

# Διάλεξη #10 - Access Control

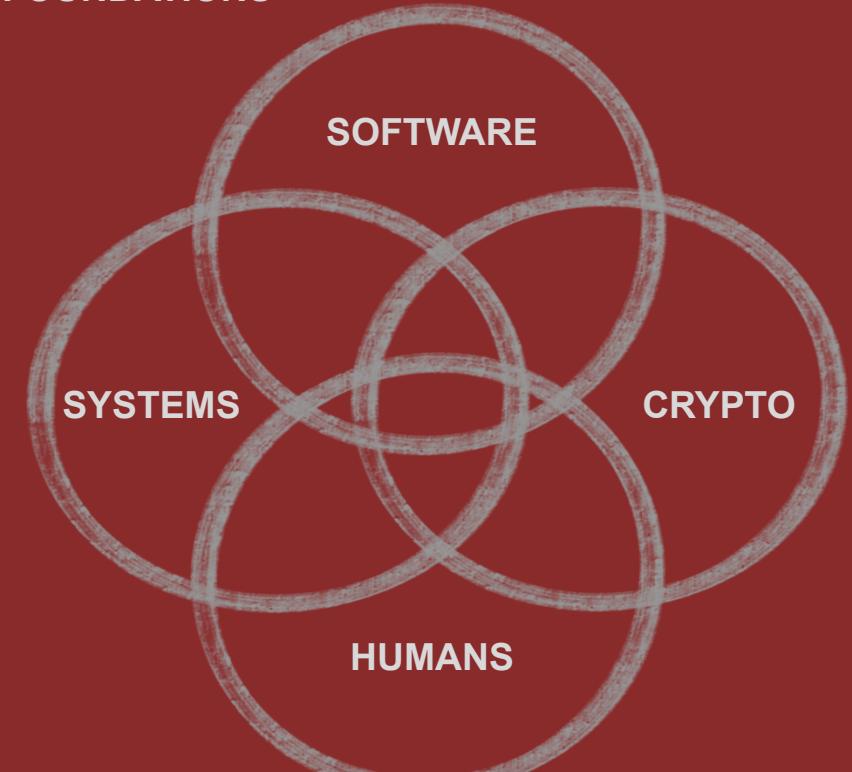
Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Εισαγωγή στην Ασφάλεια  
Θανάσης Αυγερινός

\*some slides by John Mitchell

Huge thank you to [David Brumley](#) from Carnegie Mellon University for the guidance and content input while developing this class

FOUNDATIONS

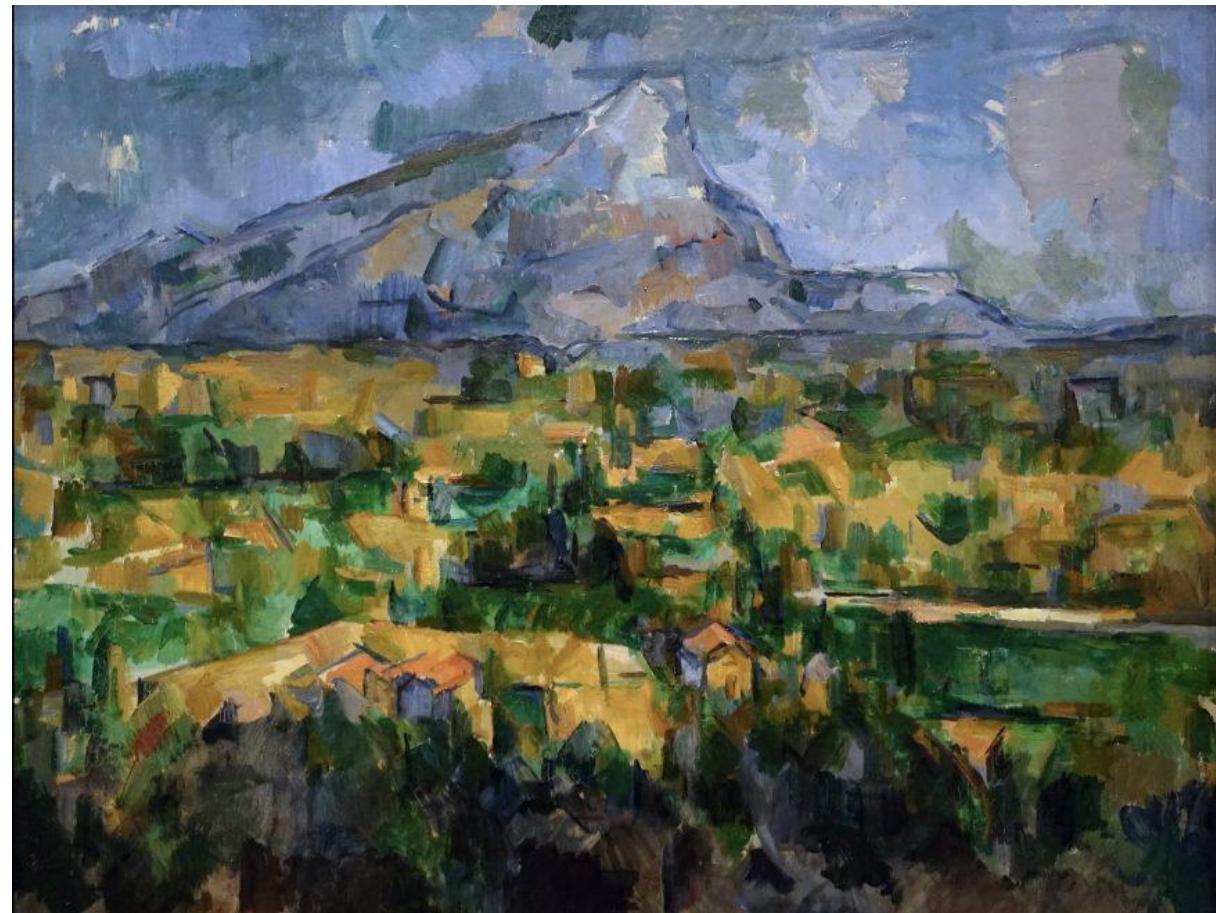


# Ανακοινώσεις / Διευκρινίσεις

- Βγήκε η Εργασία #1 - Προθεσμία 30 Απριλίου 23:59!

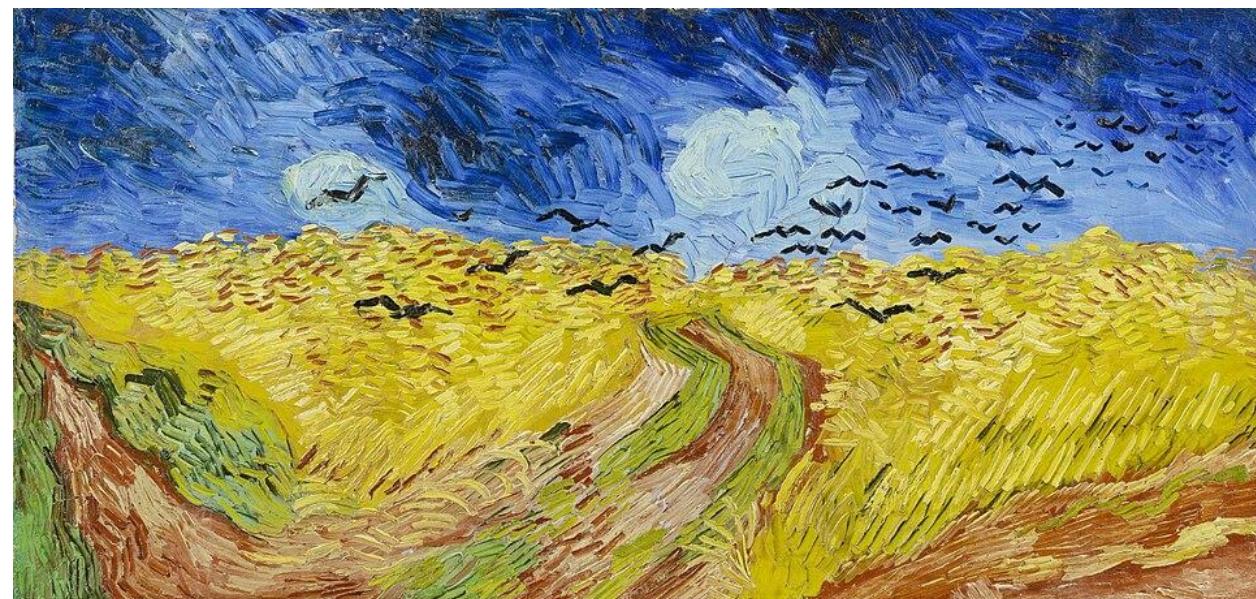
# Την Προηγούμενη Φορά

- Bypassing Mitigations
- Return-Oriented Programming (ROP)
- Static and Dynamic Linking



# Σήμερα

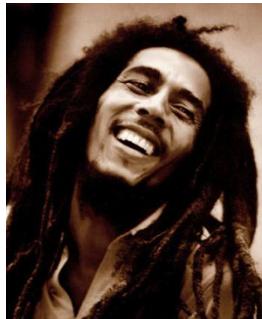
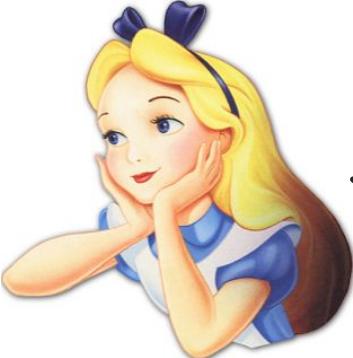
- Reference Monitors
- "Gold" (Au) Standard: Authentication + Authorization + Audit
- Authorization Mechanisms / Access Control
  - Access Control Lists (ACLs) and Capabilities (CAP)
  - Discretionary Access Control (DAC)
  - Role-Based Access Control (RBAC)





**Access to  
Resources and  
Control**

# General Purpose Computers



Resource1  
(Alice)

Resource2  
(Bob)

Resource3  
(Shared)

# 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

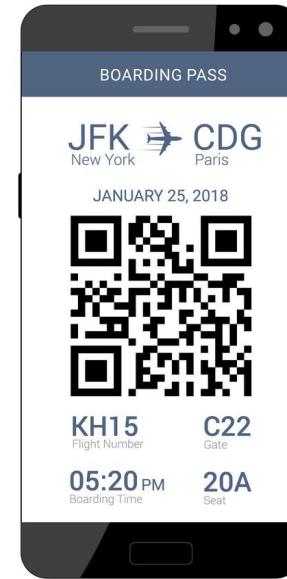
Did Alice try to delete Bob's file?

- Audit

# Authentication vs Authorization

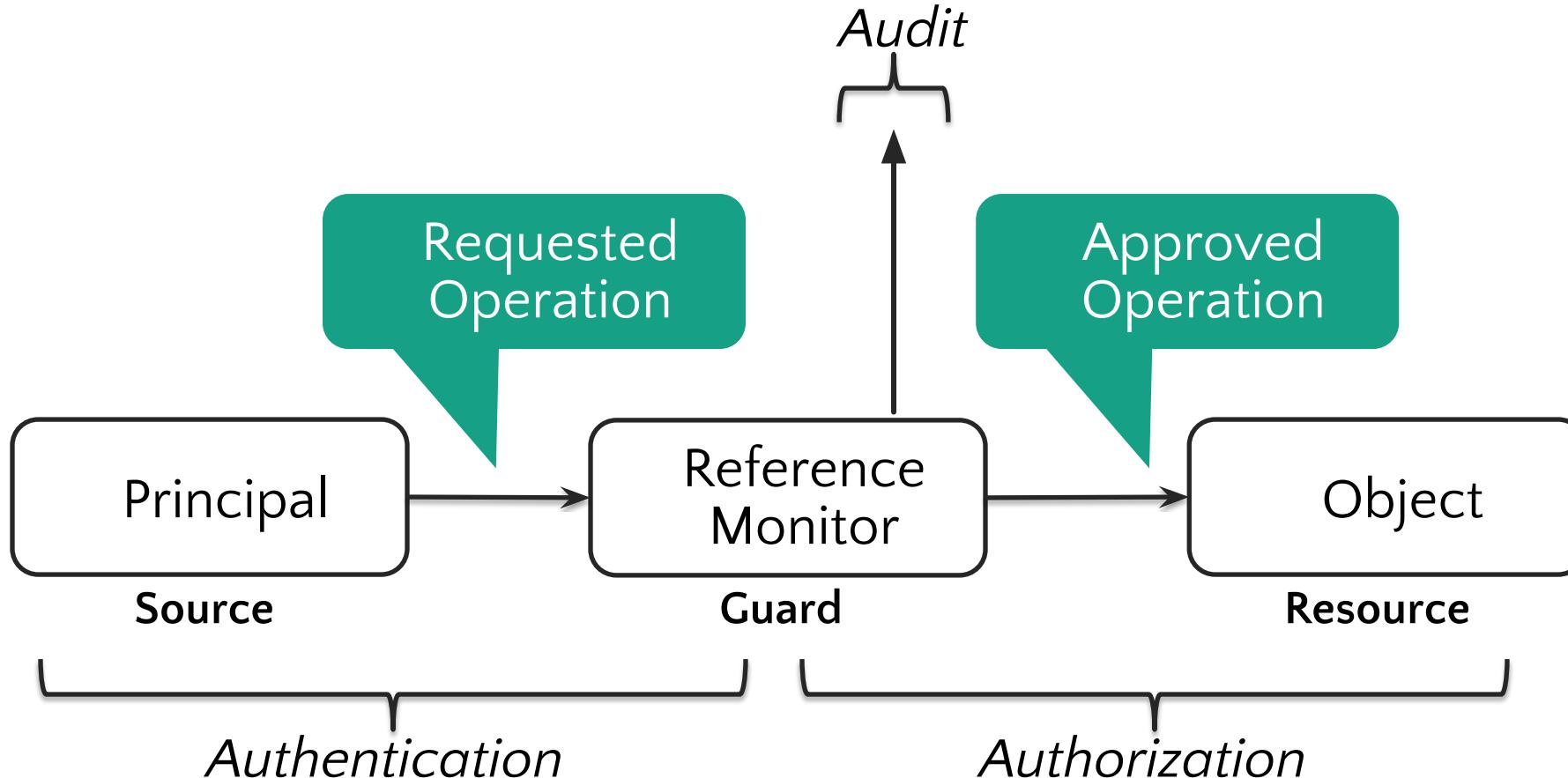


Authentication: Who a user is  
Example: Drivers license authenticates your identity



Authorization: What users can access  
Example: Boarding pass authorizes you to board a flight

# Abstract Access Control Model



# Principals for Authentication

- “Who did that” or “Who is getting access”
- Commonly: user
- Also:
  - Program / phone app
  - Network port
  - Process
  - Machine

# Mechanisms for Authentication

and sometimes authorization

- For humans:
  - Something you know, something you have, something you are
    - E.g., password, YubiKey, fingerprint
  - Two-factor or multi-factor mechanisms
    - Ideally, using factors from different categories
- For code / data / processes:
  - Trusted metadata
    - E.g., process ID
  - Cryptographic authentication
    - E.g., digital signature, MAC

# Authorization

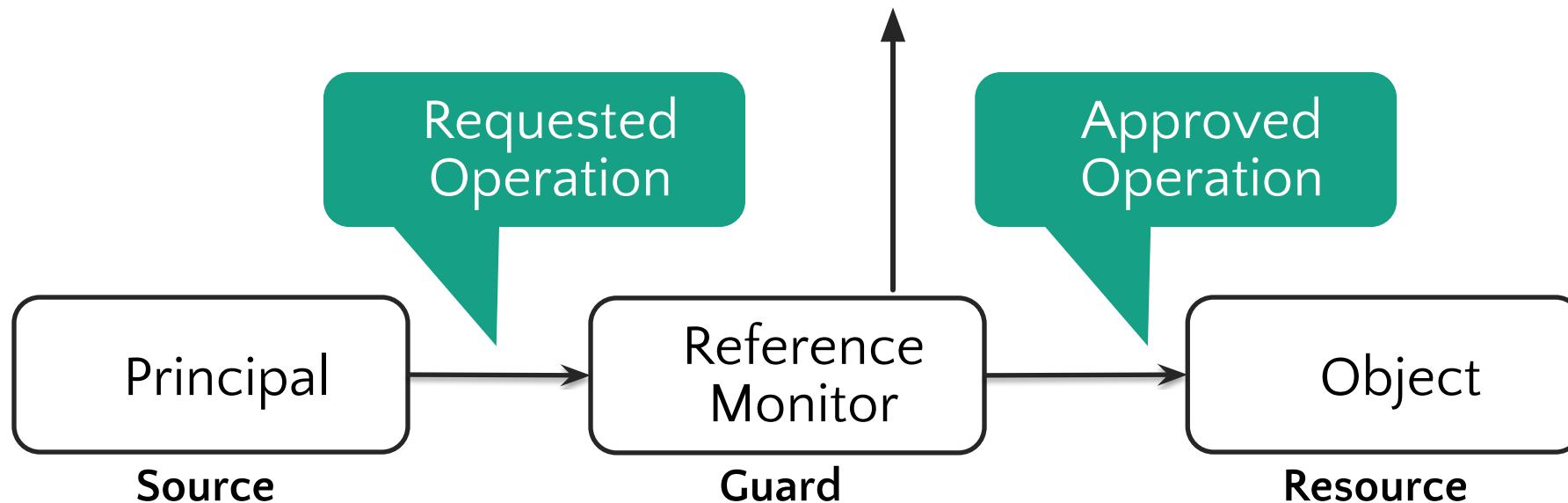
Who is trusted to perform “what” operations on this object

Important question: How do we specify the policy?  
(spoiler: Access Control!)

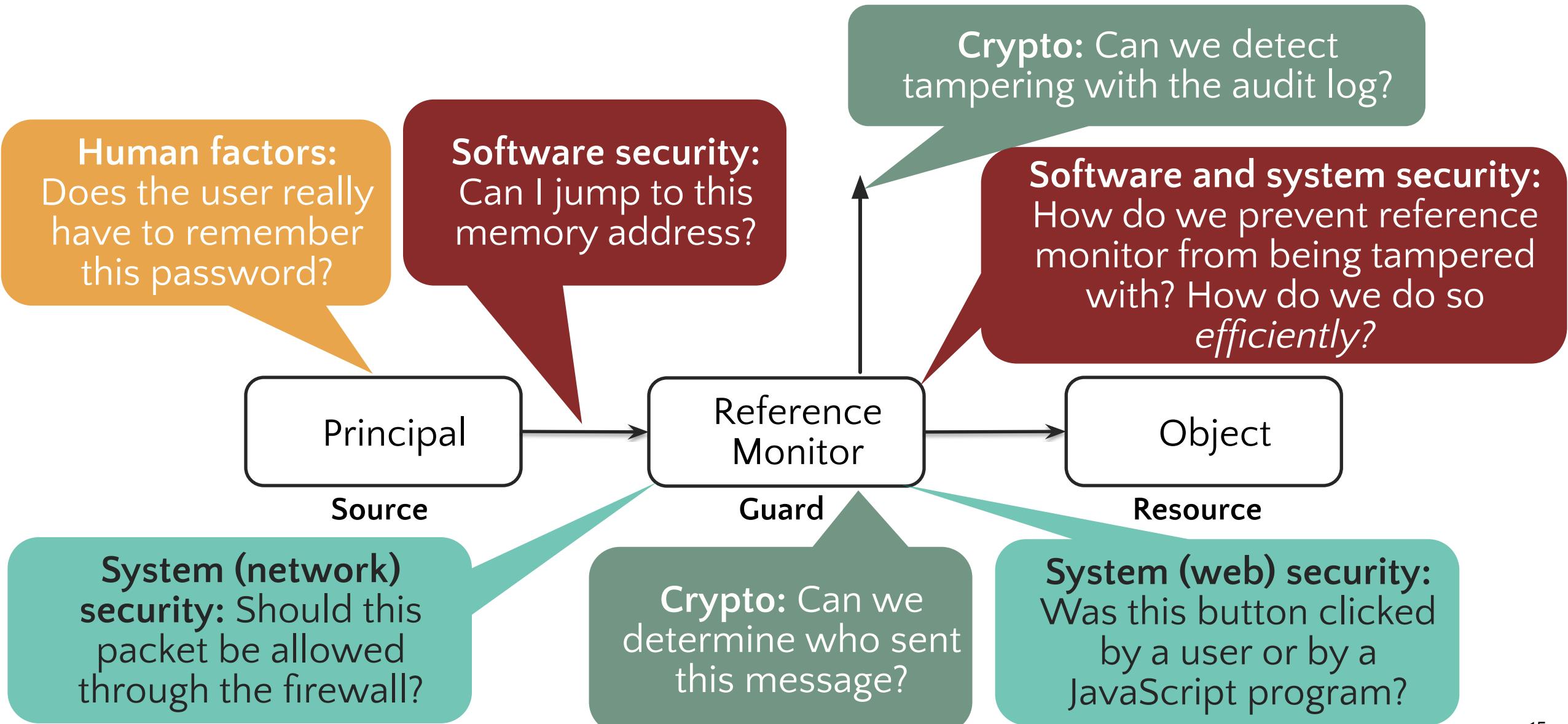
# Audit

- Evidence of and for decisions being made
- Why do we need audit?
  - Useful for forensics
  - Useful as a diagnostic tool
  - Audit trail can help track attacks
- Extremely important, but often forgotten

# Abstract Access Control Model



# Abstract Access Control Model



# Participation Question

Which of the following is an example of an authorization check?

- A. Consulting the system log to see which admin introduced a backdoor
- B. my-studies checks to see if you are registered as a professor for the course before allowing you to change the grade of a student
- C. Showing your driver's license at the airport's security checkpoint
- D. An ice skating rink wants you to sign a waiver before you start skating

# **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***: One kind of 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  $S \subseteq O$ 
  - Running processes, users, ...
- Rights (r):
  - Read
  - Write
  - Execute
  - Append
  - Own

# Examples

## UNIX

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

## AFS

- Subjects: Kerberos principals
- 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

# Lampson's Access Matrix

- Subjects are row headings, objects are column headings
- Access control entry  $[s,o]$  determines rights for subject  $s$  when accessing object  $o$

		objects (entities)					
		$o_1$	...	$o_m$			
subjects	$s_1$						
	$s_2$						
	...						
	$s_i$						
	$s_n$						

- 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  $o_j$

# Example: Processes and files

- Processes p,q      (subjects & objects)
- Files f,g              (objects)
- Rights r,w,x,a,o

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

# Playing with Unix

# *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 user's discretion what to allow
  - Example: UNIX file ownership
- **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: Even the author of a TOP SECRET file cannot make it PUBLIC

# Types of Access Control

- **Role-Based Access Control (RBAC):**  
Access based on *role*, not *identity*
  - A role defines a set of permissions
  - Thanassis as Faculty @ DI vs Thanassis as member of UOA vs Thanassis as Hacker vs ...
- *Orthogonal* to DAC vs MAC
- Roles and groups (e.g., UNIX groups) are related, but not identical
  - Role  $R \supseteq$  Role  $S \Rightarrow$  Role  $S$  has *at most* the permissions of Role  $R$
  - Group  $G \supseteq$  Group  $H \Rightarrow$  Group  $H$  has *at least* the permissions of Group  $G$

# Access Control Mechanisms

- Ways of specifying and enforcing policy
- Access control matrices:
  - Can be precise
  - Can be huge
  - Two common implementation strategies:
    - Access Control Lists
    - Capabilities

# Access Control Lists

	f	g	Ethan	Andre
Ethan	rwo	r	rwxo	w
Andre	a	ro	r	rwxo

An ACL corresponds to a column in an Access Control Matrix

- f: {((Ethan, rwo), (Andre, a))}
- g: {((Ethan, r), (Andre, ro))}

Formally: an ACL for an object o is a list of pairs:  $\{(s_i, r_i)\}$

# Default Permissions

- Normal: if not named in ACL, *no* rights over file
  - Recall from last time: Principle of Fail-Safe Defaults!
- Unusual but possible: If not explicitly denied, has rights
- If many subjects, may use groups or wildcards in ACL

# Design Decisions for ACLs

1. Which subjects can modify an ACL?
2. Which, if any, ACLs apply to privileged users (e.g., root)?
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?

# Aside: abbreviations & UNIX

- ACLs can be long ... so combine users into classes
  - UNIX: 3 classes of users: Owner, Group, All
  - rwx rwx rwx
  - e.g., `chmod 644 /var/www/index.html`
  - Ownership (and default permissions) assigned based on creating process
- 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

# Capabilities

	f	g	Ethan	Andre
Ethan	rwo	r	rwxo	w
Andre	a	ro	r	rwxo

A Capability corresponds to a row in an Access Control Matrix

- Ethan:  $\{(f, \text{rwo}), (g, r), (\text{Ethan}, \text{rwxo}), (\text{Andre}, w)\}$
- Andre:  $\{(f, a), (g, ro), (\text{Ethan}, r), (\text{Andre}, \text{rwxo})\}$

Formally: a Capability for subject  $s$  is a list of pairs:  $\{(o_i, r_i)\}$

# Semantics of a capability

- Like a bus ticket
  - 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
- This file descriptor is a capability!
  - Even if file is deleted and a new file with the same name is created, the capability still works
  - Aside: this is a common way of creating a temporary file that will be deleted as soon as it's closed. See man unlink

# Capability Implementations

## 1. Tags:

- Capabilities are stored in memory words with an associated tag bit that can only be modified in kernel mode
- e.g., [CHERI capability machine](#) (coming soon to a RISC-V near you!)

## 2. Protected memory:

- Capabilities stored in kernel memory and can only be accessed indirectly (i.e., via syscall)

## 3. Cryptography

- Capabilities are cryptographically authenticated and cannot be modified by user process
- Can be stored in user space

# How can we revoke capabilities?

- Scan all outstanding capabilities, delete or invalidate relevant ones
  - 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

# Comparing ACLs and Capabilities

- Both can be used to describe the same access control policy
- Consider these questions
  - Given a subject, what objects can it access, and how?
    - Capabilities more efficient
  - Given an object, what subjects can access it, and how?
    - ACLs more efficient
- Tracking which subjects can access a given object is more common, thus ACLs are more popular because they are more efficient for this case
- Other trade-offs may be worthwhile, e.g., revocation on per-subject basis is easier with capabilities

# Participation Question

Which of the following is an example of a capability in the real world?

- A. The bouncer at a party looks at your ID and checks if your name is on the invite list
- B. You use your dorm room key to unlock and enter your room
- C. You run a 5-minute mile to show that you're qualified to join the track team

# Process effective user id (EUID)

- Each process has three Ids (+ more under Linux)
  - Real user ID (RUID)
    - same as the user ID of parent (unless changed)
    - used to determine which user started the process
  - Effective user ID (EUID)
    - from set user ID bit on the file being executed, or sys call
    - determines the permissions for process
      - file access and port binding
  - Saved user ID (SUID)
    - So previous EUID can be restored
- Real group ID, effective group ID, used similarly

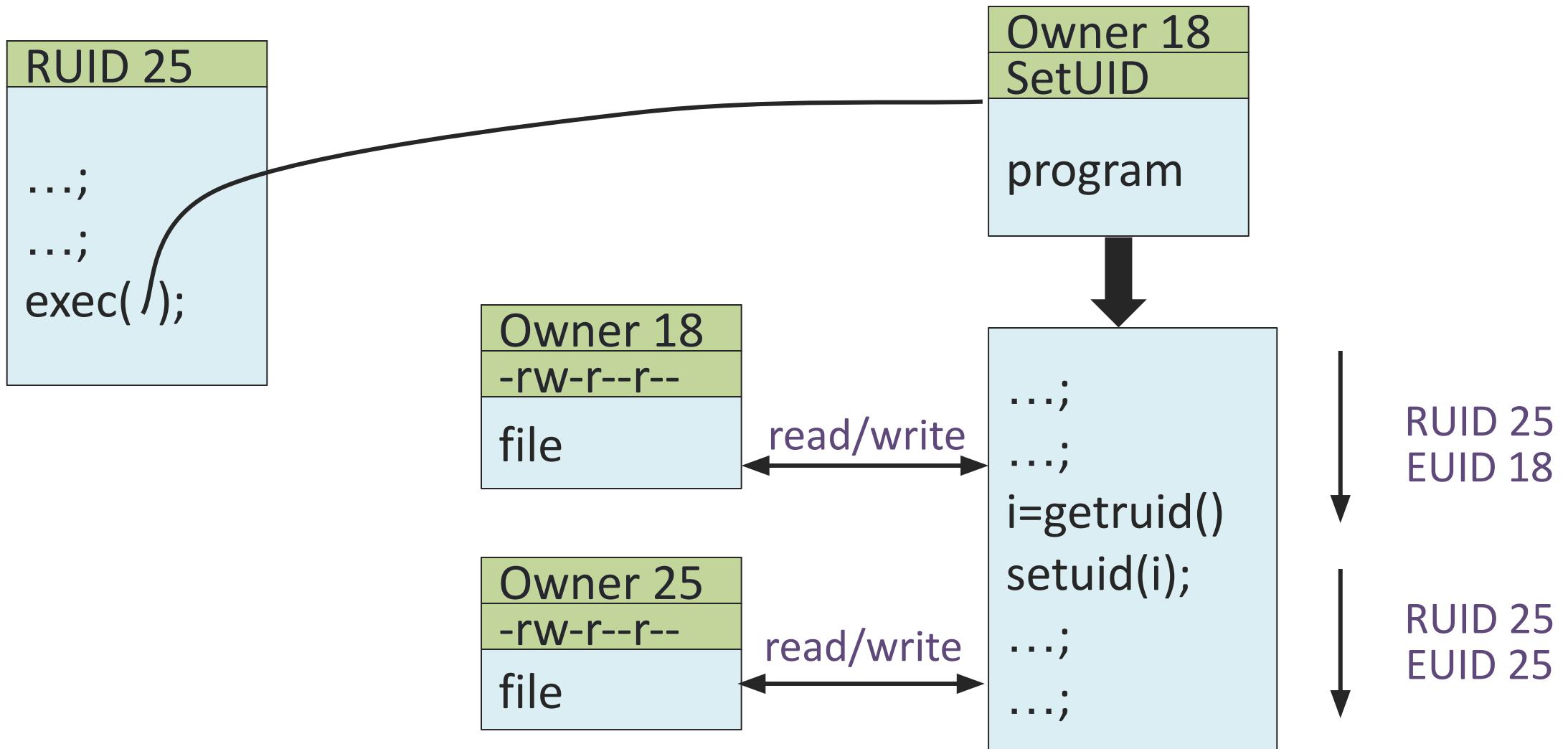
# Process Operations and IDs

- Root
  - ID=0 for superuser root; can access any file
- Fork and Exec
  - Inherit three IDs, except exec of file with setuid bit
- Setuid system call
  - seteuid(newid) can set EUID to
    - Real ID or saved ID, regardless of current EUID
    - Any ID, if EUID is root
- Details are actually more complicated
  - Several different calls: setuid, seteuid, setreuid

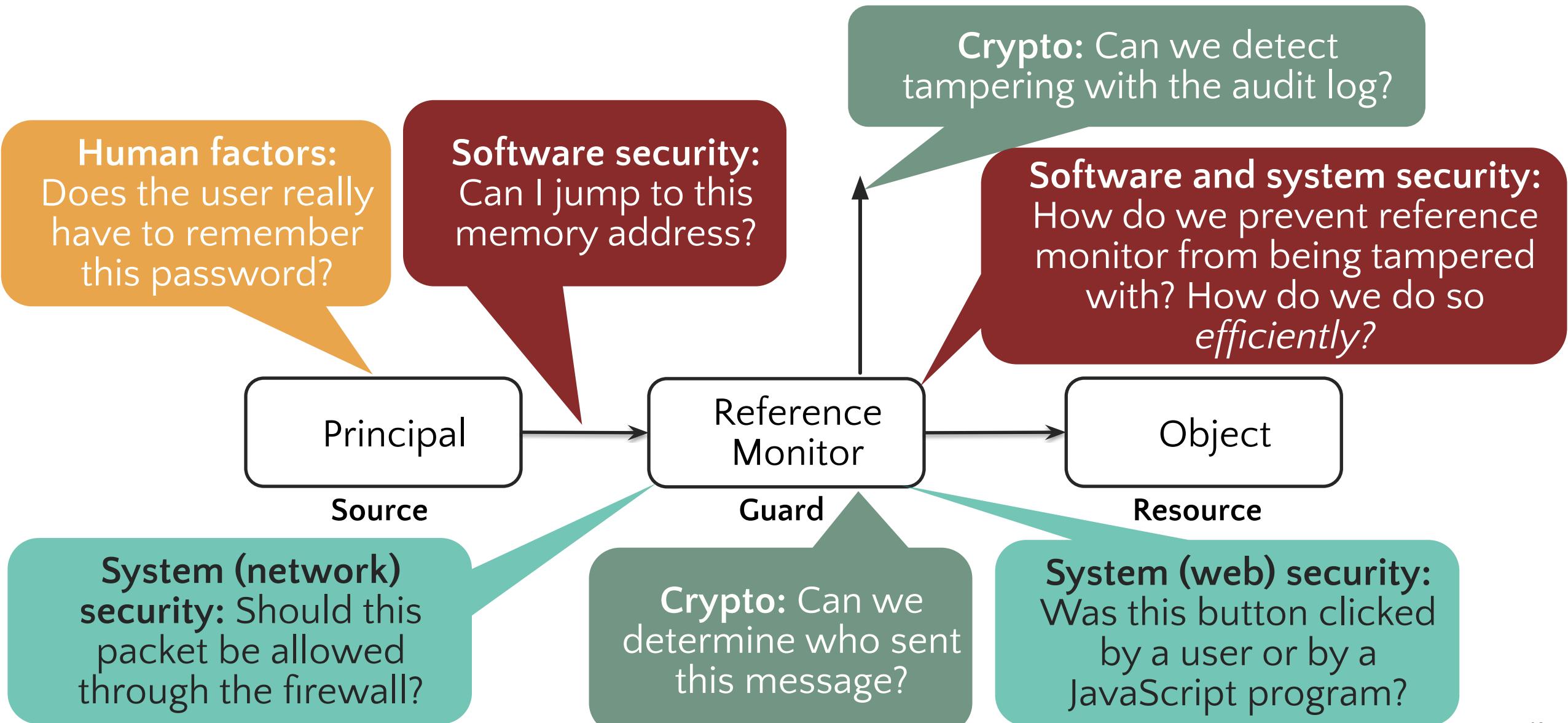
# Setid bits on executable Unix file

- Three setid bits
  - Setuid – set EUID of process to ID of file owner
  - Setgid – set EGID of process to GID of file
  - Sticky
    - Off: if user has write permission on directory, can rename or remove files, even if not owner
    - On: only file owner, directory owner, and root can rename or remove file in the directory

# Example



# Abstract Access Control Model



# Takeaways

- A reference monitor is the security mechanism in an abstract model of access control
- The “Gold” standard
  - Authentication, authorization, audit
  - Relevant to every corner of security!
- Know types of authorization mechanisms
  - Mandatory vs discretionary, capabilities vs ACL
- Principal  $\neq$  principle

**Ευχαριστώ και καλή μέρα εύχομαι!**

Keep hacking!