

Secure Systems

In the beginning ..

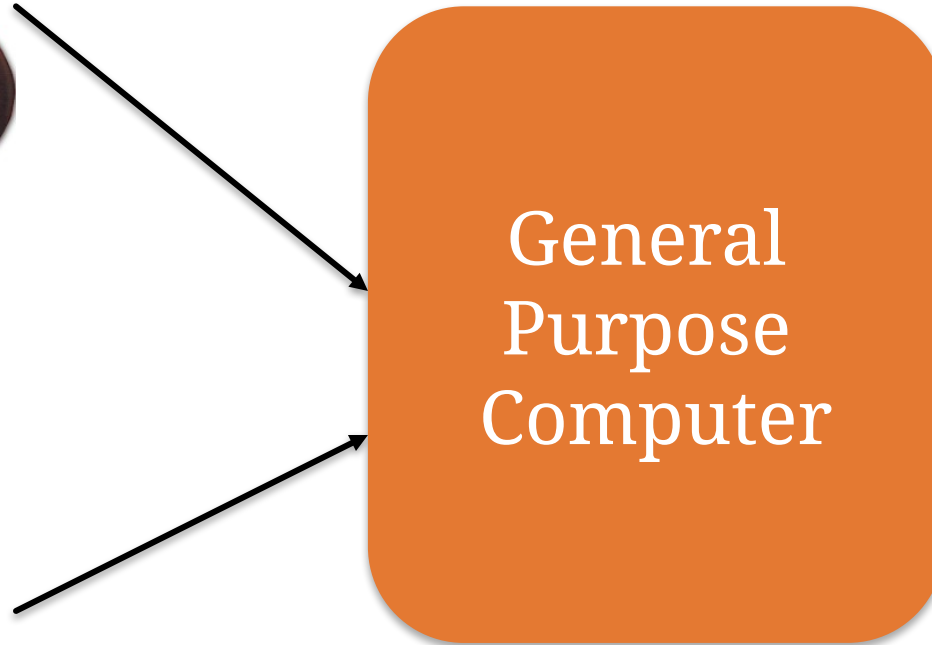
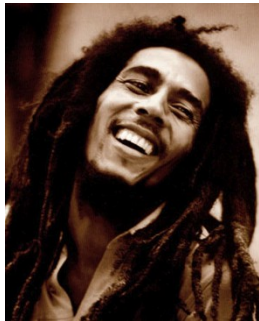
Life was simple

Limited function

Single user



General Purpose Computers



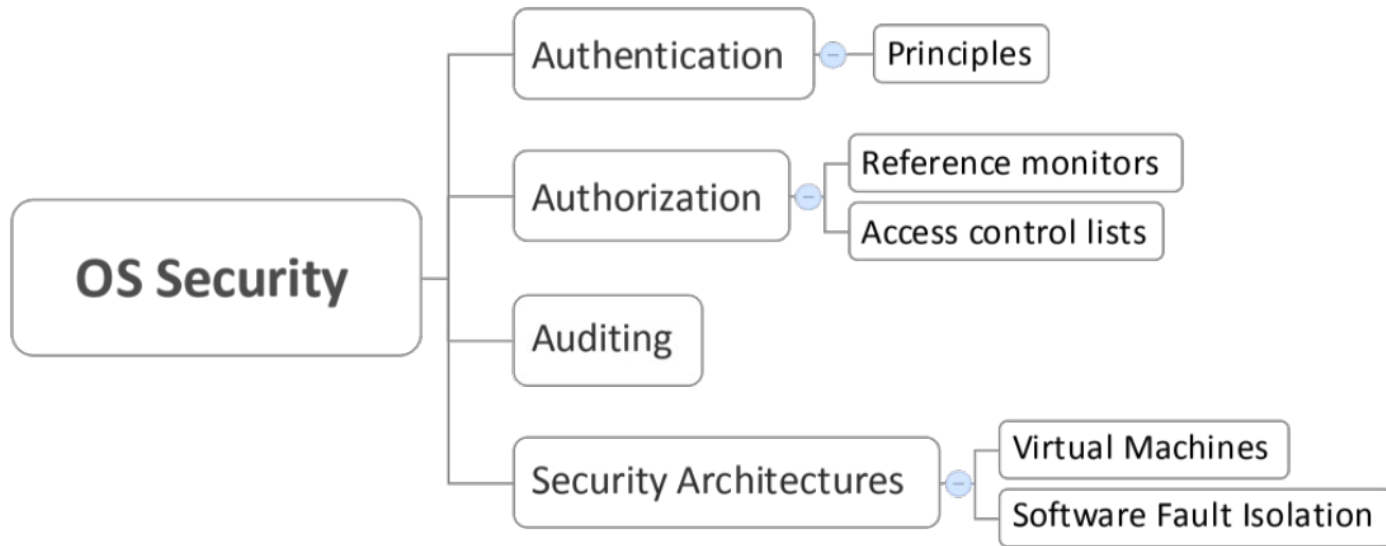
Resource1
(Alice)

Resource2
(Bob)

Resource3
(Shared)

Protection Mechanism

Control Transfer of Information
Among Users of the Utility



Goals of this lecture

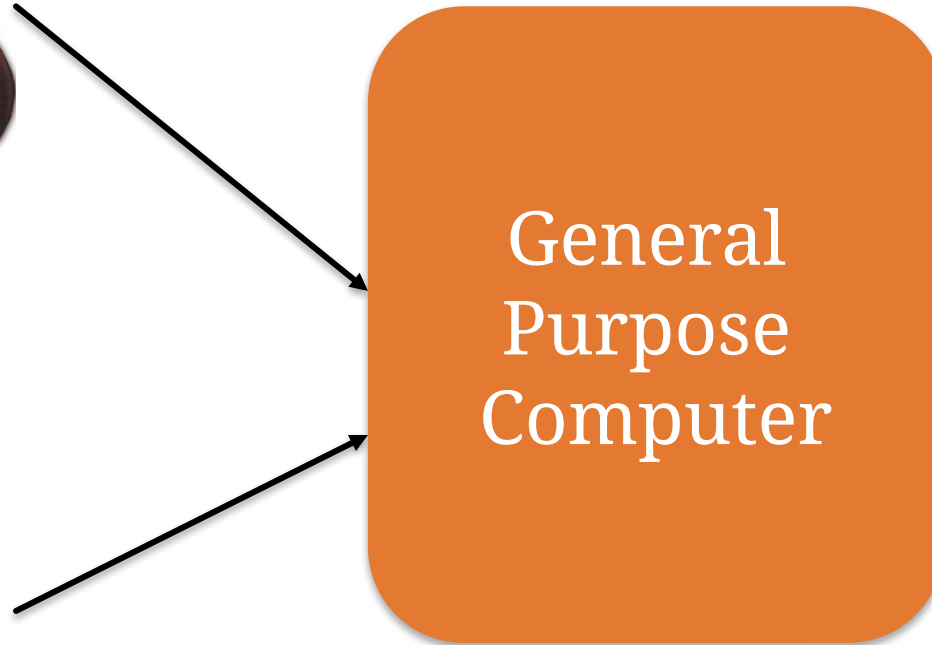
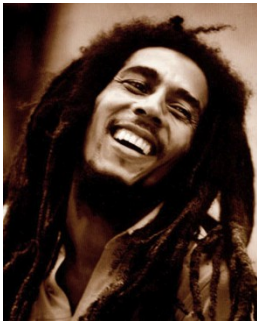
- Know Lampson's “gold” standard
 - Authentication
 - Authorization
 - Audit
- Know types of authorization mechanisms
- Understand concept of TCB
- Internalize design principles for secure systems

AAA definitions

Useful read:
Security in the Real World
Butler Lampson

<https://www.usenix.org/legacy/event/sec05/tech/lampson.pdf>

General Purpose Computers



Resource1
(Alice)

Resource2
(Bob)

Resource3
(Shared)

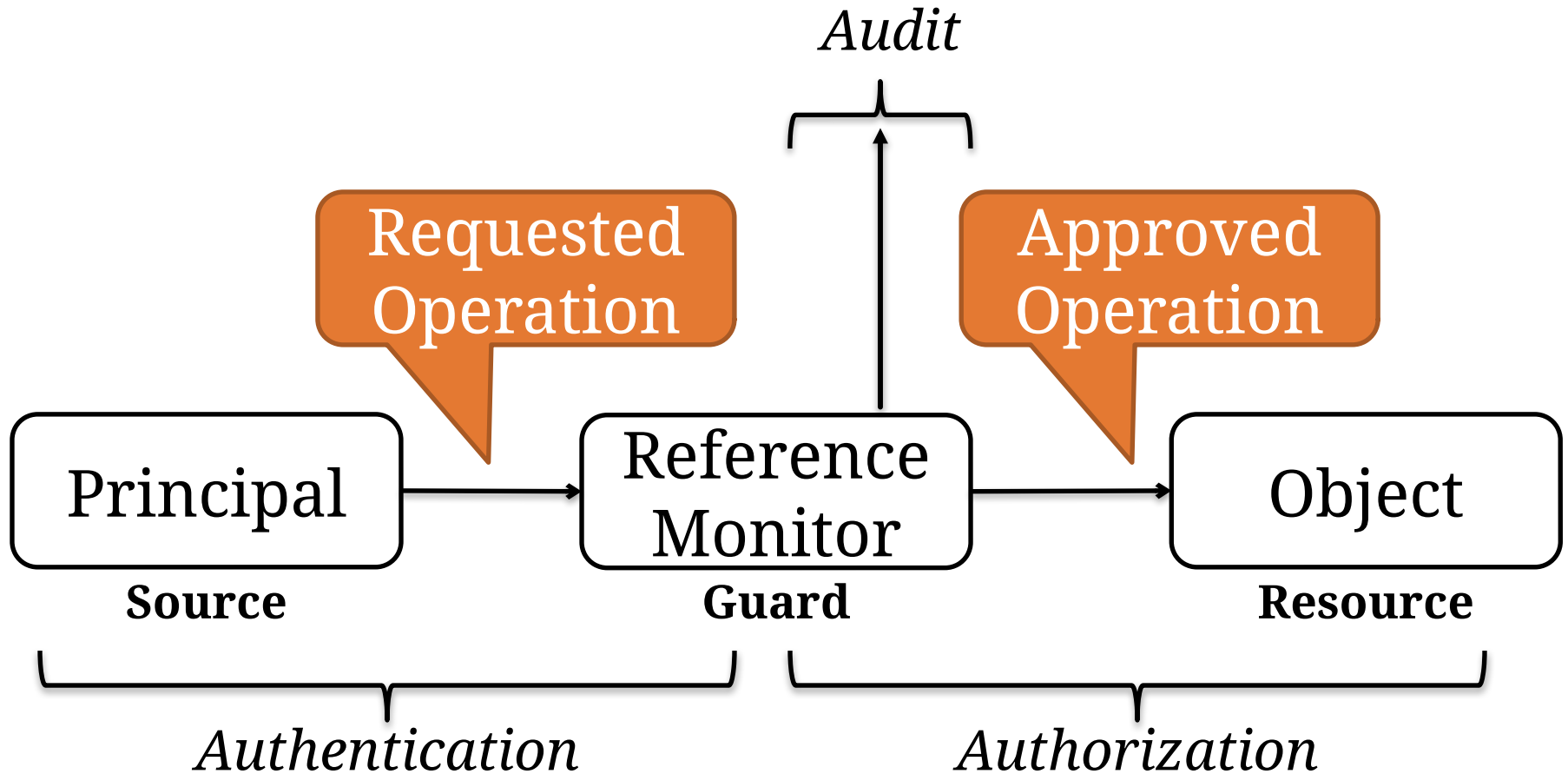
Obvious questions

- How do I know that it is in fact Alice?
- Can Alice access Bob's file? Can she access the shared file?
- Did Alice try to delete Bob's file?

Obvious questions

- How do I know that it is in fact Alice?
 - Authentication
- Can Alice access Bob's file? Can she access the shared file?
 - Authorization
- Did Alice try to delete Bob's file?
 - Audit

Abstract Access Control Model



Principals for Authentication

- “Who did that” or “Who is getting access”
- Typically user
- Can also be groups, machines, or programs

Mechanisms for Authentication

- Passwords
- Secure hardware; e.g., hardware dongle
- Two-(multi-) factor mechanisms

Authorization

Who is trusted to perform “what” operations on this object

Key question: How do we specify the policy?

(More later)

Audit

- Evidence for decisions being made
- Useful for forensics
- Useful as a diagnostic tool
- Audit-trail can help track attacks
- Extremely important!

Principles of Access Control

Protection State

- ***State of system***: current values for all resources of the system
- ***Protection State***: subset of state that deals with protection



- ***Security Policy***: Characterizes states in Q
- ***Access control matrix***: Precise representation of Q
- ***Security Mechanism***: Prevents system from entering $P-Q$



Subjects, Objects, Rights

- Objects (o): Set of protected entities relevant to system
 - Files
 - Directories
 - Memory
 - Processes
- Subjects (s): set of active objects
 - So
 - Running processes, users, ...
- Rights (r): $S \subseteq O$
 - Read
 - Write
 - Append
 - Own
 - ...

Examples

UNIX

- Subjects: Running processes
- Objects: Files, directories, processes,...
- Rights:
 - read
 - write
 - execute

AFS

- Subjects: Kerberos Principles
- Objects: Files, directories, processes, ...
- Rights
 - Lookup – List contents of directory
 - Insert – Add new files to directory
 - Delete – Remove Files
 - Administer – Change access controls
 - Read
 - Write
 - Lock - Programs that need to flock

Basic Idea: Lampson's Access Matrix

Subjects are row headings, objects are column headings. Entry $M[s,o]$ determines rights for subject s when accessing object o .

subjects

	objects (entities)					
	o_1	\dots				o_m
s_1						
s_2						
\dots						
s_n						

- Subjects $S = \{ s_1, \dots, s_n \}$
- Objects $O = \{ o_1, \dots, o_m \}$
- Rights $R = \{ r_1, \dots, r_k \}$
- Entries $A[s_i, o_j] \subseteq R$
- $A[s_i, o_j] = \{ r_x, \dots, r_y \}$
means subject s_i has rights

r_x, \dots, r_y over object o_j

From Computer Security, Art & Science 21

Example 1: File System Level

- Processes p, q
- Files f, g
- Rights r, w, x, a, o

	f	g	p	q
p	rwo	r	rwXO	w
q	a	ro	r	rwXO

Types of Access Control

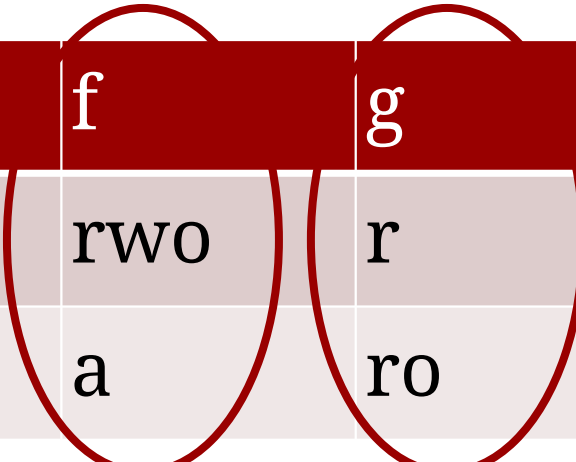
- ***Discretionary Access Control (DAC):***
User can set an access control mechanism to allow or deny access to an object.
 - It's at the users *discretion* what to allow.
 - Example: UNIX
- ***Mandatory Access Control (MAC):***
System mechanism controls access to an object and an individual user cannot alter that access.
 - What is allowed is *mandated* by the system (or system administrator).
 - Example: Law allows a court to access driving records without owners consent. **It's mandated.**

Access Control Mechanisms

- ACM's great in theory. Can be very precise. But are huge
- Two Implementation strategies:
 - Access Control Lists
 - Capabilities

Access Control Lists

- Columns in ACM's



	f	g	David	Andre
David	rwo	r	rwXO	w
Andre	a	ro	r	rwXO

- f: {(David, rwo), (Andre, a)}
- g: {(David, r), (Andre, ro)}
- More formally: An ACL is a set of pairs $(s, r): s \in S, r \subseteq R$

Default Permissions

- Normal: if not named, *no* rights over file
 - Principle of Fail-Safe Defaults
- Not-Normal: If not explicitly denied, has rights
- If many subjects, may use groups or wildcards in ACL

Abbreviations & UNIX

- ACLs can be long ... so combine users
 - UNIX: 3 classes of users: owner, group, rest
 - rwX rwX rwX
 - Ownership assigned based on creating process
 - Some systems: if directory has setgid permission, file group owned by group of directory (SunOS, Solaris)
- Limitations
 - Suppose Anne wants:
 - all rights for herself.
 - Beth to have read access
 - Caroline to have write access
 - Della to have read and write
 - Elizabeth to execute
 - 5 desired arrangements, so three triples insufficient

ACLs + Abbreviations

- Augment abbreviated lists with ACLs
 - Intent is to shorten ACL
- ACLs override abbreviations
 - Exact method varies
- Example: IBM AIX
 - Base permissions are abbreviations, extended permissions are ACLs with user, group
 - ACL entries can add rights, but on deny, access is denied

Thoughts on the ACL's

1. Which subjects can modify an ACL?
2. If there is a privileged user, what if any ACL's apply to that user?
3. Does the ACL support groups or wildcards?
4. How are contradictory access control permissions handled?
5. If a default setting is allowed, does it apply only when subject is not explicitly mentioned?

Capabilities

- Rows of access control matrix

	f	g	David	Andre
David	rwo	r	rwXO	w
Andre	a	ro	r	rwXO

- David: $\{(f, rwo), (g, r)\}$
- Andre: $\{(f, a), (g, ro)\}$
- Formally: A capability list c is a set of pairs $\{(o, r) : o \in O, r \subseteq R\}$

Semantics

- Like a bus ticket
 - Mere possession indicates rights that subject has over object
 - Object identified by capability (as part of the token)
 - Name may be a reference, location, or something else
- Must prevent process from altering capabilities
 - Otherwise subject could change rights encoded in capability or object to which they refer

Example

- UNIX `open()` call returns a file descriptor. The file descriptor is a capability
 - Even if file is deleted and a new file with the same name is created, your capability works.

Capability Implementation

1. Tags:

- Capabilities stored in memory words with associated tag bit that can only be modified in kernel model

2. Protected Memory:

- Capabilities stored in kernel memory and can only be accessed indirectly.

3. Cryptography

- Capabilities are encrypted and cannot be modified w/o detection by user process
- Can be stored in user space

Capability Revocation

- Scan all C-lists, remove relevant capabilities
 - Far too expensive!
- Use indirection
 - Each object has entry in a global object table
 - Names in capabilities name the entry, not the object
 - To revoke, zap the entry in the table
 - Can have multiple entries for a single object to allow control of different sets of rights and/or groups of users for each object

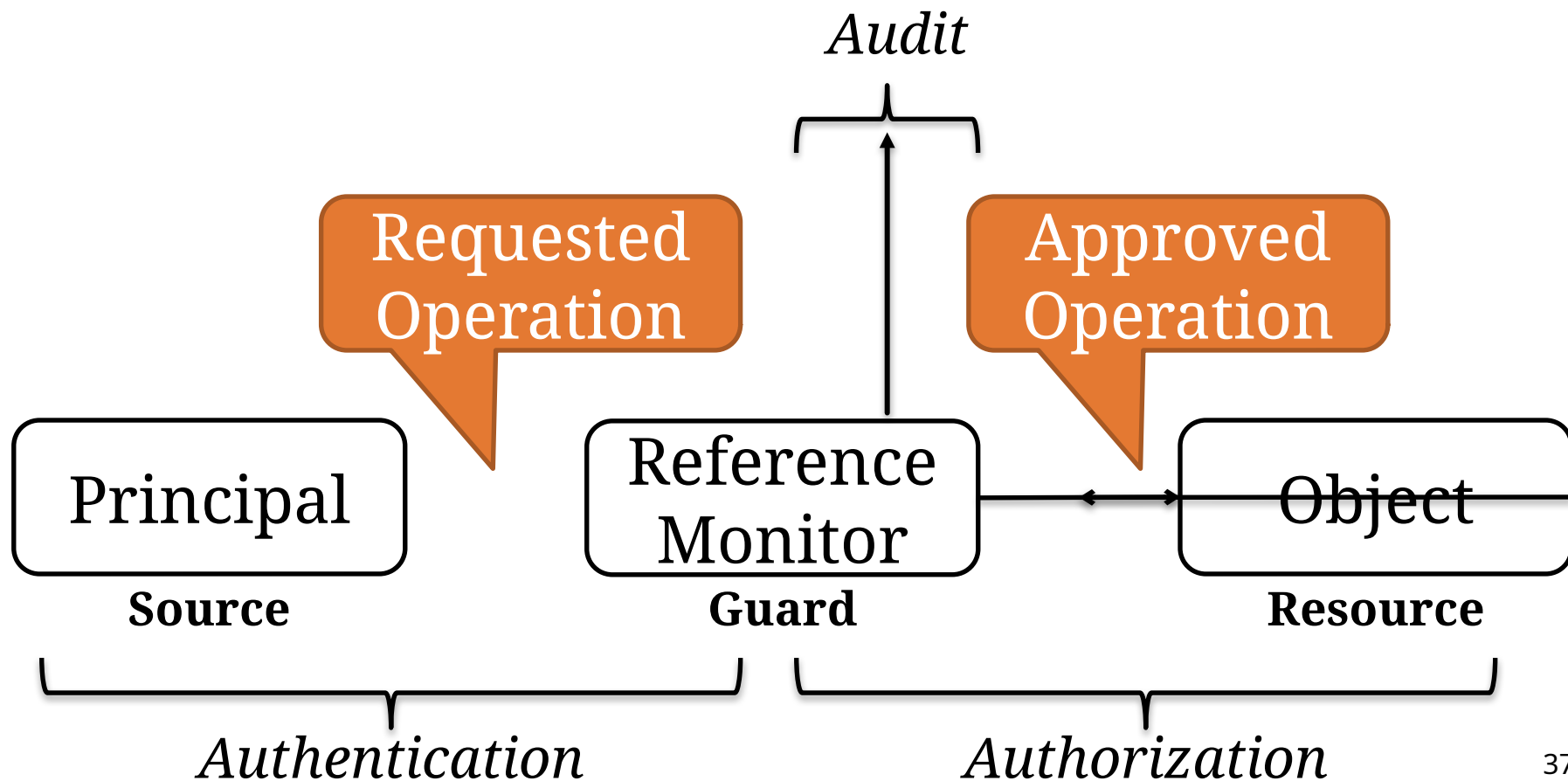
ACLs vs. Capabilities

- Both theoretically equivalent.
- Consider these questions
 1. Given a subject, what objects can it access, and how?
 - Capabilities better
 2. Given an object, what subjects can access it, and how?
 - ACLs better
- Tracking which subjects can access an object more common, thus ACL's more popular because they are more efficient.
- Other choices possible, e.g., revocation on per-subject basis easier with Capabilities.

Notion of Trusted Computing Base

Why should I trust this?

What does it mean for the access control mechanism to work correctly?



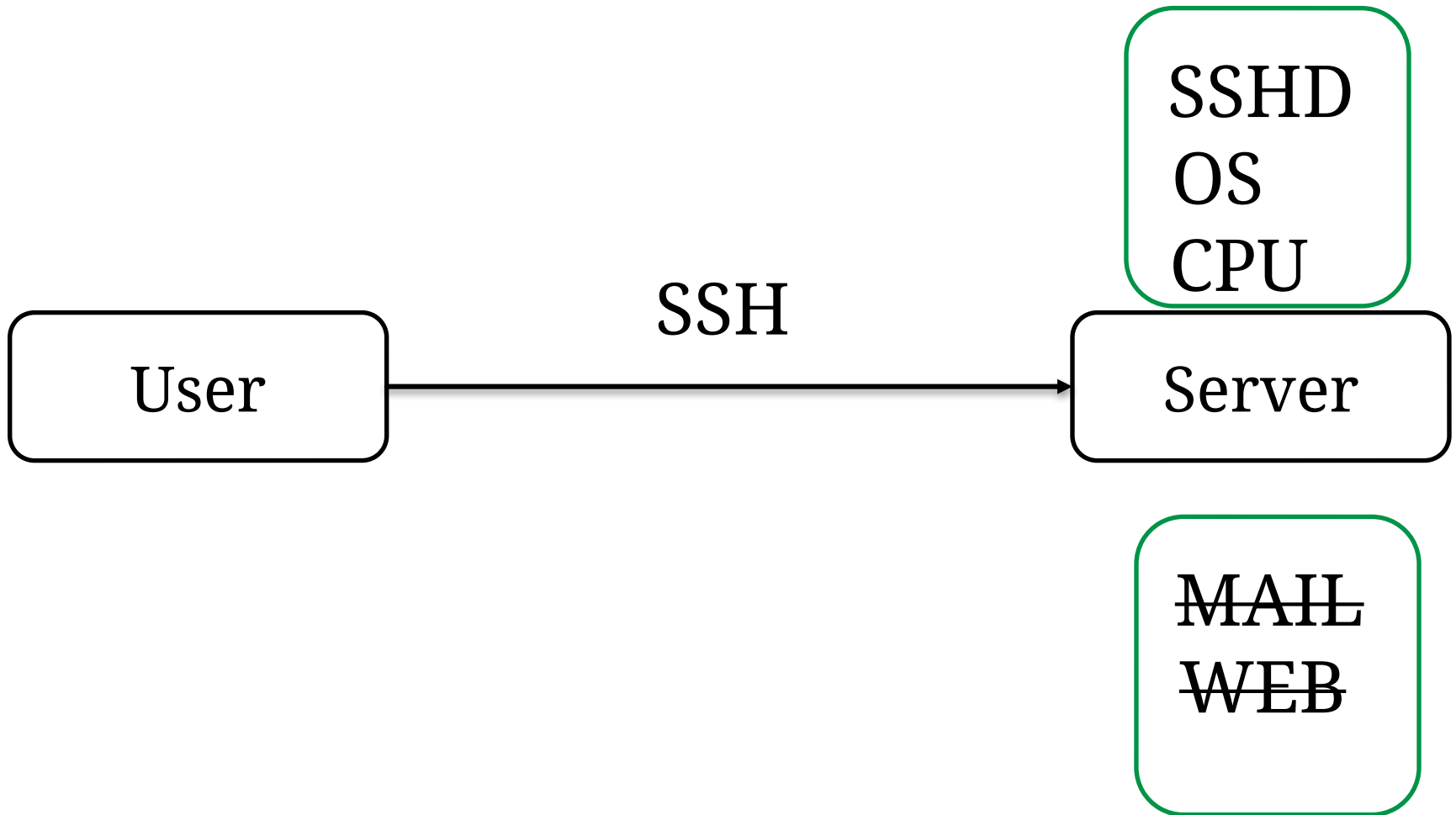
Need a “trusted” component

- We rely upon to operate correctly
- Necessary for the whole system to be secure
- Corollary: If this misbehaves all bets re: security goals are off!

Example of TCB

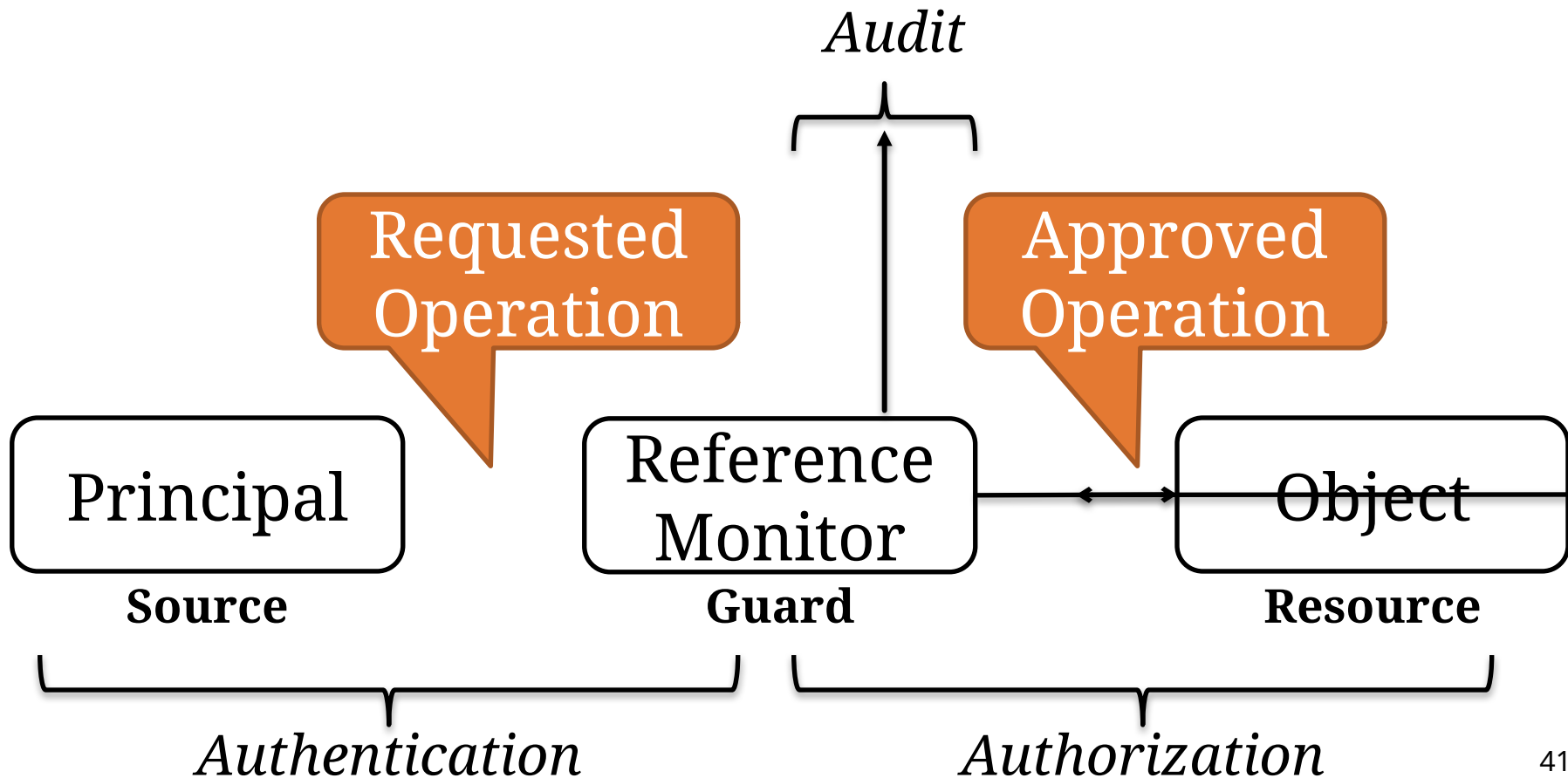


Example of TCB



What is in the TCB here?

What does it mean for the access control mechanism to work correctly?



Ideal TCB Design

- Cannot be bypassed
 - E.g., finding a route that firewall doesn't check
- Tamper resistant
 - E.g., messing with the SSHD or OS executables
- Verifiable
 - Implies you want TCB to be as small as possible

Why do we need a TCB?

- Securing every piece of a system is hard!
- TCB is a pragmatic compromise
 - Separate system into trusted and untrusted
- Can focus security efforts on the trusted piece
 - Reason about security more rigorously
- Caveat: Determining TCB is easier said than done

Design Principles

Useful read:

The Protection of Information in Computer Systems

Jerome Saltzer and Michael Schroeder

The Key Principles

- Economy of mechanism a.k.a KISS
- Fail-safe defaults
- Complete mediation
- Separation of privilege
- Least privilege
- Factor in users/acceptance/psychology
- Work factor/economics
- Detect if you cant prevent
- Don't rely on security by obscurity

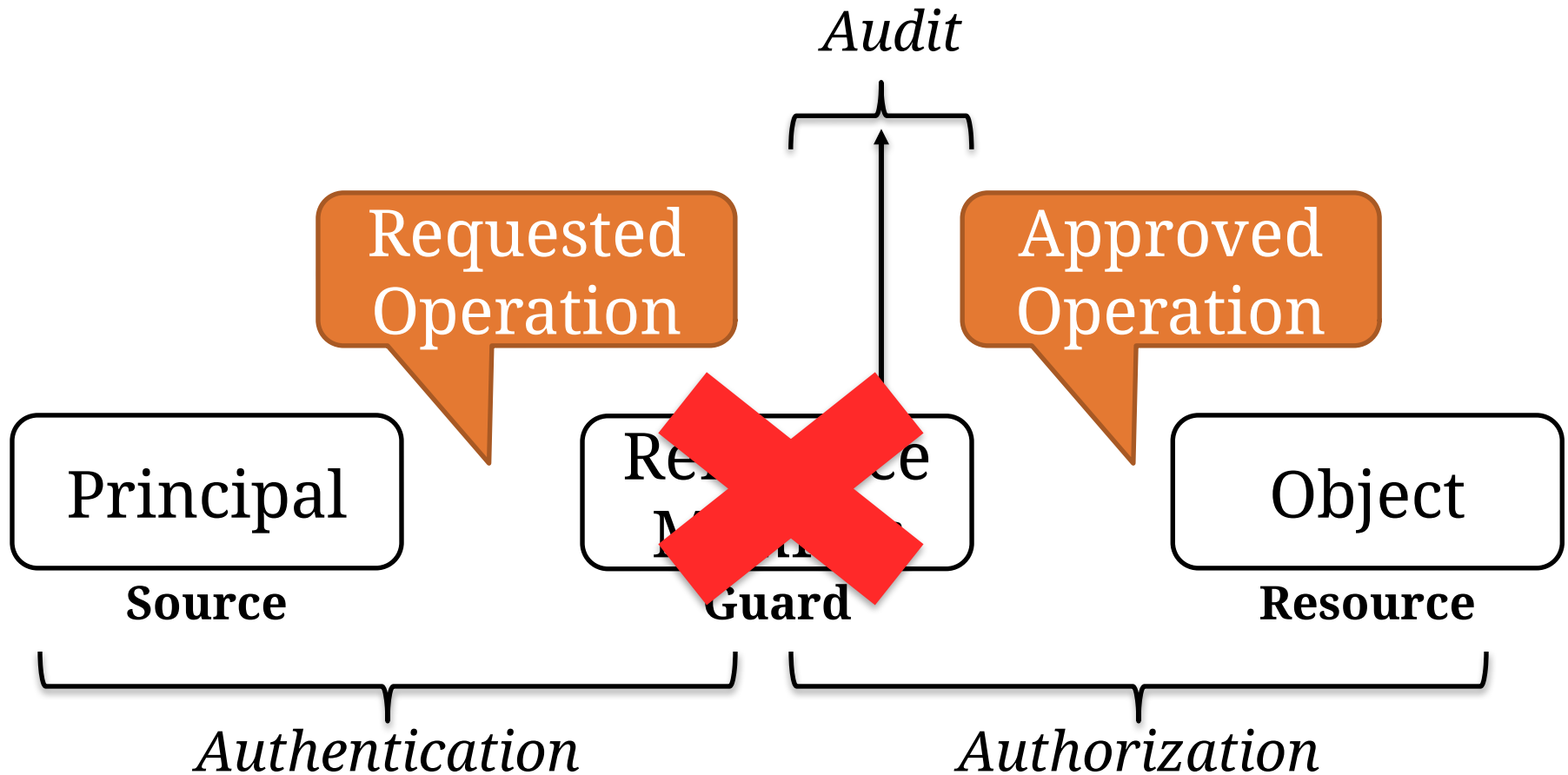
Keep it Simple, Stupid (KISS)

- Rule of thumb:
1-5 defects per 1K lines of code
- Windows XP = 40 MLOC
 - All in TCB
- Smaller, simpler TCB is easier to reason about

The Key Principles

- Economy of mechanism a.k.a KISS
- Fail-safe defaults
- Complete mediation
- Separation of privilege
- Least privilege
- Factor in users/acceptance/psychology
- Work factor/economics
- Detect if you cant prevent
- Don't rely on security by obscurity

Fail safe Defaults



The Key Principles

- Economy of mechanism a.k.a KISS
- Fail-safe defaults
- Complete mediation
- Separation of privilege
- Least privilege
- Factor in users/acceptance/psychology
- Work factor/economics
- Detect if you cant prevent
- Don't rely on security by obscurity

Complete Mediation

- Every access to every object is checked by the reference monitor
- Easier said than done!
- TOCTTOU problems


Mediation: TOCTTOU

Vulnerabilities

Time-Of-Check-To-Time-Of-Use

```
int openfile(char *path){  
    struct stat s;  
    if (stat(path,&s) < 0))  
        return -1;  
    if (!S_ISREG(s.st_mode)){  
        error("only regular files allowed");  
        return -1;  
    }  
    return open(path,O_RDONLY)  
}
```

Change path




Mediation: TOCTTOU

Vulnerabilities

Time-Of-Check-To-Time-Of-Use

```
Void withdraw(int w){  
    b = getbalance();  
    if (b < w)  
        error("not enough $$");  
    b = b - w;  
    send(w)  
}
```



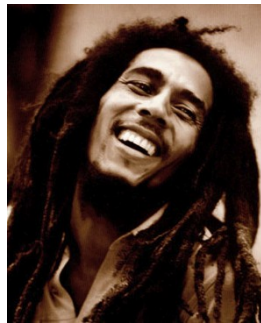
The Key Principles

- Economy of mechanism a.k.a KISS
- Fail-safe defaults
- Complete mediation
- Separation of privilege
- Least privilege
- Factor in users/acceptance/psychology
- Work factor/economics
- Detect if you cant prevent
- Don't rely on security by obscurity

Separation of Privileges



Launch Missile



Launch Missile



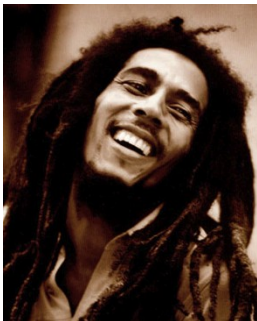
The Key Principles

- Economy of mechanism a.k.a KISS
- Fail-safe defaults
- Complete mediation
- Separation of privilege
- Least privilege
- Factor in users/acceptance/psychology
- Work factor/economics
- Detect if you cant prevent
- Don't rely on security by obscurity

Least Privilege



Read File1



General
Purpose
Computer

File1
(Alice)

File2
(Bob)

File3
(Shared)

The Key Principles

- Economy of mechanism a.k.a KISS
- Fail-safe defaults
- Complete mediation
- Separation of privilege
- Least privilege
- Factor in users/acceptance/psychology
- Work factor/economics
- Detect if you cant prevent
- Don't rely on security by obscurity

Lecture summary

- Know Lampson's “gold” standard
 - Authentication
 - Authorization
 - Audit
- Know types of authorization mechanisms
 - Mandatory vs discretionary, Capabilities vs ACL
- Understand concept of TCB
- Internalize design principles for secure systems