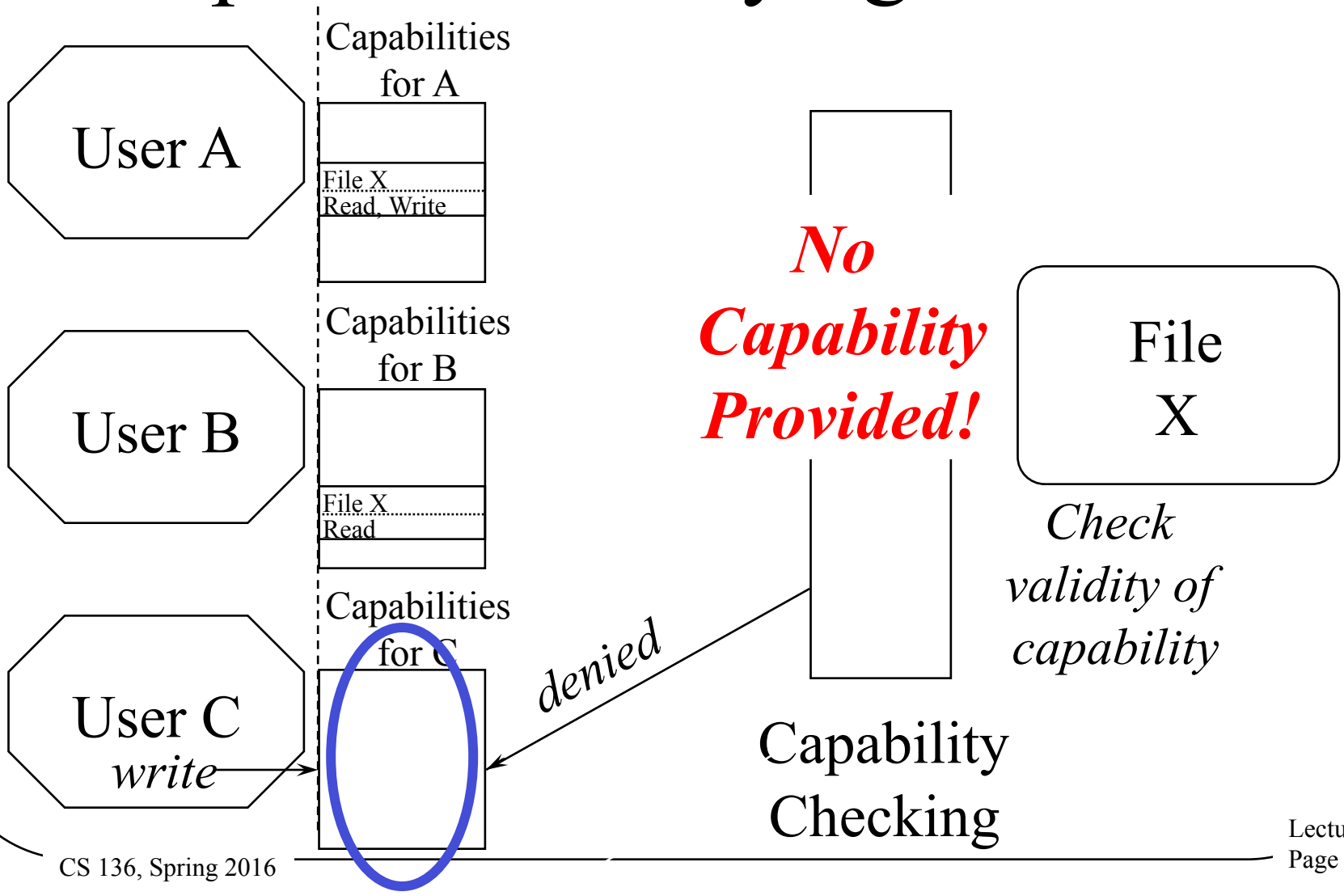
- Must be unforgeable
  - In single machine, keep capabilities under control of OS
  - What about in a networked system?
- In most systems, some capabilities allow creation of other capabilities
  - Process can pass a restricted set of capabilities to a subprocess

process can only do things it has capabilities for —> encapsulating the environment of what a process can do

Capabilities Protecting a File



Capabilities Denying Access

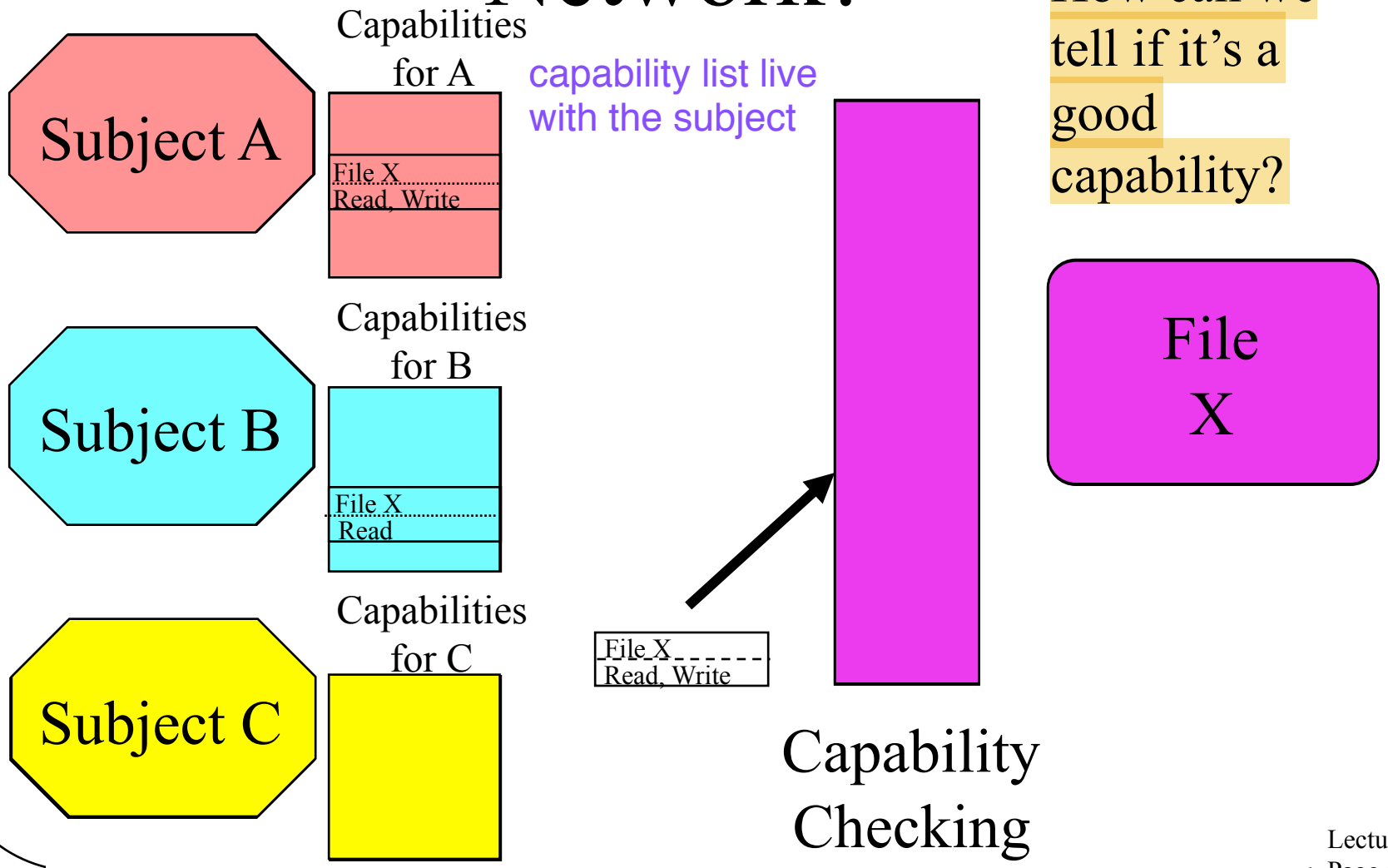


machine that store file can't trust the machine that stores the capability list

How Will This Work in a Network?

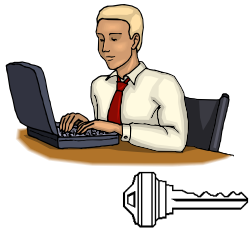
the problem:

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

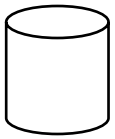


another problem with capabilities

Revoking Capabilities



Fred



Accounts receivable

How do we take away Fred's capability?

least priviledge: don't give access people who don't need access

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

how to ensure that there's no copy of machine anywhere else that Fred can use to access to the resource?

solution: keep a list of the object capabilities mapped to the person

Generation numbers - update the capabilities that are not revoked

pros and cons are good for compare and contrast questions, so look into it

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
  capabilities in distributed systems have vulnerabilities
  - 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

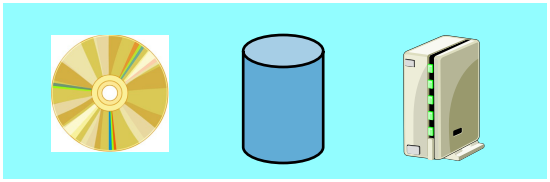
- An enhancement to ACLs or capabilities
- Each user has certain roles he can take while using the system
- At any given time, the user is performing a certain role
- Give the user access to only those things that are required to fulfill that role
- Available in some form in most modern operating systems

A Simple Example

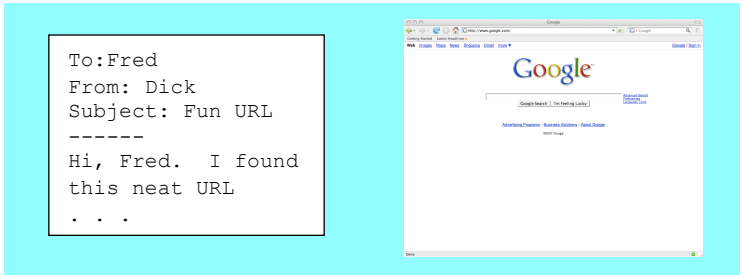
one person can have many different roles

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

as sysadmin, can't look at his own email, so Evil Malware in Fred's email can't upgrade the C++ compiler, so role-based prevented an attack

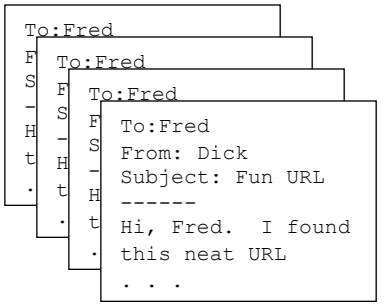
Continuing With Our Example

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 needs to be very complex, so that it proves that Fred, human user, can take on the sysadmin role. Ex. done with password

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

sysadmin overhead
- set everything up

extra usability problems, and extra management problems…

reference monitor - creates actual code

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