Lecture 15: Discretionary Access Control

CS 5430

3/21/2018

Where we were…

- **Authentication:** mechanisms that bind principals to actions
- **Authorization:** mechanisms that govern whether actions are permitted
- **Audit:** mechanisms that record and review actions

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
	clarkson	r	r
	student		r

Access Control Mechanisms

- A **reference monitor** is consulted whenever one of a predefined set operations is invoked
  - operation $\langle P, O, op \rangle$ is allowed to proceed only if the invoker P is authorized to perform op on object O
- Can enforce **confidentiality** and/or **integrity**
- **Assumption:** Predefined operations are the sole means by which principals can learn or update information.
- **Assumption:** All predefined operations can be monitored (complete mediation).

Who defines authorizations?

- **Discretionary Access Control:** owner defines authorizations
- **Mandatory Access Control:** centralized authority defines authorizations

Design Principles

- **Principle of Failsafe Defaults** favors defining an access control policy by enumerating privileges rather than prohibitions.
- **Principle of Least Privilege** is best served by having fine-grained principals, objects, and operations.

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

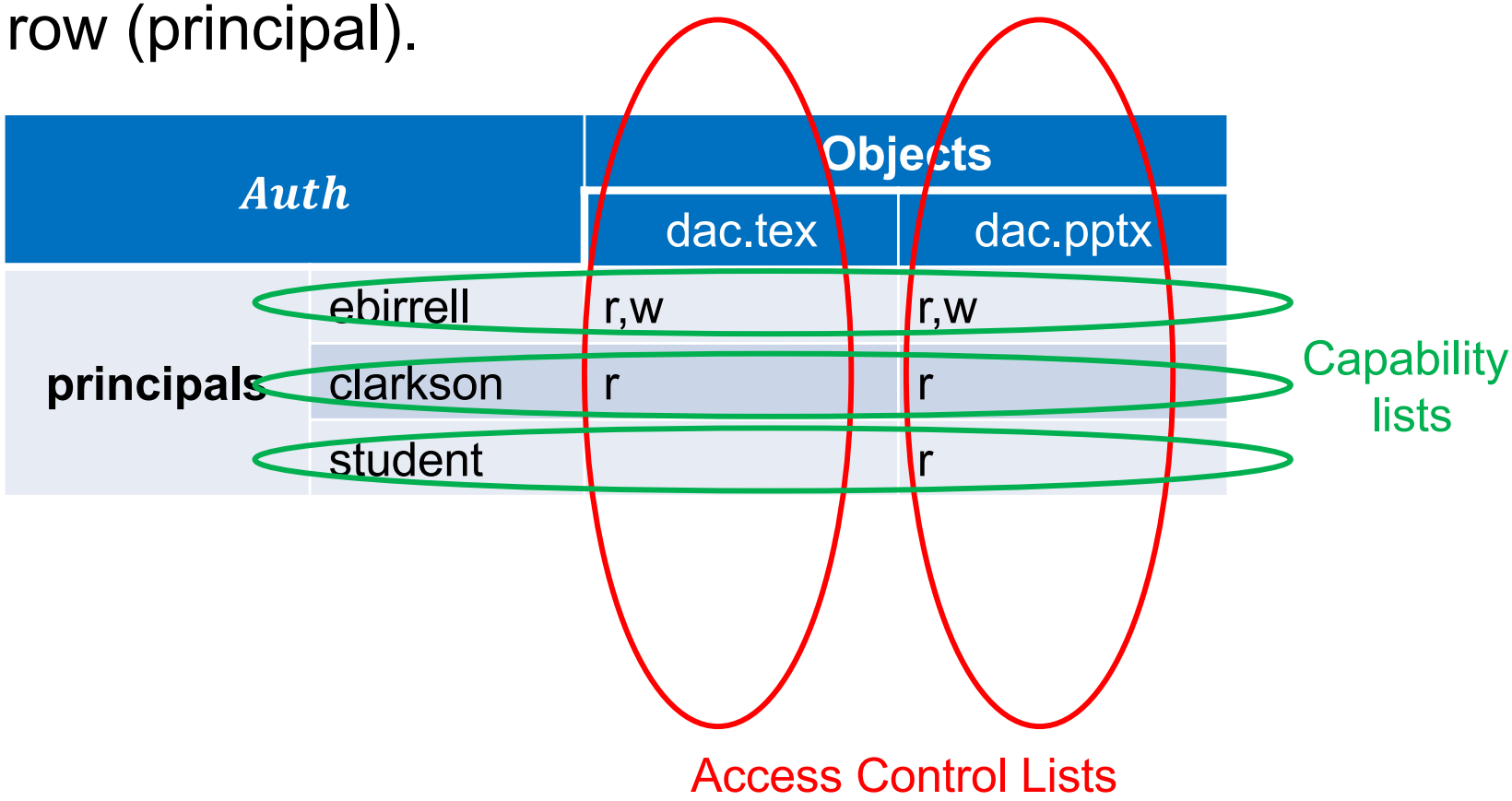
		Objects				
		dac.tex	dac.pptx	ebirrell @sh	ebirrell @edit	ebirrell@ powerpoint
principals	ebirrell@sh			e	e	e
	ebirrell@edit	r,w				
	ebirrell@powerpoint		r,w			
	clarkson@sh					
	clarkson@edit	r				
	clarkson@powerpoint		r			
	student@sh					
	student@edit					
	student@powerpoint		r			

Implementing DAC

- Need some way to representing authorization relation (matrix) $Auth$.
- That scheme must support certain functionality:
  - computing whether $\langle P, O, op \rangle \in Auth$ holds and (i.e., whether principal P is authorized to perform operation op on object O,
  - changing $Auth$ in accordance with defined commands
  - associating a protection domain with each thread of control
  - performing transitions between protection domains as execution proceeds.

Instead of Matrices…

- An access control list encodes the non-empty cells associated with a column (object).
- A capability list encode the non-empty cells associated with a row (principal).



Access Control Lists

- The access control list for an object O is a list

$$\langle P_1, Privs_1 \rangle, \langle P_2, Privs_2 \rangle, \ldots, \langle P_n, Privs_n \rangle$$

  - e.g., ⟨ebirrell, {r,w}⟩ ⟨clarkson, {r}⟩ ⟨student, {r}⟩
- To check whether P_i is allowed to perform op on object O,
  - Look up P_i in ACL. If not in list, reject op.
  - Check whether op is in the sent $Privs_i$. If not , reject op.

Access Control Lists

- Advantages:
  - Efficient review of permissions for an object
  - Centralized enforcement is simple to deploy, verify
  - Revocation is straightforward
- Disadvantages:
  - Inefficient review of permissions for a principal
  - Large lists impede performance
  - Vulnerable to confused deputy attack

Groups in ACLs

- A group declaration associates a group name with a set of principals.
- The set is specified either by enumerating its elements or by giving a predicate that all principals in the set must satisfy.
- An ACL entry $\langle G, Privs \rangle$, where G is a group name and $Privs$ is a set of privileges, grants all privileges in $Privs$ to all principals P that are members of G.

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?

Wildcards

- Many advocate terse representations for ACL entries, assuming that checking shorter access control lists is faster.
- One approach is to employ patterns and wildcard symbols for specifying names of principals or privileges, so that a single ACL entry can replace many

Prohibitions

- In order to conclude that P does not hold op for an object O, we would have to enumerate and check the entire ACL.
- Some systems allow a prohibition to appear in an ACL-entry.
  - The prohibition $\overline{op}$ specifies that execution of operation op is prohibited.
  - Conflict resolution is not always specified (often first)

Demo: Access Control Lists

```
drwxr-xr-x   5 eleanor  staff     160 Mar 21 12:14 .
drwx------+ 54 eleanor  staff    1728 Mar 21 09:45 ..
-rw-r--r--@  1 eleanor  staff   98971 Mar 21 05:15 download.png
-rwxr-xr-x   1 root     wheel  103632 Mar 21 12:14 java
-r--------@  1 eleanor  staff    2085 Mar 21 12:07 rsa-demo.pem
```

Confused Deputy

Server: operation(f : file)

S1: buffer := FileSys.Read(f)

S2: results := F(buffer)

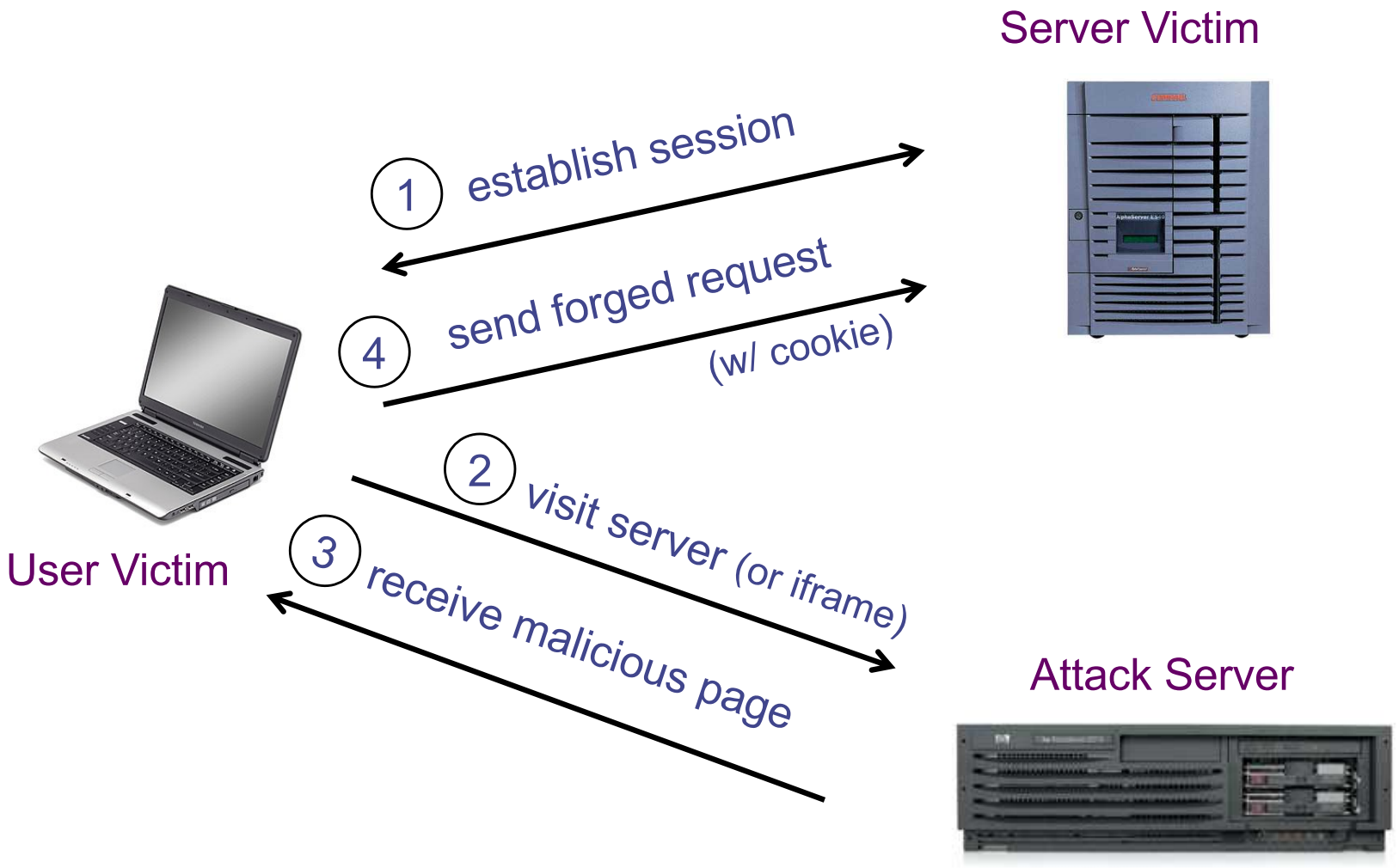
S3: diff:= calcDiff(results)

S4: FileSys.Write(f , results)

S5: FileSys.Write(log.txt, diff) end Server

Privilege Escalation

Cross-Site Request Forgery (CSRF)



Solving the Confused Deputy Problem

Server: operation(f : file)

S1: buffer := FileSys.Read(f)

S2: results := F(buffer)

S3: diff:= calcDiff(results)

S4: FileSys.Write(f , results)

S5: FileSys.Write(log.txt, diff) end Server

Capability Lists