

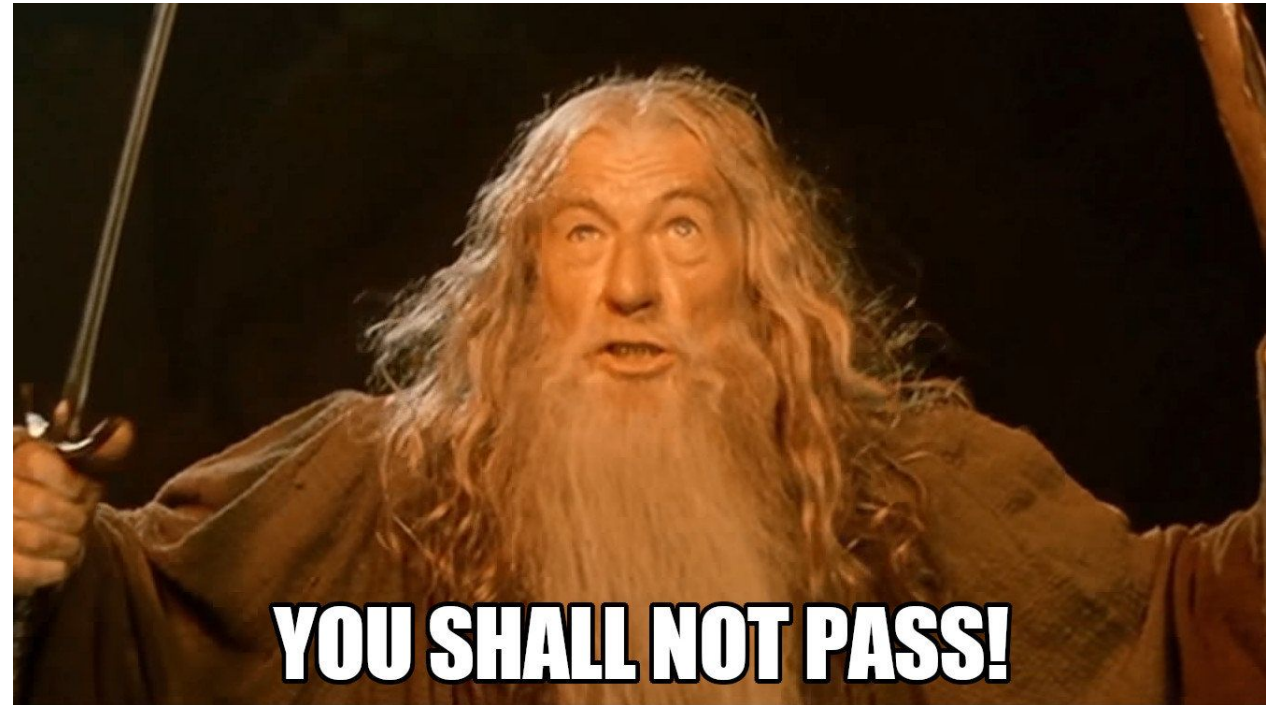
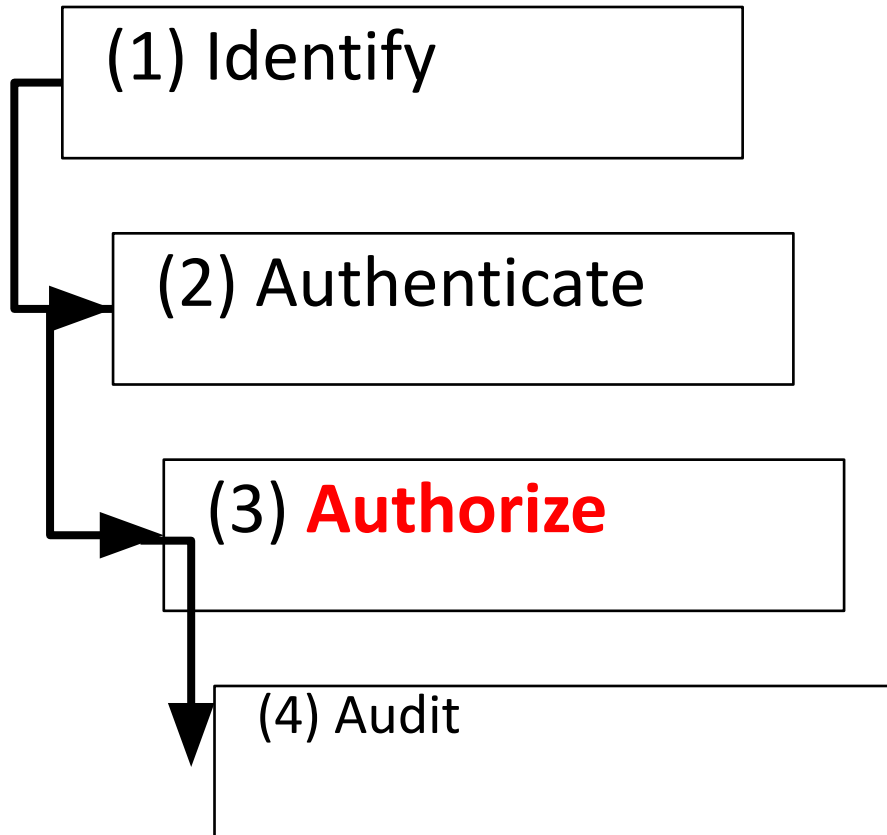
Introduction to Computer Security Lecture Slides

© 2024 by [Mihai Chiroiu](#) & [Florin Stancu](#)

is licensed under [Attribution-NonCommercial-ShareAlike 4.0
International](#)

Authorization, Access Control, *Operating System Security*

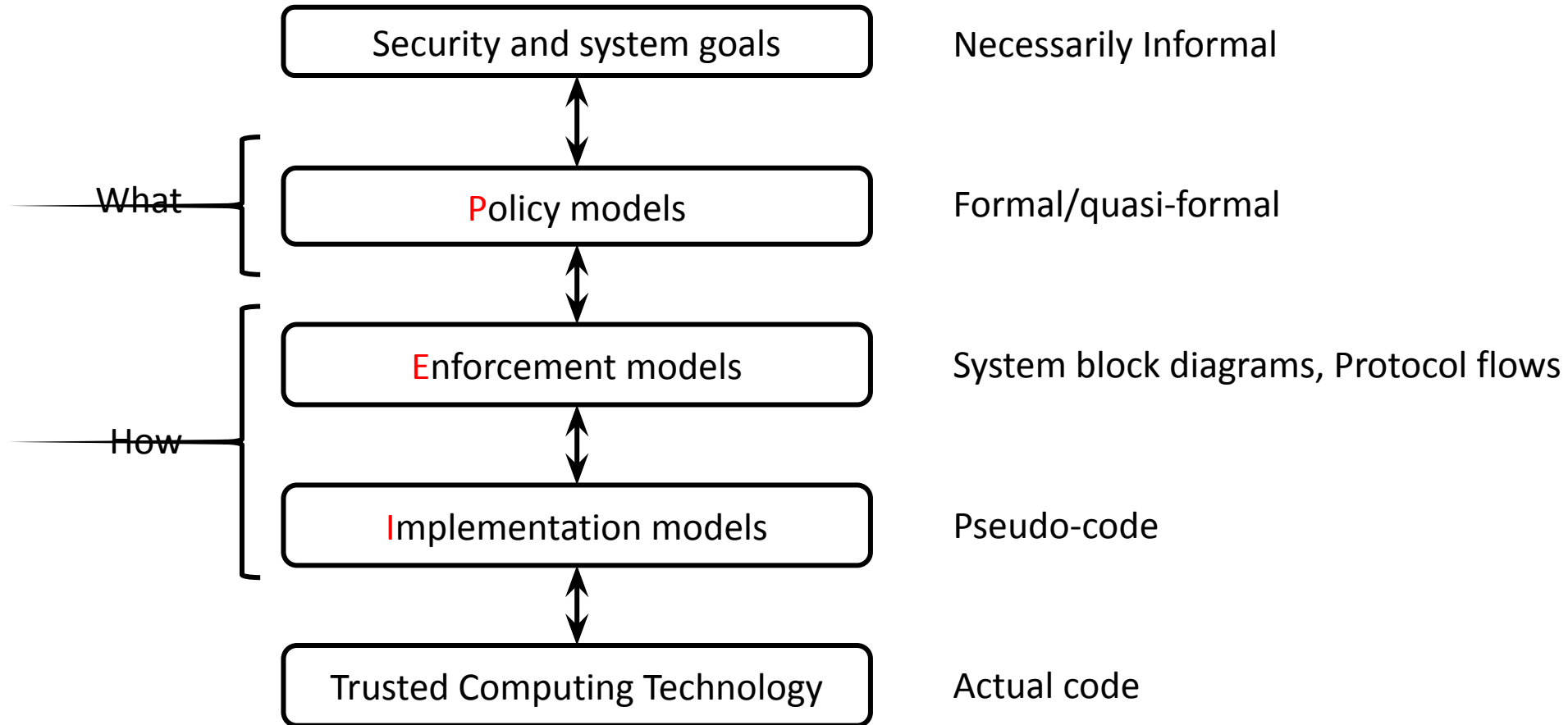
Access control



Examples of Access Control

- Social Networks: access to personal information.
- Web Browsers: access only to a website (same origin policy).
- Operating Systems: one user cannot arbitrarily access/kill another user's files/processes.
- CPU Memory Protection: code in one region (e.g., Ring 3), cannot access the data in another more privileged region (e.g. Ring 0).
- Firewalls: If a packet matches with certain conditions, it will be dropped.

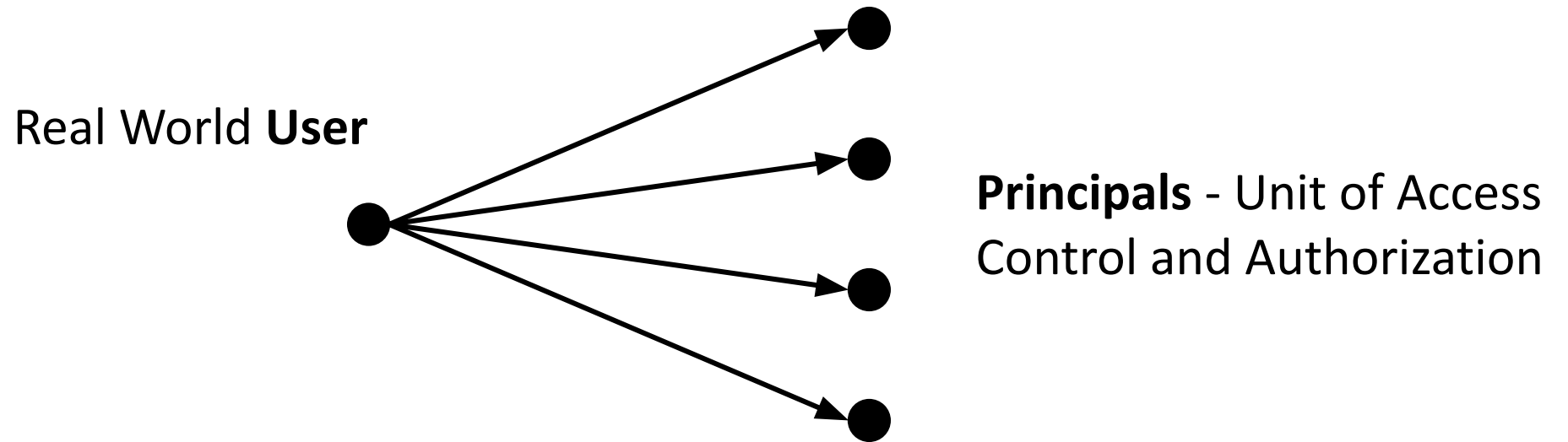
PEI Model [1]



Vocabulary

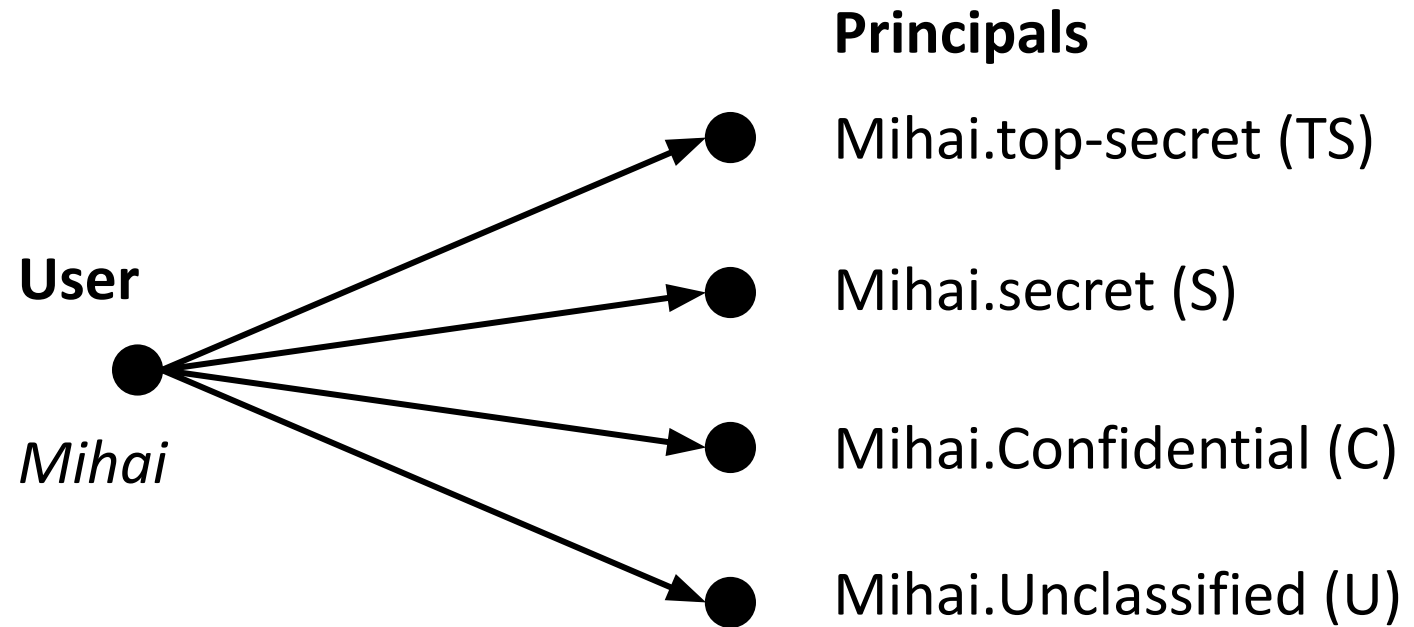
- Basic abstractions:
 - Subjects
 - Objects
 - Rights
- A **subject** is an entity who wishes to access a certain **object**, which is a resource (e.g., a file or a network packet). The different modes of access (e.g., reading, writing) are called **permissions**.

Vocabulary – Users and Principals



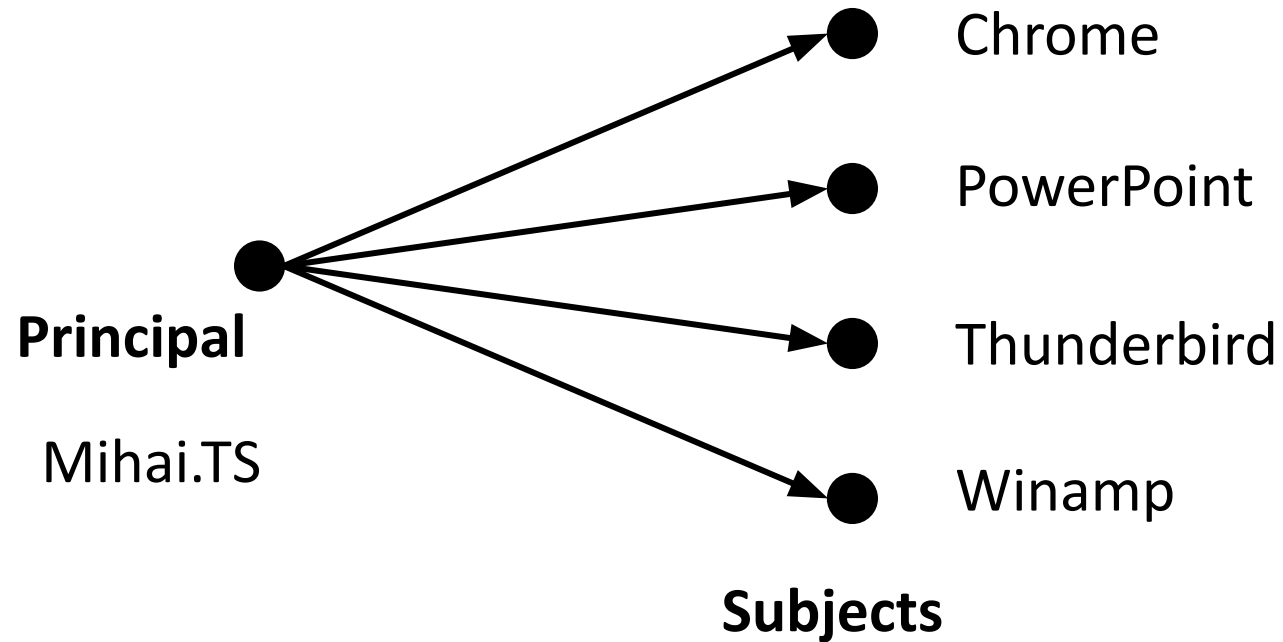
A Principal is an User authenticated in a context

Vocabulary – Users and Principals



Example: the user can emit principals with downgraded privileges

Vocabulary – Principals and subjects



A subject is a program executing on behalf of a principal

Vocabulary

- The relation between Users and Principals is One-To-Many
 - Allows accountability of user's actions, use least privileges required for a task
 - E.g., service accounts / API keys (authentication w/o password)
- For simplicity, a principal and subject can be treated as identical concepts.

Vocabulary - Objects

- An object is anything on which a subject can perform operations (mediated by rights)
- Usually objects are passive, for example:
 - File
 - Directory (or Folder)
 - Memory segment
- But, subjects (e.g., processes) can also be objects, with specific operations
 - kill
 - suspend
 - resume

Access control models

Access control enforcement

- **Discretionary access controls (DAC)** – the access of objects (or subjects) can be propagated from one subject to another. Possession of an access right by a subject is sufficient to allow access to the object.
- **Mandatory access controls (MAC)** – the access of subjects to objects is based on a system-wide policies (based on security labels) that can be changed only by the administrator.
- **Role-Based Access Control (RBAC)** – can be configured as both MAC or DAC, access to objects is based on roles.

Access control enforcement

- **Attribute-Based Access Control (ABAC)** – properties of an object are used when usage decision are made.
- **Usage Control (UCON)** – generalization of access control to include authorization, obligations, conditions (e.g., quotas), continuity and mutability of attributes.

Discretionary access controls

DAC

- No precise definition.
- Basically, DAC allows access rights to be propagated at subject's discretion
 - often has the notion of owner of an object
 - used in UNIX, Windows, etc.

Formal rule representation

- Let S be the set of all subjects, O the set of all objects, and P the set of all permissions. The description of access control can be given by a set $A \subseteq S \times O \times P$.
- When new permissions are added, triplets are added to A ; when they are removed (revoked), triplets are deleted.

Matrix Representation

- An access control matrix is a matrix $(M_{s,o})$ whose rows are subjects and columns are objects. Element $(M_{s,o}) \subseteq P$ is the set of permissions that subject **S** is authorized for object **o**.

————— Objects (and Subjects) —————→

		catalog.csv	cat-anonim.txt	/dev/kmem	/sbin/sudo	PID 1001
Subjects ↓	Root	rw	rwX	-	rwX	<i>kill</i>
	mihai	rw	rw	-	rx	<i>kill</i>
	student	-	r	-	rx	-
	guest	-	-	-	-	-

Access Control Lists (ACL)

- An access control list is a set $\{A_o \mid o \in O\}$, one element for each **object**. The elements of the list are the pairs (s, p) of **subjects** s who have **permission** p to that object.

catalog.csv
root: rw
mihai: rw

cat-anonim.txt
root: rw
mihai: rw
student: r

/sbin/sudo
root: rwx
mihai: rx
student: r

Classic POSIX Model

- **Objects:** files; **Permissions:** R, W, X + specials (SUID/SGID/sticky)
- **Subjects:** **users**, **groups**, **others**
- Stored as bit masks (written in base 8 – octal) on **inodes**

→ `ls -l /usr/bin/ping`

`-rwxr-xr-- 1 root admin 92K Jan 18 08:05 /usr/bin/ping`

Bit mask: `111|101|100` => octal `754`

- Modern OSes support Full Access Control Lists
 - **multiple subjects!**

Capability Lists

- Alternate access control implementation: each user stores a list of his/hers **capabilities** instead of objects' storing ACLs.
- Storing capabilities means giving to each subject tokens which give them access to the permissions they are entitled

	catalog.csv	cat-anonim.txt	/dev/kmem	/sbin/sudo	PID 1001
Root	rw	rw	rw	rwX	<i>kill</i>
mihai	rw	rw	-	rx	<i>kill</i>
student	-	r	-	rx	-
guest	-	-	-	-	-

Capability examples

- **Posix API** descriptors:

```
int fd = open("/etc/passwd", O_RDWR);
```

Code flow: `fork()` -> `setuidgid()` -> `exec()`

-> *new process inherits **fd** (the authorization “token”)*

- **Linux:** per-process *capabilities*
- **Windows:** Security Identifier (SID) on Active Directory

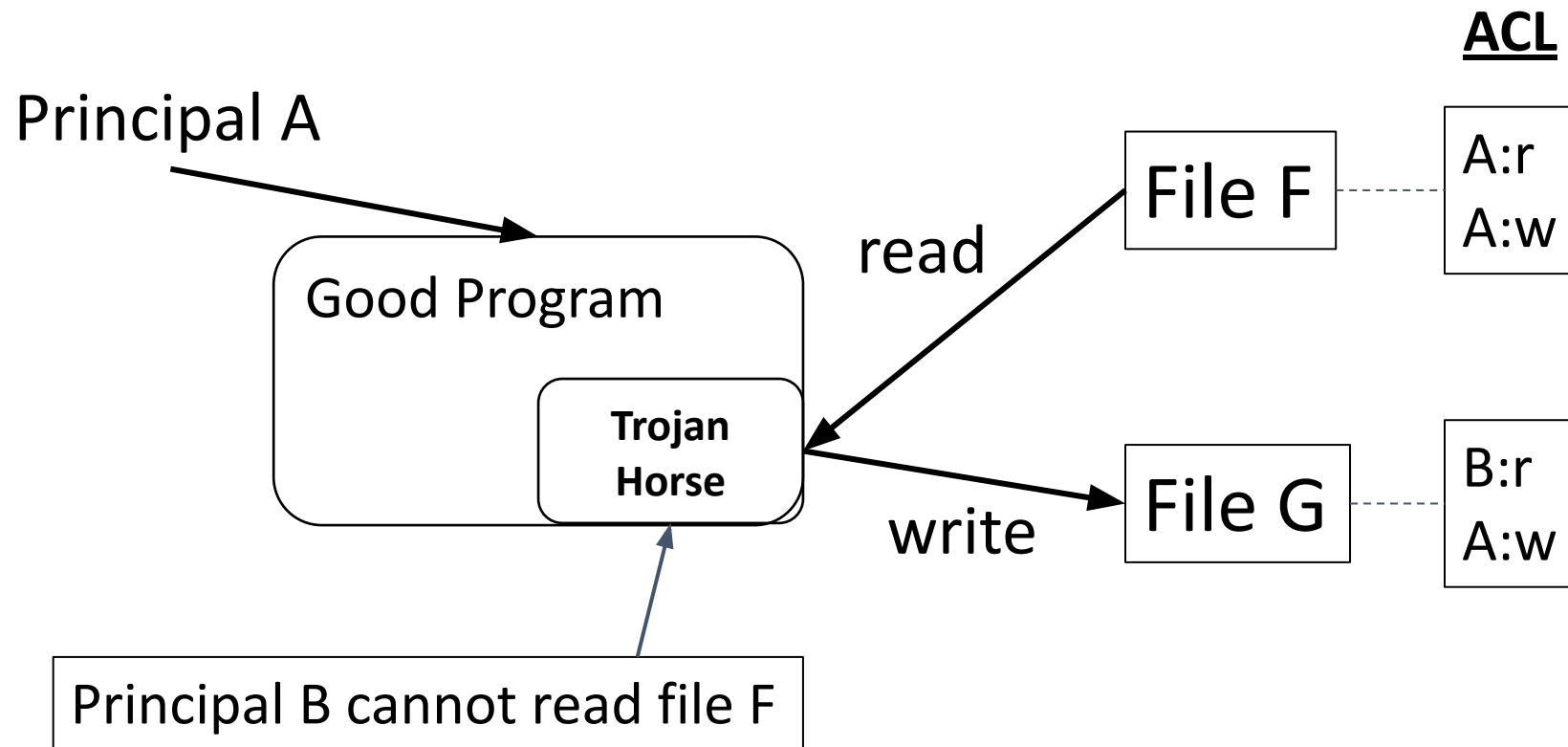
ACL vs. Capabilities

- ACL require authentication of subjects
- Capabilities do not require authentication of subjects, but do require unforgeability and control of propagation of capabilities. Usually implemented through cryptography.
- The Confused Deputy Problem [1986]
 - E.g.: Cross-Site Scripting / Forgery (XSS / CSRF), *setuid* privilege escalation (e.g., sudo)
 - Solution: bundle resource access together with capability

DAC Problems

- The underlying philosophy of DAC is that subjects can determine who has access to their objects
 - There is a difference, though, between trusting a person and trusting a program
- The copies of a file are not controlled
- Trojan Horse attack [1970]
 - Solution: use MAC 😊

Trojan Horse attack



Buggy software can become Trojan Horses

- When a subject (e.g., buggy software) is exploited, it executes the code / intention of the attacker, while using the privileges of the user who started it!
- This means that DAC-only systems cannot be trusted with classified information!

Principle of Least Privilege

- Each subject should have only necessary privileges!
- Privilege elevation / dropping
 - Unix: `setuid()` / `setgid()` family of system calls
- Example POSIX scenario:
 - Only root can open ports ≤ 1024
 - Web server (e.g., apache2) starts as root
 - Opens log files, sockets etc. when root then drops all root privileges (user changes to **www-data**)!
 - **Modern alternative**: Linux capabilities (`CAP_NET_BIND_SERVICE`)
- Better yet: DAC + Mandatory Access Control!

Mandatory access controls

Mandatory Access Control

- Assigning access rights based on regulations by a central authority
- Implemented using a *“reference monitor”*
 - Small Trusted Computing Base (TCB) [John Rushby, 1981, OSP]
 - Kernel < Hypervisor < Hardware
- TOCTTOU (Time Of Check To Time of Use) problem:
 - authority checks access to an object
 - *unknowingly to him, attacker replaces object with another one*
 - privileged subject operates on attacker controlled object!

MAC implementations

- Type Enforcement (e.g.: SELinux)
 - Subjects => grouped in domains (*labels*)
 - Objects => grouped in types (*another / same kind of labels*)
 - Domain-Domain + Domain-Type permissions
- If a MAC rule fails => DAC not checked, access denied!

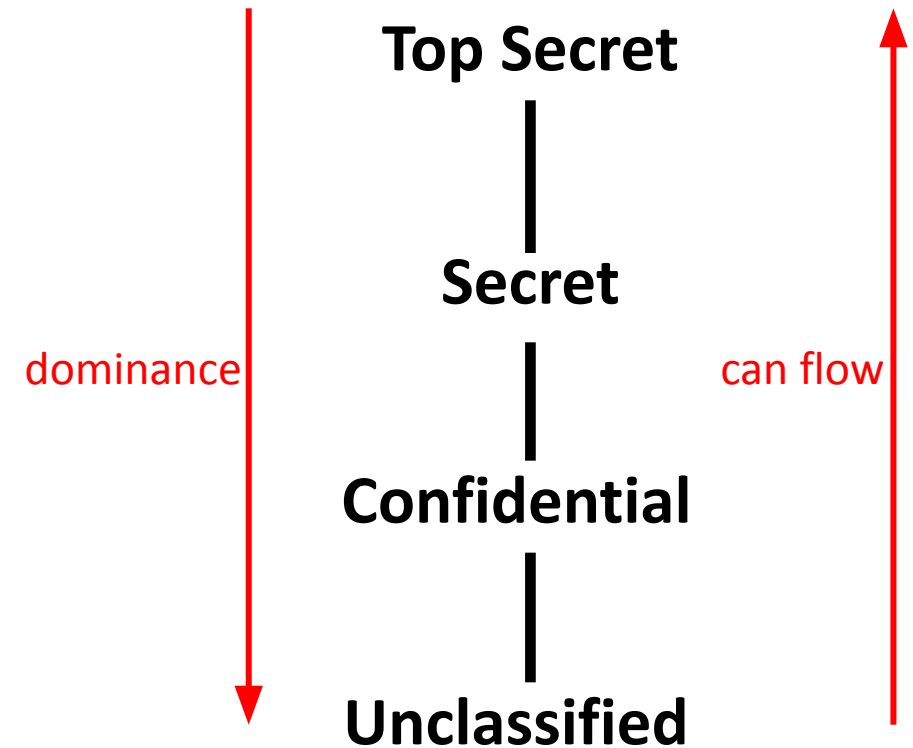
```
# TE rule: allow passwd_t shadow_t : file {read, write ...}
# ls -Z /etc/shadow
-r----- root    root    system_u:object_r:shadow_t  shadow
# ps -aZ
gigel:user_r:passwd_t    16532 pts/0    00:00:00 passwd
```

Modeling Access Control

- Multi-level security (MLS)
 - Bell-LaPadula (BLP)
 - Biba Model
- Chinese Wall

Multi-level security (MLS)

- The capability of a computer system to carry information with different sensitivities
- Bell-LaPadula (BLP) Model [1973]
- Biba Model



BLP Model

- Aims to capture confidentiality (read) requirements only
- Modelled as transitions through a set of states, starting from an initial state.
 - State = Object, access matrix, current access information
- State transition rules describe how a system can go from one state to another
- Each object has a classification level
- Each subject s has a security clearance

BLP Model

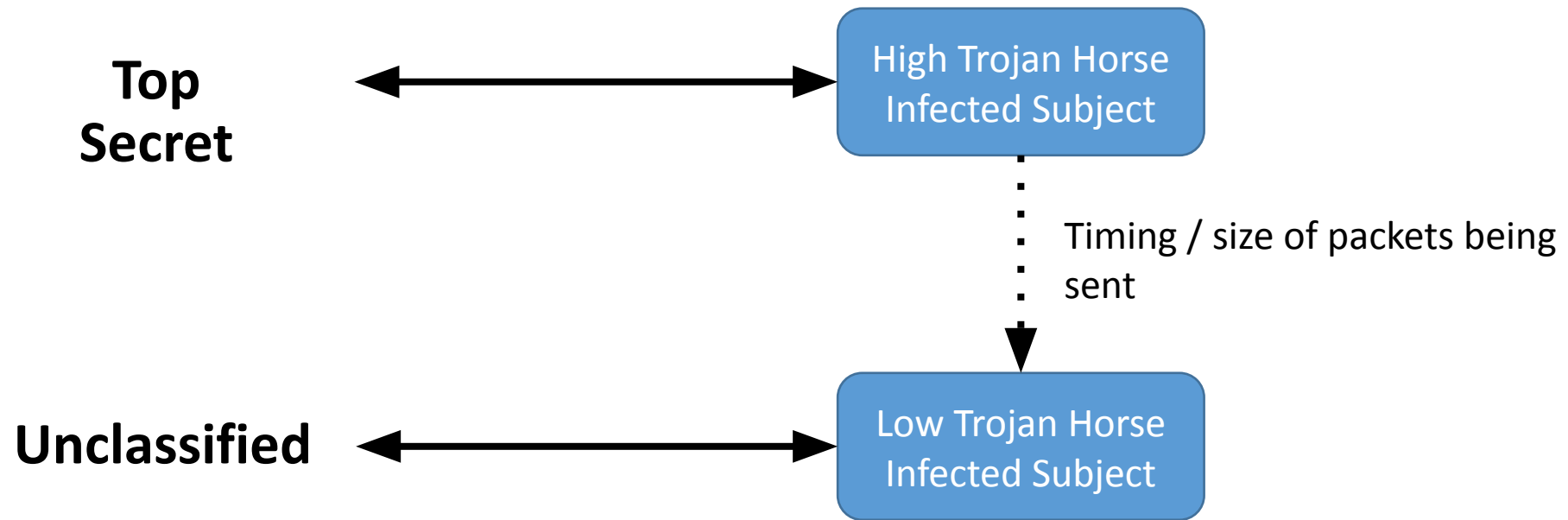
- A state is secure if:
 - A) Simple Security Property (SS): no subject may read data at a higher level
 - B) The *(Star)-Property (SP): no subject may write data at a lower level
(due to the fear of Trojan Horse / information leaks)
- A system is secure if and only if every reachable state is secure.

BLP Problems

- No communication (e.g., acknowledges) from High to Low
- Not all system components can be enforced by BLP, e.g., memory management must have access to all levels
 - Called “*trusted subjects*” (part of TCB)
- Can overwrite high and more important files
 - Prevent overwrites unless same level!

BLP Problems

- Covert channels cannot be blocked by star-property



Biba Model

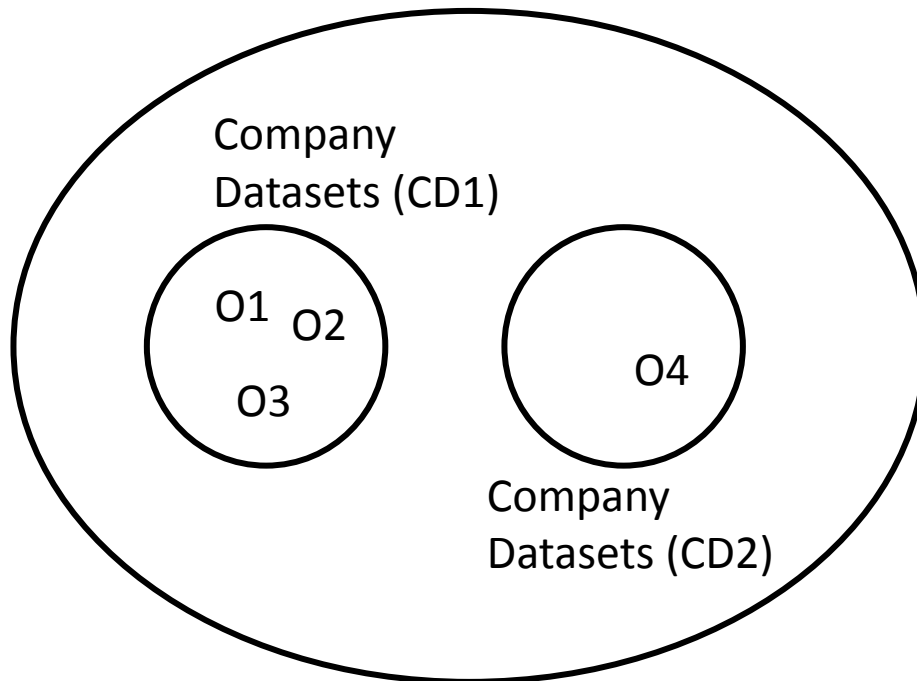
- Integrity is also very important
- Each subject (process) has an integrity level; Each object has an integrity level ; Integrity levels are totally ordered
- NO read down; NO write up
 - BLP upside down
- The integrity of an object is the lowest level of all the objects that contributed to its creation

Biba Model

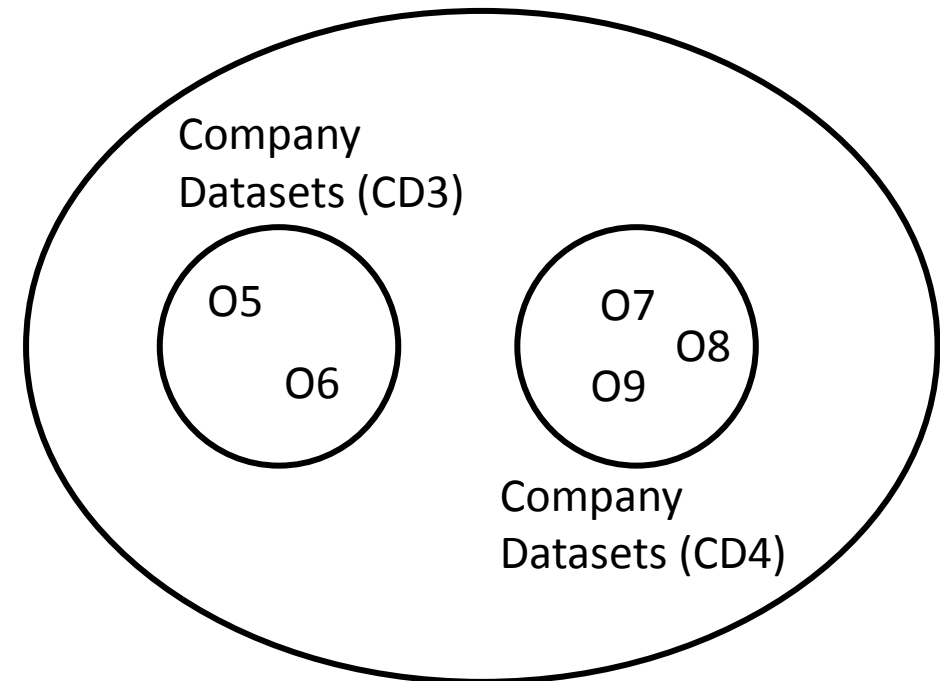
- Used by Windows
- E.g., A Internet Explorer Browser can download a file (created with a low integrity level) and read everything in the system. It cannot write to a higher level object.

Chinese Wall (Brewer and Nash model) [1989]

Conflict of Interest Classes (CIC)



Conflict of Interest Classes



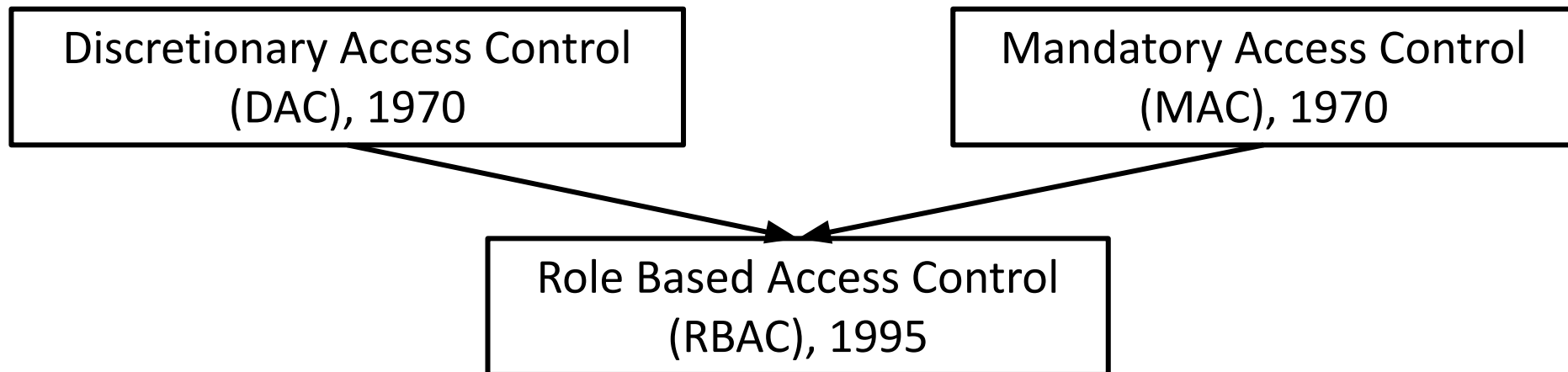
Chinese Wall

- S can read O only if
 - O is in the same company dataset as some object previously read by S (i.e., O is within the wall) **or**
 - O belongs to a conflict of interest class within which S has not read any object (i.e., O is in the open)
- S can write O only if
 - S can read O by the simple security rule **and**
 - no object can be read which is in a different company dataset to the one for which write access is request

Role-Based Access Control

Role-Based Access Control

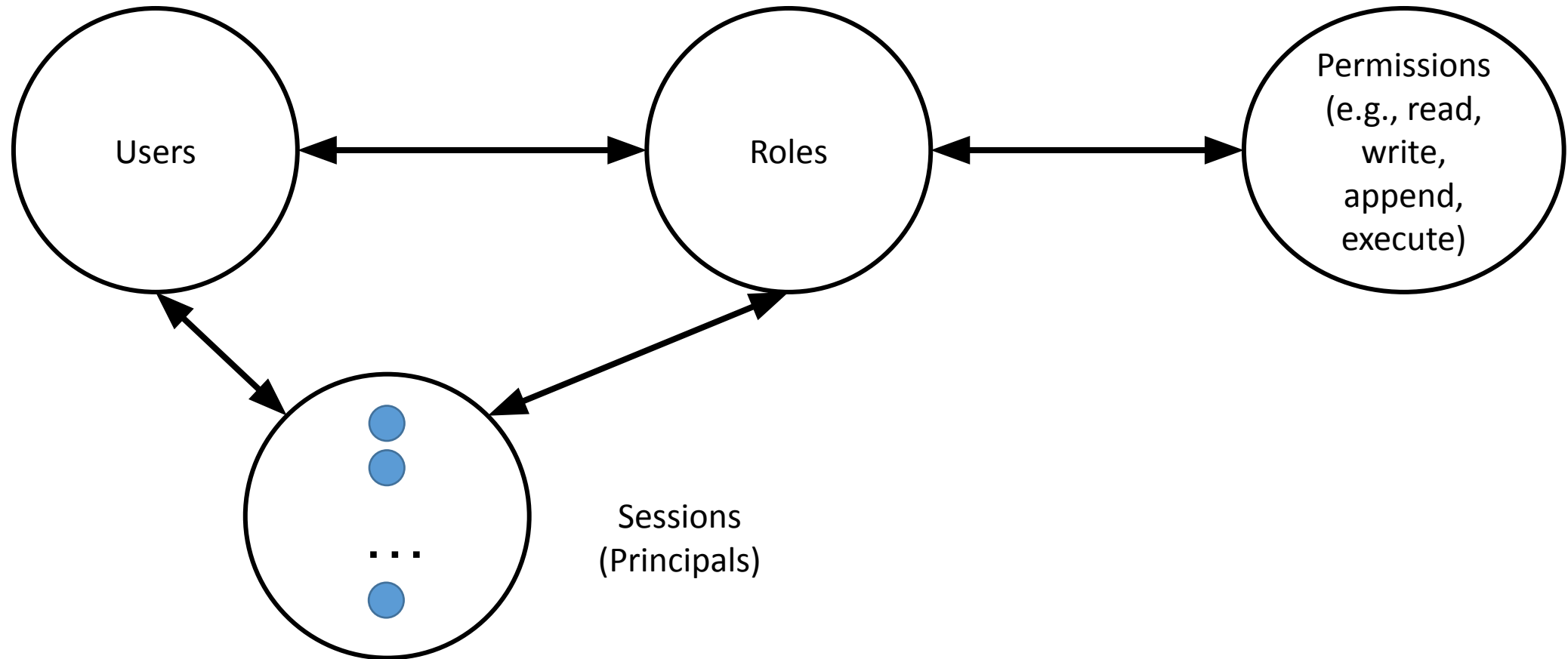
- In the real world, security policies are dynamic.
- E.g., a user promotes at his job, therefore his rights must change (deleted, added, etc.)
- RBACs are more flexible: can simulate MAC & DAC!



Roles as policy

- A role brings together
 - a collection of users
 - a collection of permissions
- These collections can be modified independently
- A user can be a member of many roles
- Each role can have many users as Each role can have many users as members
- Roles may be hierarchical

Role-Based Access Control



RBAC Shortcomings

- Role granularity may lead to role explosion
- Role design and engineering is difficult and expensive
- Assignment of users/permissions to roles is cumbersome
- Adjustment based on local/global situational factors is difficult

Authorization Implementations

OAuth

- Open **Authorization**, not **Authentication**!
- Users delegate API access to third party services without giving password!
 - e.g. give Google Calendar API access to task management app
- JSON Web Token (JWT) – may contain subject IDs + capability lists
- Authorization flow: client / server-side
- OpenID Connect: OAuth popular choice for **SSO authentication**!
 - Obtain token with read-only access to Google API endpoint returning your email address => third-party service identifies you!

Writing Authorization Code

- Tons of conditionals?

```
if is_admin or (can_read(obj.parent) and can_write(obj)) ...
```

- Solution: use language features & authorization frameworks

```
@authorize.create(Article)
def create_article(name):
    # implementation here
```

```
@authorize.read
def read_article(article):
    # implementation here
```


Best Practices

- Design during early requirements phase
 - Model as subjects, objects and permissions
- Use an appropriate policy model (DAC, RBAC, ABAC etc.)
- Use middleware / framework if available
 - If not, create your own! DO NOT copy-paste duplicate code!
- Implement resource limits / quotas
- Sanitize/normalize user input !!!

```
requested_file.startswith("/home/user/share/")
```

```
requested_file = "/home/user/share/../../../../etc/shadow"
```

The Human Factor

Security and humans

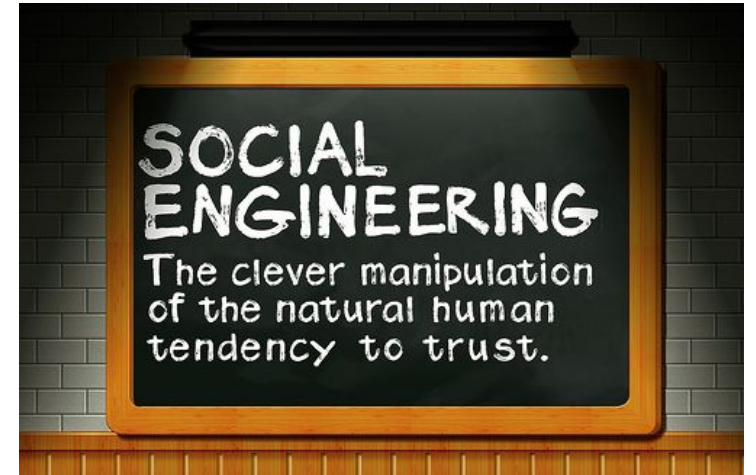
- Security policies must be in place
...and must be followed.
- Regardless of how strong (and expensive) your secure deployment is:
 - Humans can still write their passwords on post-it notes
 - Humans can still give their passwords to anyone they trust
 - Humans can still open tempting attachments...

Social engineering

- Non-technical intrusion
- Involves tricking people to break security policies
 - Manipulation
- Relies on false confidence
 - Everyone trusts someone
 - Authority is usually trusted by default
 - Non-technical people don't want to admit their lack of expertise
 - They ask fewer questions.
 - Most people are eager to help.
 - When the attacker poses as a fellow employee in need.

Social engineering

- People are not aware of the value of the information they possess.
- Vanity, authority, eavesdropping – they all work.
- When successful, social engineering bypasses ANY kind of security.



Types of phishing

- By used technology
 - Smishing (SMS)
 - Vishing (Voice)
 - Email phishing
 - Angler phishing (via social networks)
- By target
 - Watering Hole Phishing (people visiting a certain website)
 - Spear phishing (a specific organization)
 - Whaling (C-level from a specific organization)

Resources

- [1] <http://www.profsandhu.com/confrnc/asiaccs/asiaccs06-pei.pdf>
- [2] <http://www.cs.cornell.edu/courses/cs5430/2011sp/NL.accessControl.html>
- [3] http://cnitarot.github.io/courses/cs526_Spring_2015/s2014_526_ac.pdf
- [4] <https://people.cs.rutgers.edu/~pxk/419/notes/access.html>