Lecture 17: Capabilities

CS 181S

Spring 2024

Where we were…

- **Authentication:** mechanisms that bind principals to actions

- **Authorization:** mechanisms that govern whether actions are permitted
  - Discretionary Access Control
  - Mandatory Access Control

Access Control Policy

- An **access control policy** specifies which of the **operations** associated with any given **object** each **principal** is authorized to perform
- Expressed as a relation $Auth$:

$Auth$		Objects	
		dac.tex	dac.pptx
principals	ebirrell	r,w	r,w
	faculty	r	r
	student		r

Capability Lists

Access Control Lists

Protection Domains

- Motivation: users are too coarse-grained to define privileges
- **Protection Domains**:
  - Each thread of control is associated with a protection domain
  - Each protection domain is associated with a different set of privileges
  - We allow transitions from one protection domain to another as execution of the thread proceeds.

Protection Domains

- Typical implementation: certain system calls cause protection-domain transitions.
  - System calls for invoking a program or changing from user mode to supervisor mode are obvious candidates.
- Some operating systems provide an explicit domain-change system call instead
  - the application programmer or a compiler's code generator is then required to decide when to invoke this domain-change system call
- We use the term **attenuation of privilege** for a transition into a protection domain that eliminates privileges.
- We use the term **amplification of privilege** for a transition into a protection domain that adds privileges.

Protection Domains

		Objects				
		dac.tex	dac.pptx	ebirrell @sh	ebirrell @edit	ebirrell@ powerpoint
principals	ebirrell@sh			x	x	x
	ebirrell@edit	r,w				
	ebirrell@powerpoint		r,w			
	drdave@sh					
	drdave@edit	r				
	drdave@powerpoint		r			
	studenta@sh					
	studenta@edit					
	studenta@powerpoint		r			

Role-Based Access Control

- Particularly in corporate and institutional settings, users might be granted privileges by virtue of membership in a group.
  - E.g., students who enroll in a class should be given access to that semester's class notes and assignments simply due to their new **role**
- Without groups, implementing role-based access control is error prone
  - Adding or deleting a member might require updating many access control lists. That can be error-prone.
  - Revocation is subtle. Should permission be removed with principal is removed from a group?

Exercise 3: RBAC

- What roles might you want to include in a course management system?

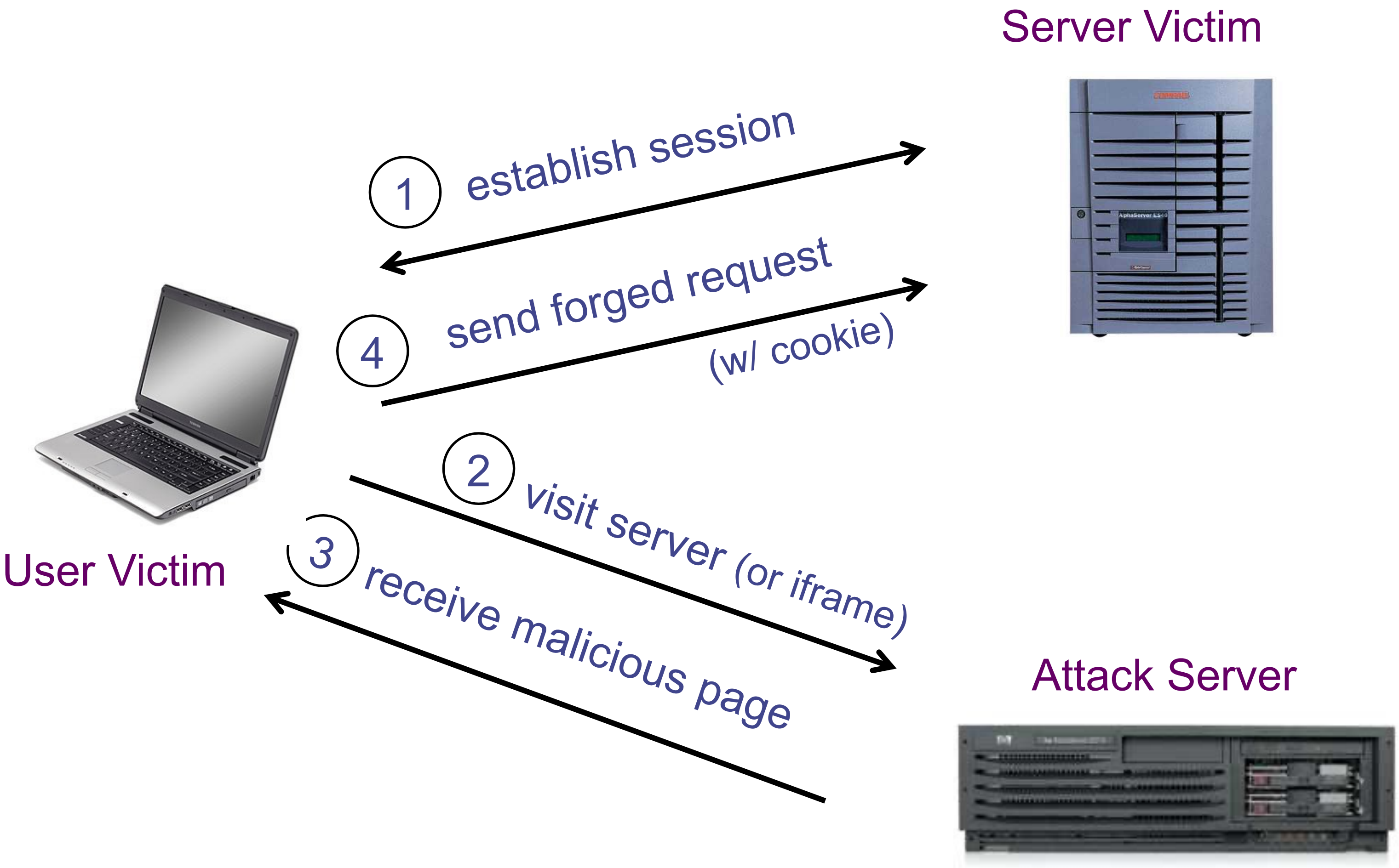
Confused Deputy

Server: operation(f : file)

```
buffer := FileSys.Read( f )
results := F( buffer )
diff:= calcDiff( results )
FileSys.Write( f , results )
FileSys.Write( log.txt, diff )
```

Privilege Escalation

Cross-Site Request Forgery (CSRF)



Access Control Policy

- An **access control policy** specifies which of the **operations** associated with any given **object** each **principal** is authorized to perform
- Expressed as a relation $Auth$:

$Auth$		Objects	
		dac.tex	dac.pptx
principals	ebirrell	r,w	r,w
	faculty	r	r
	student		r

Capability Lists

Access Control Lists

Capability Lists

- The capability list for a principal P is a list
  $$\langle O_1, Privs_1\rangle, \langle O_2, Privs_2\rangle, \dots, \langle O_n, Privs_n\rangle$$
  - e.g., ⟨dac.tex, {r,w}⟩ ⟨dac.pptx, {r,w}⟩
- **Capabilities** carry privileges.
  1) **Authorization:** Performing operation op on object O_i requires a principal P to hold a capability $C_i = \langle O_i, Privs_i\rangle$ such that $op \in Privs_i$
  2) **Unforgeability:** Capabilities cannot be counterfeited or corrupted.
- Note: Capabilities are (typically) transferable

Capabilities

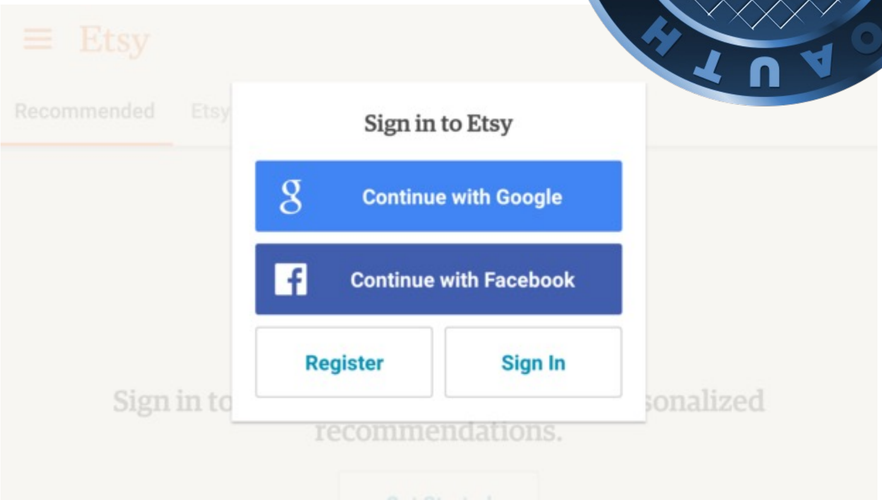
- Advantages:
- Disadvantages:

Exercise 1: Capabilities

- Consider the following proposal: capabilities will be represented using a pair $\langle Name(Obj), Privs \rangle$, where $Name(Obj)$ is a random 128-bit string and $Privs$ is the set of privileges conferred by the capability. The function $Name$, if it exists at all, is kept secret. What functionality expected for capabilities does this alternative support and where (if at all) does it fall short?

Example: OAuth2

- Industry standard authorization protocol
- Used for single sign-on by major IDPs
  - Facebook, Google
- A **bearer token** contains a unique identifier



Authenticity: Tagged Memory

1	obj	1	type	p1p2…pN

- Example: IBM System 38
- tag = 0: normal memory
- tag = 1: this word + next are a capability
- In user mode, cannot modify tag bit or modify word with tag = 1
  - Exception: can copy capabilities
- pass capabilities in function calls

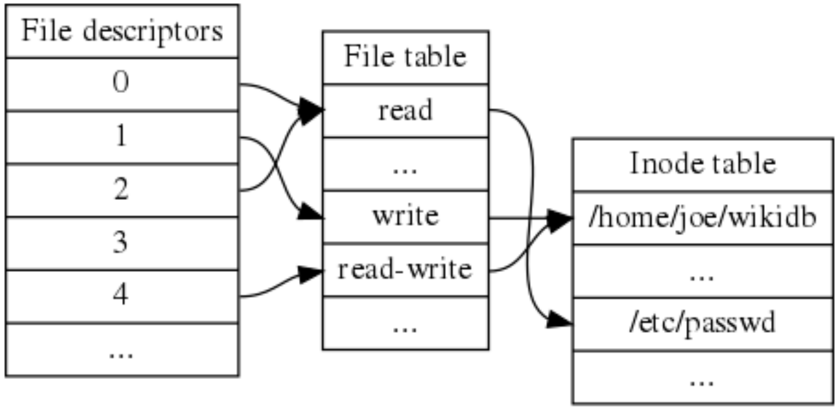
Authenticity: Protected Address Space

- General idea: store capabilities in region of memory we know how to protect
  - Option 1: protected kernel memory
  - Option 2: protected memory segment
- Note: OS must be trusted

- Store list of capabilities in process control block
- Capabilities referenced by index into c-list

Example: File Descriptor Table

- In Unix etc, a file descriptor is a handle used to reference files and I/O resources
- File descriptors have modes (read, write) and are stored in per-process file descriptor table
- File descriptors can be passed between processes using sendmsg()



Cryptographically-protected capabilities

- Object owner creates capabilities using a digital signature scheme
- Capabilities are triples $C = \langle O, Privs, \text{Sig}(O, Privs; k_O) \rangle$
- **Authorization:** P is permitted to perform op on O if P produces a capability for O with $op \in Privs$ and a valid signature
- **Unforgeability:** digital signatures are unforgeable to adversaries who don't know private key k_O

- Note: assumes PKI

Restricted Delegation

- $C_0 = \langle O, Privs_0, pk_1, \sigma_0 \rangle$
  - where $\sigma_0 = \text{Sig}(O, Privs_0, pk_1; sk_0)$
- $C_1 = \langle O, Privs_1, pk_2, (Privs_0, pk_1, \sigma_0), \sigma_1 \rangle$
  - Where $\sigma_1 = \text{Sig}(O, Privs_1, pk_2, (Privs_o, pk_1, \sigma_0); k_1)$

To Authorize op with C_0:

1. Verify σ_0 is a valid signature of $(O, Privs_0, pk_1)$
2. Check that $op \in Privs_0$

To Authorize op with C_1:

1. Verify σ_0 is a valid signature of $(O, Privs_0, pk_1)$
2. Verify σ_1 is a valid signature of $(O, Privs_1, pk_2, (Privs_o, pk_1, \sigma_0))$
3. Check that $Privs_1 \subset Privs_0$
4. Check that $op \in Privs_1$

Exercise 2: Restricted Delegation

- Assume you have a credential

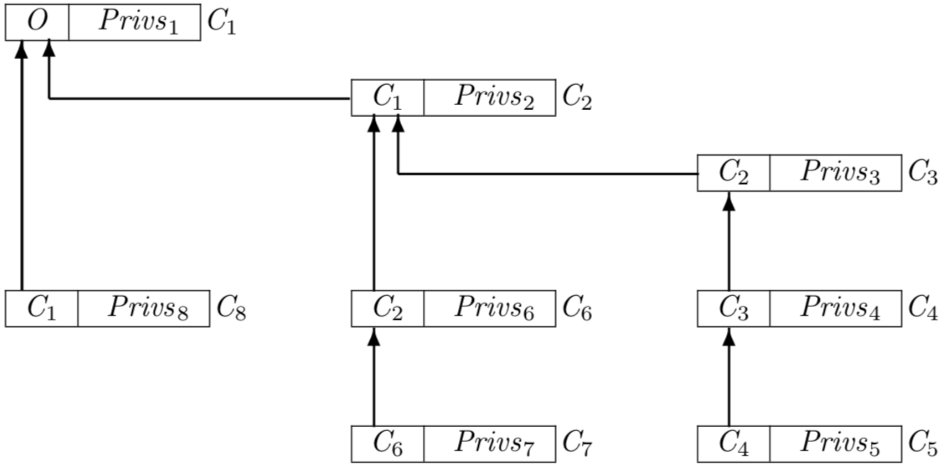
$$C_1 = \langle dac.pptx, \{r,w\}, pk_2, (\{r,w,x\}, pk_1, \sigma_0), \sigma_1 \rangle$$

1. Generate a credential C_2 that would authorized the holder to read (but not write) dac.pptx

2. Define the sequence of steps that should be taken to authorize op with C_2

Revocation

- Revocation Tags
  - Capabilities are tuples $C = \langle O, Privs, rt_c, \mathrm{Sig}(O, Privs, rt_c; k)\rangle$
  - Access to object O is guarded by a reference monitor; monitor maintains a list of revoked tags rt_c
- Capability Chains
  - Objects can be other capabilities!
  - P is authorized to perform op on O if P holds a capability C_i and $op \in Privs_k$ holds for every capability C_k in the chain from C_i to C_1

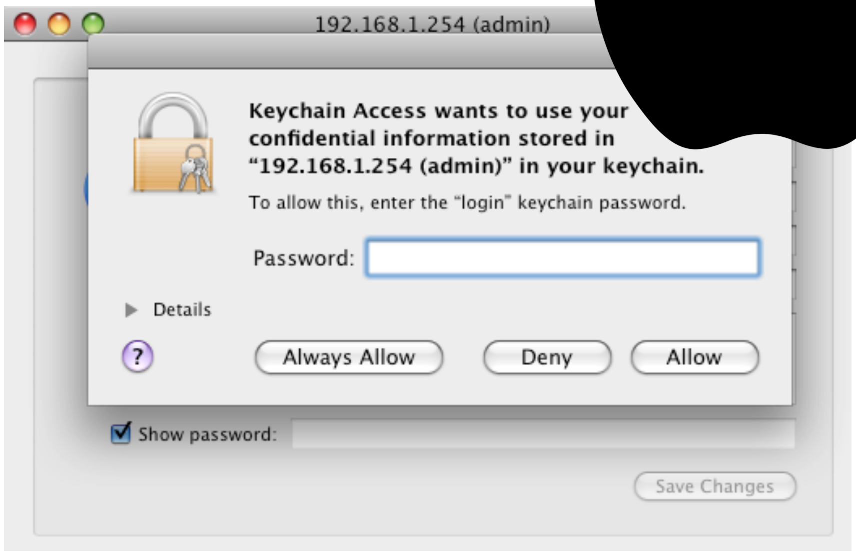


Keys as capabilities

- Encrypt object
- Decryption method functions as reference monitor:
  - **Authorization:** correct key will decrypt object -> allow access
  - **Unforgeability:** incorrect key will not decrypt
- Note: no notion of separate privileges

Example: Mac keychains

- OSX/iOS password manager
- uses password-based encryption (AES-256) to store username/password credentials
- supports multiple keychains



What about privacy?