Access Control

Dr George Danezis
(g.danezis@ucl.ac.uk)

Resources

- Key paper: Carl E. Landwehr: Formal Models for Computer Security. ACM Comput. Surv. 13(3): 247-278 (1981)
  - See references to other optional papers throughout slides.

- Ross Anderson “Security Engineering” Parts 4.1 – 4.2
- Dieter Gollmann “Computer Security” Chapter 4

- Special thanks to: Ninghui Li's course on “Access Control: Theory and Practice” (CS590U Purdue 2006)

What is “access control”?

- Access control systems are a security mechanism that ensures all accesses and actions on system objects by principals are within the security policy.

- Example questions access control systems need to answer:
  – Can Alice read file “/users/Bob/readme.txt”?
  – Can Bob open a TCP socket to “http://abc.com/”?
  – Can Charlie write to row 15 of table BILLS?
- If yes, we say they are “authorized” or has “permission”,
- If not they are “unauthorized” and “access is denied”.
- Only events within the security policy should be authorized.

- Seems like a simple enough mechanism to implement. <u>It is not.</u>

What can go wrong with Access Control?

- **Expressiveness**: How to completely express high level policies in terms of access control rules?
- **Efficiency**: Access control decisions occur often, and need to be dealt with quickly.
- **Full Mediation**: How do you know you have not forgotten some checks?
- **Safety**: How do you know your access control mechanism matches the policy?

Within top-25 CWE vulnerabilities

- **CWE-306 Missing Authentication for Critical Function**
- **CWE-862 Missing Authorization**
- **CWE-798 Use of Hard-coded Credentials**
- CWE-311 Missing Encryption of Sensitive Data
- CWE-807 Reliance on Untrusted Inputs in a Security Decision
- **CWE-250 Execution with Unnecessary Privileges**
- **CWE-863 Incorrect Authorization**
- **CWE-732 Incorrect Permission Assignment for Critical Resource**
- CWE-327 Use of a Broken or Risky Cryptographic Algorithm
- **CWE-307 Improper Restriction of Excessive Authentication Attempts**
- CWE-759 Use of a One-Way Hash without a Salt

Where does access control (usually) fits?

- (Usually) The system needs to bind the actor to a principal before authorization.
  - What is a principal? It is the abstract entity that is authorized to act.
  - Principals control users, connections, processes, …
- That is called "Authentication" (e.g. user name / password)
- The mechanisms that do authentication and authorization are in the TCB!

Mandatory and Discretionary Access Control

- Key concept: “**Mandatory Access Control**” (MAC)
  - Permission are assigned according to the security policy.
    - e.g. (Privacy) Hospital records can only be accessed by medical staff. Doctor cannot decide to give non-staff access.
  - Use within organizations with a strong need for central controls and a central security policy.

- Key concept: “**Discretionary Access Control**” (DAC)
  - All objects have “owners”.
  - Owners can decide who get to do what with “their” objects.
  - UNIX, Windows, Facebook (?)
  - Note: there is still a security policy! DAC is a mechanism.

Key Concept: The Access Control Matrix

- Consider sets of:
  - Objects (o).
  - A subset of objects called subjects (s).
  - A set of access rights (r).
- The access control matrix represents all permitted triplets of (subject, action, access right).

- Optional Reading: B. Lampson. Protection. Proc. 5th Princeton Conf. on Information Sciences and Systems, Princeton, 1971. Reprinted in ACM Operating Systems Rev. 8, 1 (Jan. 1974), pp 18-24.

An example Access Control Matrix

- Consider:
  - S: Alice, Bob
  - O: file1, file2, file3 (we omit Alice and Bob)
  - R: read, write

	file1	file2	file3
Alice	Read, write		read
Bob		Read, write	Read, write

Can Alice read file1?
Can Bob write file1?
Can Alice write file3?

Beyond “static” Access Control

- Who sets the access control matrix?
  - DAC: the owners of objects set the permissions.
- Dual role of the access control matrix:
  - Manages the rights of subjects to perform actions on objects.
  - Manages the rights subjects can give (or take) to other subjects
- The access control matrix can now change according to some rules. Which rules?

The Graham-Denning Model

- Each object has an “owner”
- Each subject has a “controller”
- A right may be transferable (with *) or not.

	Alice	Bob	file1	file2	file3
Alice	control		owner		read
Bob		control		Read, write	Owner, read

Can Alice read file1?
Can Alice read file3?
Can Bob read file3?

Graham-Denning Model: 8 Commands Creating objects and subjects

- (1) Subject x <u>creates</u> object o
  - Add column for o
  - Add (x, o, “owner”)

Objects start off being owned by whoever created them.

- (2) Subject x <u>creates</u> subject s
  - Add row and column for s
  - Add (x, s, “control”) and (x, s, “owner”)

Useful for restricting privileges (as we will see)

Graham-Denning Model: 8 Commands Destroying objects and subjects

- (3) subject x <u>destroys</u> object o
  - If (x, o, “owner”) then delete column o
- (4) subject x <u>destroys</u> subject s
  - If (x, s, “owner”) then delete column s

Only owners can delete what they own.

Graham-Denning Model: 8 Commands Granting and Transferring rights

- (5) subject x <u>grants</u> a right r/r* on object o to subject s
  - If (x, o, “owner”) then Add (s, o, r/r*)

- (6) subject x <u>transfers</u> a right r/r* on object o to subject s
  - If (x, o, r*) then Add (s, o, r/r*)

- Key concept: “Delegation”

r* – means a subject has the right to transfer the right r/r*

Graham-Denning Model: 8 Commands Deleting “own” rights

- (7) subject x deletes right r/r* on object o from subject s
  - If (x, s, “control”) or (x, o, “owner”)
  - Then Delete (s, o, r/r*)
- Note:
  - Key concept: “Revocation” – removing permissions.
  - Either x owns the object or controls the subject.

Graham-Denning Model: 8 Commands Querying

- (8) subject x <u>checks</u> what rights subject s has o object o
  - If (x, s, “control”) or (x, o, “owner”)
  - Then return (s, o, *)

- Why?
  - Does not affect the state of the matrix
  - But provides a privacy property

Exercise: Implement a least privilege policy using the Graham-Denning Model

- Aim: Alice is the owner of file1. She wants to execute an application in a process, that can only read file1. How can she use Graham-Denning to achieve this?
- Starting state:
  - (“Alice”, “file1”, “Owner”)

Solution

- (1) Starting state:

	Alice	file1
Alice	owner, Control	owner

- (2) Subject Alice creates subject Alice0

	Alice	**Alice0**	file1
Alice	owner, control	**owner, control**	owner
Alice0			

- (3) subject Alice grants a right read on object file1 to subject Alice0

	Alice	Alice0	file1
Alice	owner, control	owner, control	owner
Alice0			**read**

Question: Why do all this?

Graham-Denning Cheat Sheet

- (1) Subject x <u>creates</u> object o
- (2) Subject x <u>creates</u> subject s
- (3) subject x <u>destroys</u> object o
- (4) subject x <u>destroys</u> subject s
- (5) subject x <u>grants</u> a right r/r* on object o to subject s
- (6) subject x <u>transfers</u> a right r/r* on object o to subject s
- (7) subject x <u>deletes</u> right r/r* on object o from subject s
- (8) subject x <u>checks</u> what rights subject s has o object o

	Alice	Bob	file1	file2	file3
Alice	control		owner		read
Bob		control		Read, write	Owner, read

Could Alice read file1?

The question of Safety

- The Access control matrix needs to implement the security policy.
  - It is not the security policy, it is a security mechanism!
- Discretionary mechanisms may allow owners, or others to grant rights.
- Given a specific starting state of the access control matrix, and rules for assigning rights (like Graham-Denning), can we prove any properties of all reachable states?
  - Such as (x, o, r) will never be granted.

The Harrison-Ruzzo-Ullman Model (HRU) (Brace for some theory!)

- A general framework to define access control policies.
  - e.g. Graham-Denning
- Study whether any properties about reachable sets can be stated.
  - These are “Safety properties”
  - i.e. can a sequence of transitions reach a state of the matrix with (x, o, r)?
- Why? This would be used to build a “security argument” that the access control policy realizes some properties of the security policy!

- Optional reading: Michael A. Harrison, Walter L. Ruzzo, Jeffrey D. Ullman: Protection in Operating Systems. Commun. ACM 19(8): 461-471 (1976)

Entities in the HRU model

- The definitions of a protection system
  - A fixed set of rights R
  - A fixed set of commands C
- The state of the protection system
  - A set O of objects
  - A set S of subjects (where S is a subset of O)
  - An access control matrix defining all (s, o, r)

- Commands take the system from one state to another.

Commands in the HRU model

- The general form of a command is:
  - Command c(parameter)
    If (preconditions on parameters)
    Then (operations on parameters)

- Example: grant_read
  - Command grant_read(x1, x2, y)
    If (x1, y, “own”)
    Then enter (x2, y, “read”)

Six primitive operations in the HRU model

- Enter (s, o, r):
  - s in S and o in O
- Delete (s, o, r):
  - s in S and o in O
- Create subject s
  - s not in S
- Create object o
  - o not in O
- Delete subject s
  - s in S
- Delete object o
  - o in O and o not in S

- Exercise:
  - Define the Graham-Denning model using the HRU formalism of commands and operations.

The safety problem

- “Suppose a subject s plans to give subjects s' generic right r to object o. If we enter (s',o, r) to the current matrix, could this right r be entered somewhere else?” – Li

- Set of valid states defined by command transitions
  - Should we remove s from the matrix?
  - Should we remove “reliable” subjects from the matrix?
  - Caveats ...

The safety problem is HRU

- In the general case? Undecidable
  - We can encode a Turing machine using an HRU model
- Without delete/destroy? Undecidable
- Without create? PSPACE-complete
  - finite and enumerable states
- Single-operation?
  - Each command has a single operation in its body
  - When a subject is created it cannot be assigned any rights
  - All subjects are created equal
  - Result: Decidable

The lessons from HRU

- A deceptively simple framework for describing access control rules.
- Still impossible to build a security argument in general.
- Do not despair!
  - For some models safety can be checked.
  - In discretionary models, safety may not be such an issue.
  - Mandatory access control models more strict to avoid these problems.

Note I: access control is domain specific

- Early work focuses on operating system.
  - Objects: files, devices, OS operations, ...
  - Subjects: principals are processes, pipes, ...
- Hardware:
  - Objects: Memory pages, privileged instructions
  - Subjects: processor mode, protection domains
- Databases:
  - Objects: tables, records, rows, columns, …
  - Subjects: DB specific, e.g. stored in USERS table.
- Network:
  - Objecs: hosts, ports, nets, subnets, …
  - Subjects: principals are IP or DNS addresses, TCP connections

- Mixing domains is meaningless:
  - e.g. may not use OS access control to restrict access to a certain row of a Database.
- Yet, systems build on top of each other:
  - May need to use OS access control to restrict access to the whole DB file.
- The access control tragedy: you may need to re-implement access control at all levels of abstraction.

Note 2: How to store the Access Control Matrix?

	file1	file2	file3
Alice	Read, write		read
Bob		Read, write	Read, write

(1) Store by Column:
Key concept: “**Access control List**” (ACL)

Good: can store close to the resource.
Good: revoke rights by resource easy.
Bad: Difficult to audit all rights of a user.

(2) Store by Row:
Key Concept: “**Capability**”

Good: Store at the user.
Good: Can audit all user permissions.
Bad: Revocation, transferability, authenticity?

More to capabilities that a row representation! (More later)

(3) Through compact representations or redirection: key and lock, labels, roles, groups, multiple levels of indirection, … (see RBAC later)

Key concept: “The reference monitor”

- Definition: the part of the systems (usually OS) that enforces access control decisions.
- 3 properties:
  - Complete mediation: must always be called.
  - Tamper proof: adversary cannot influence it (in the TCB!)
  - Small: to verify its correctness.

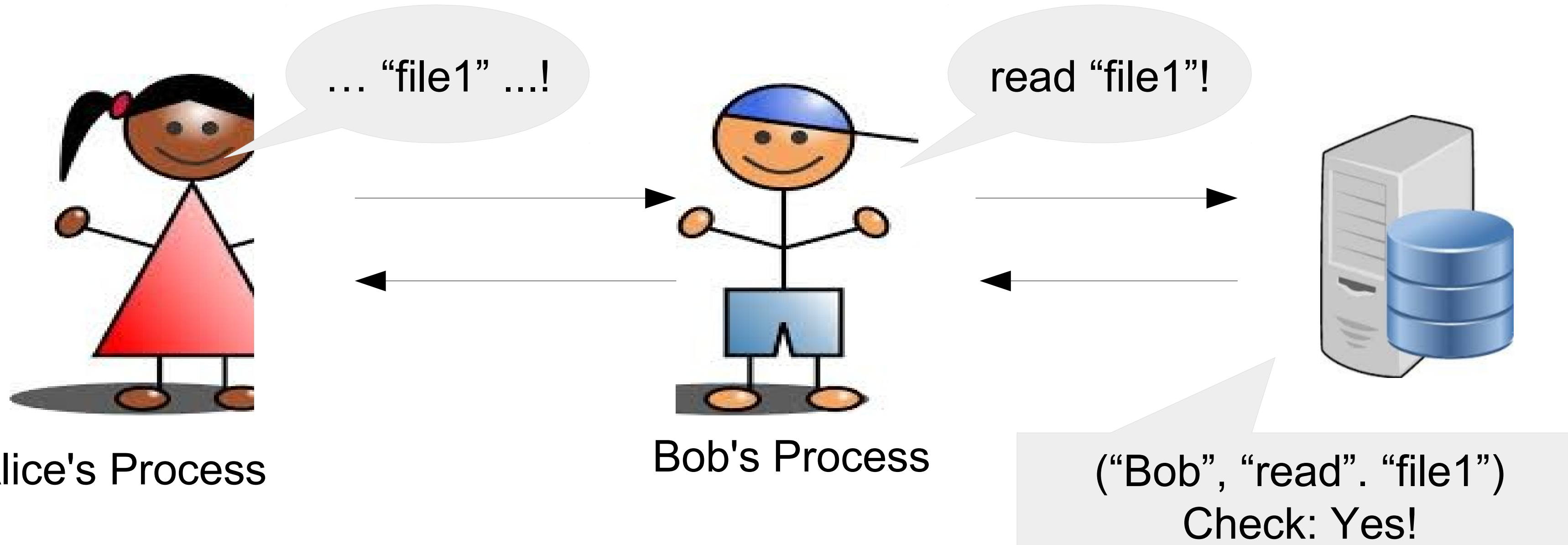
- Optional historical reading: Anderson, J. 'Computer Security Technology Planning Study', ESD-TR-73-51, US Air Force Electronic Systems Division (1973). Section 4.1.1

Key Concept: “Ambient Authority”

- An implementation strategy for access control.
- Definition: The “principal” (authority) is implicit from some global property of process.
  - “authority that is exercised, but not selected, by its user” (Shapiro et al.)
  - Example: open(“file1”, “rw”)
    (Note: the subject is missing, but inferred from the process owner)
- Upside:
  - no need to repeat all the time the subject.
- Downside:
  - least privilege harder to enforce.
  - Confused deputy problem.

The Confused Deputy

- Alice (OS user) asks Bob (OS server) to read a file1, and give her the content nicely formatted.



```python
#!python
import glob
import os.path

import cherrypy ## Need cherrypy web framework
from cherrypy.lib.static import serve_file

class Root:
    def index(self, directory="."):
        html = """<html><body><h2>Here are the
        files in the selected directory:</h2>
        <a href="index?directory=%s">Up</a><br />
        """ % os.path.dirname(os.path.abspath(directory))

        for filename in glob.glob(directory + '/*'):
            absPath = os.path.abspath(filename)
            if os.path.isdir(absPath):
                html += '<a href="/index?directory=' + absPath + '">' \
                    + os.path.basename(filename) + "</a> <br />"
            else:
                html += '<a href="/download/?filepath=' + absPath + '">' \
                    + os.path.basename(filename) + "</a> <br />"

        html += """</body></html>"""
        return html
    index.exposed = True

class Download:
    def index(self, filepath):
        return serve_file(filepath, "application/x-download", "attachment")
    index.exposed = True

if __name__ == '__main__':
    root = Root()
    root.download = Download()
    cherrypy.quickstart(root)
```

Case Study:

cherrypy web framework documentation, on how to implement file downloads

(1) What is going on here?
(2) Find the security bug.
(3) Why is this a case of a confused deputy?
(4) How do you fix it?

From http://docs.cherrypy.org/stable/progguide/files/downloading.html

Case Study: The UNIX suid mechanism

- In UNIX "everything is a file".
  Coarse grained ACL:
  - Principals: "user", "group", "world".
  - Rights: read, write, execute.
  - Programs execute with the permissions ("effective userid") of "caller".
  - Access control: compare the "effective userid" with the quasi-ACL.

- But how to implement a database?
  - Alice needs to write in some records but must not on others.

- Solution: suid bit permission
  - The program executes with the permission of the "owner" not the "caller".
  - Confused deputy problem … (and other problems).

How to avoid confused deputies?

- Problem is very real:
  - In systems with ambient authority it is difficult to express that an action is taking place "*on behalf*" of another principal.
  - Examples: web servers, system utilities, …
- Solutions:
  - Re-implement access control in Bob's process (usual)
  - Allow Bob to check authorization for Alice.
  - Capability-based architectures may help...

Bob
in TCB!

Capability based architectures

- 3 models of capability systems:
  - Capabilities as Access Matrix rows (ACLs as columns)
  - Capabilities as physical keys or tickets
  - Full object-capability models
- Key paper: Miller, Mark S., Ka-Ping Yee, and Jonathan Shapiro. Capability myths demolished. Technical Report SRL2003-02, Johns Hopkins University Systems Research Laboratory, 2003.

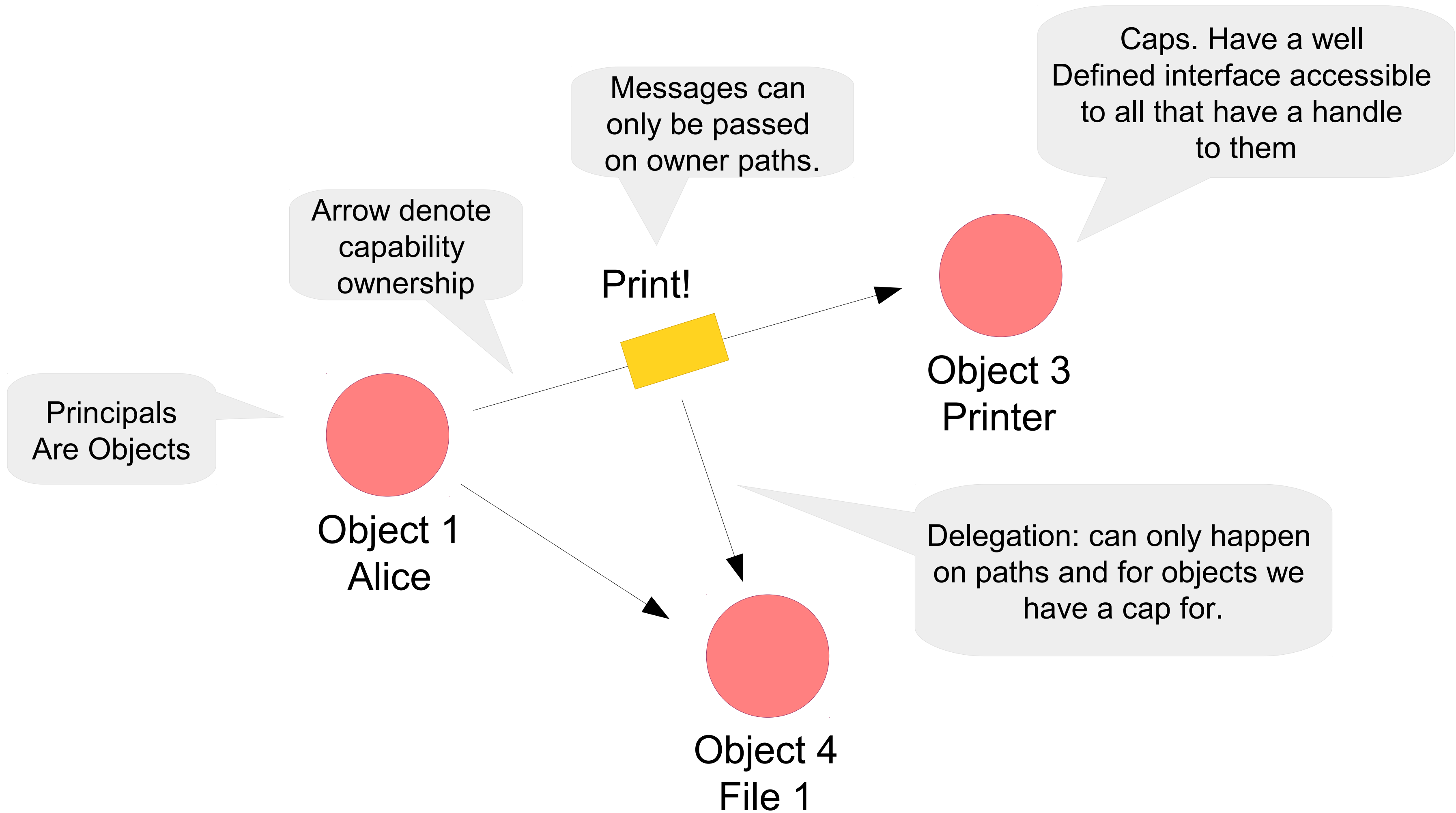
Controversies with Capabilities

- Revocation:
  - If capabilities are like “tickets” in the hands of subjects, how can they be revoked (e.g. by owner)?
- Delegation:
  - If capabilities are like tickets, and are first class objects (i.e. can be referred to and passed as arguments), how can we restrict delegation?

The object-capability model

- Model:
  - Objects interact only by sending messages on references
  - References are unforgeable (managed by TCB!)
  - A reference can be obtained by:
    - Through initialization of process.
    - Parenthood: References to created objects are known to object/subject creator.
    - Endowment: Given by object parent (if they have one)
    - Introduction: If A has ref to B and C, A can send B a message to B with ref. C. B keeps it for future use.
- Examples:
  - Close to: Java object references! (Except: globals, libraries, etc.)

Example: Object Capabilities



Seven properties of an access control mechanism implementations

Useful to understand ACLs, and different Capability models

Property	Quick test
A. No Designation Without Authority	Does designating a resource always convey its corresponding authority?
B. Dynamic Subject Creation	Can subjects dynamically create new subjects?
C. Subject-Aggregated Authority Management	Is the power to edit authorities aggregated by subject?
D. No Ambient Authority	Must subjects select which authority to use when performing an access?
E. Composability of Authorities	Are resources also subjects?
F. Access-Controlled Delegation Channels	Is an access relationship between two subjects X and Y required in order for X to pass an authority to Y?
G. Dynamic Resource Creation	Can subjects dynamically create new objects?

What are the differences between cap. systems?

	ACL	Cap. as row	Cap. as keys	Object cap.
A. No Designation Without Authority	No	Maybe	No	Yes
B. Dynamic Subject Creation	Not usually	Yes	Yes	Yes
C. Subject-Aggregated Authority Management	Not usually	Yes	Yes	Yes
D. No Ambient Authority	No	No	Yes	Yes
E. Composability of Authorities	Maybe	Maybe	No	Yes
F. Access-Controlled Delegation Channels	Maybe	Maybe	No	Yes
G. Dynamic Resource Creation	Yes	Yes	Yes	Yes

How to implement a revocation?

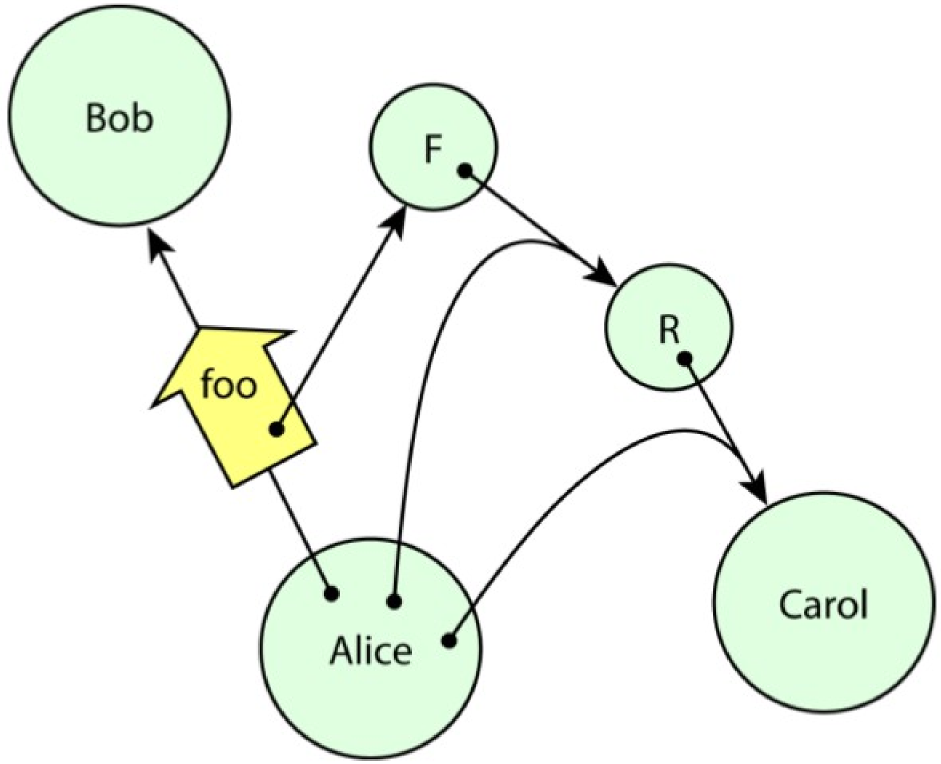


Figure 6. Alice provides Bob with revocable access to Carol.

Why? (Miller, Yee and Shapiro)

Conclusions

- Where next?
  - Implementation strategies.
  - Policy Definition Languages (e.g. SecPAL).
  - Static / Dynamic checks for efficiency.
  - Distributed Access control?

- Access control is the workhorse of industrial security systems.
  - Mechanism not policy.
  - Safety is hard to determine in general.
  - Implementation and programming models as ACL / Cap opens up different possibilities and attacks.