# [Th 4] Access Control (Authorization)

Access control is the application of constraints on who or what is authorized to perform actions or access resources.

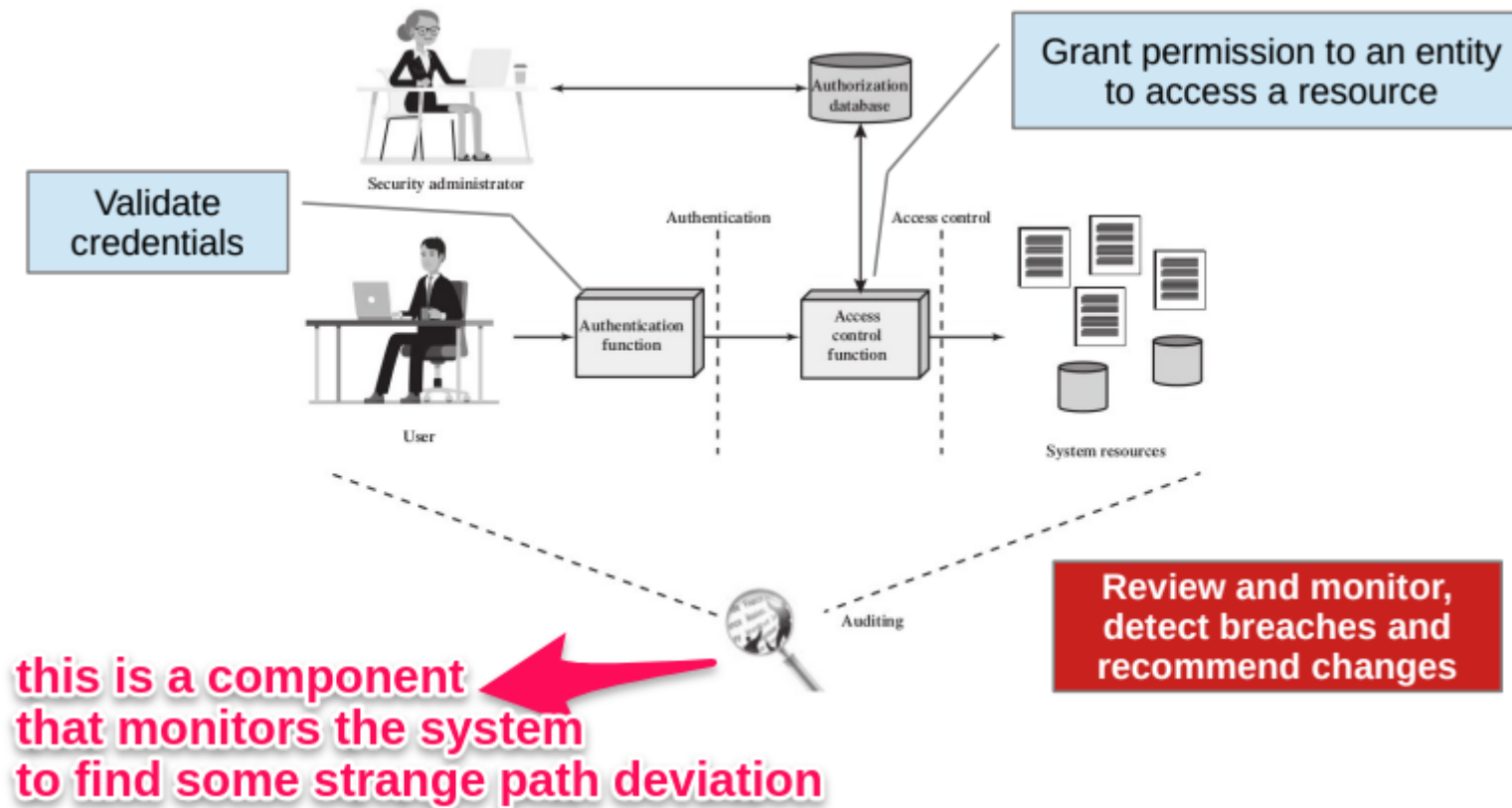In the context of web applications, access control is dependent on authentication and session management:
- **Authentication** confirms that the user is who they say they are.
- **Session management** identifies which subsequent HTTP requests are being made by that same user.
- **Access control** determines whether the user is allowed to perform the action he wants to perform.

Broken access controls are critical vulnerabilities because when they are present, a user can have access to unauthorized resources.

Example:
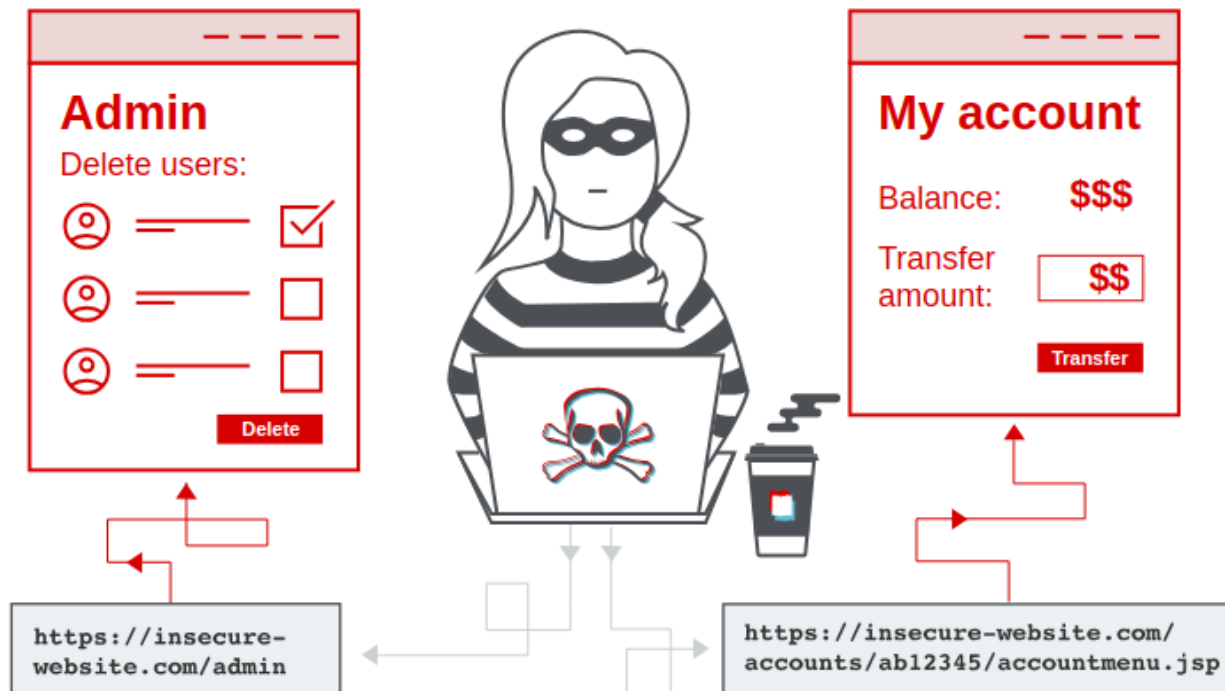
Access control implements a **security policy** that specifies who or what may have access to each specific system resource and the type of access that is permitted in each instance.

Grant permission to an entity to access a resource

Validate credentials

Security administrator

Authorization database

Authentication

Access control

User

Authentication function

Access control function

System resources

this is a component that monitors the system to find some strange path deviation

Auditing

Review and monitor, detect breaches and recommend changes

Access Control Policies

An access control policy is a set of constraints made of triples: (**subjects, objects, access rights on the object**)

A **subject** can be:
- a user, a group, or a role
- the owner of an object
- a user with some attributes

An **object** can be:
- a resource of the system
- another user
- the policy constraints themselves

An **access** right can be:
- read
- write
- execute
- delete
- create
- Search

Different policies are not mutually exclusive, in general, they are often combined (but this is not always a good idea)

Discretionary access control (DAC policy)

In this policy the owner of each resource states who can access that resource and what can be done.

It is really classic and for example, it is used in the UNIX filesystem.

the owner of an object can give the permissions to other users or groups

DAC via Access Matrices

In this policy representation we have matrices where we have

- **1 row for each subject**
- **1 column for each system object**

Each cell contains the access rights (example user1 on object2 has read, write and execute)

One column for each file

OBJECTS

	File 1	File 2	File 3	File 4
User A	Own Read Write		Own Read Write	
User B	Read	Own Read Write	Write	Read
User C	Read Write	Read		Own Read Write

SUBJECTS

One row for each user

Access rights in each cell

DAC via Access Control Lists (ACL)

In this representation, we focus on objects.

In this representation, each **object** has assigned a list that contains:

- **the subjects and the permission they have on the object**

So it is good to determine which one subject has which right given a specific resource.

It is not good to understand all access rights of a specific subject.

File 1 is owned by A, who can read and write. B can read it. C can read and write it.
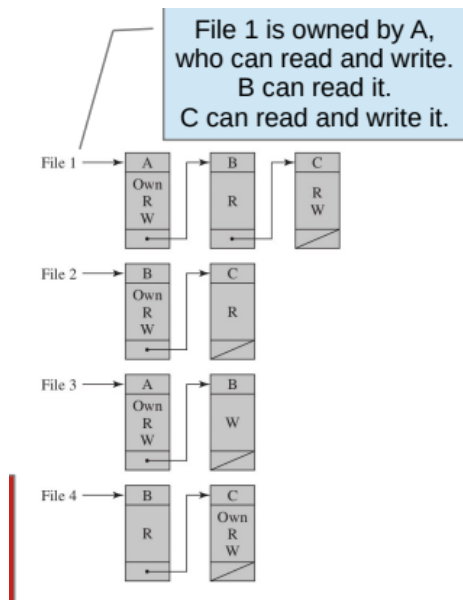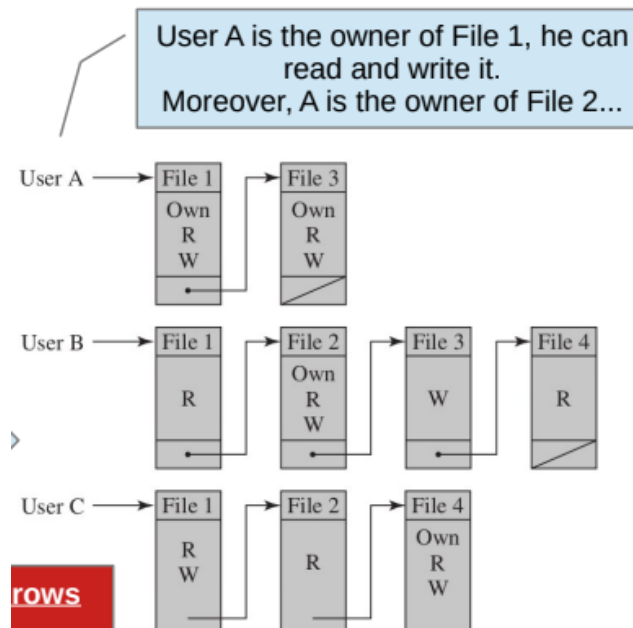
DAC via Capabilities tickets

It is quite similar to the ACL but here we focus on subjects.

In this representation, to each **subject** is assigned a list that contains:

- **the objects and the permission they have on it**

It is good to understand all access rights of a specific subject.

So it is bad to determine which one subject has which right given a specific resource.

DAC via Authorization Tables

In this representation, we just represent triples in a table.

So we have a table that contains each existing triple (subject, permission, object)

Subject	Access Mode	Object
A	Own	File 1
A	Read	File 1
A	Write	File 1
A	Own	File 3
A	Read	File 3
A	Write	File 3
B	Read	File 1
B	Own	File 2
B	Read	File 2
B	Write	File 2
B	Write	File 3
B	Read	File 4
C	Read	File 1
C	Write	File 1
C	Read	File 2
C	Own	File 4
C	Read	File 4
C	Write	File 4

Mandatory access control (MAC policy)

In this policy, each resource is assigned a security label (critical label)

Each entity (subject) is assigned to security clearances (access level)

Each subject has access to each resource of the same level and to all resources of the lower levels.
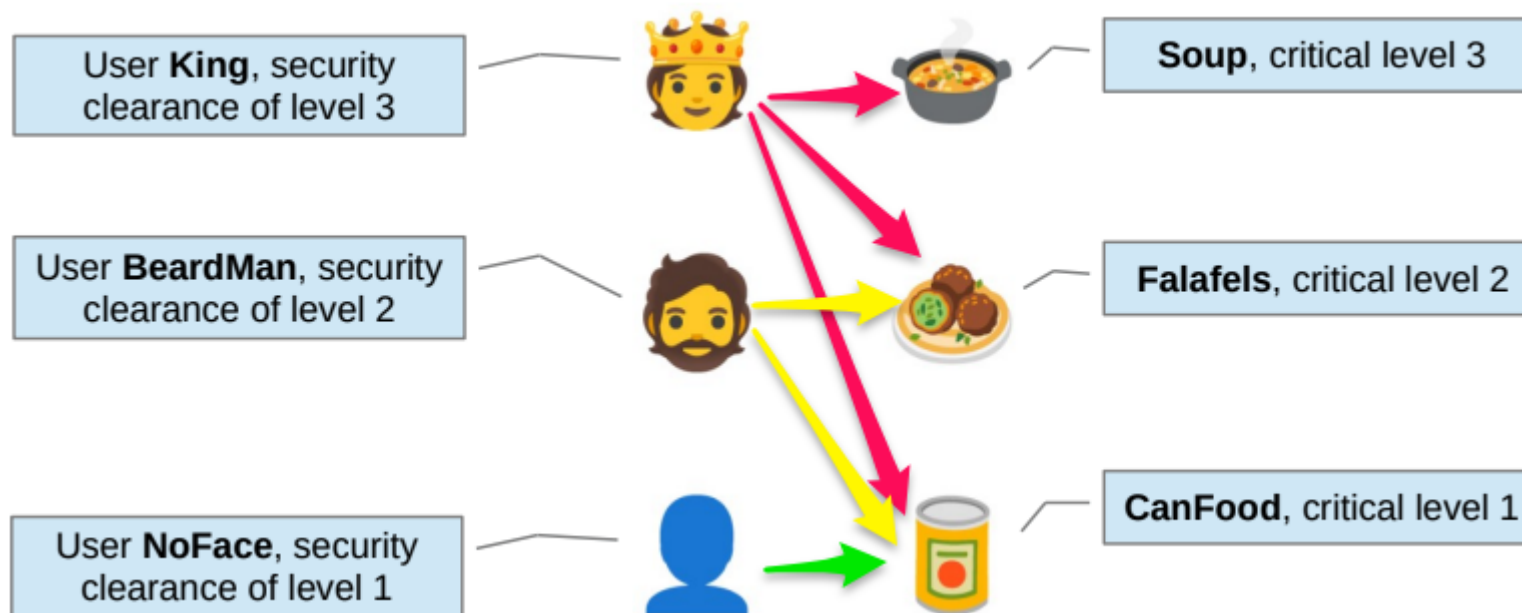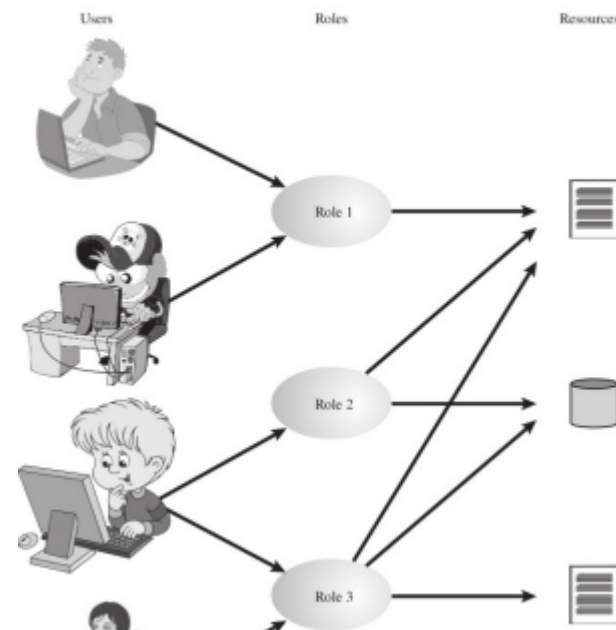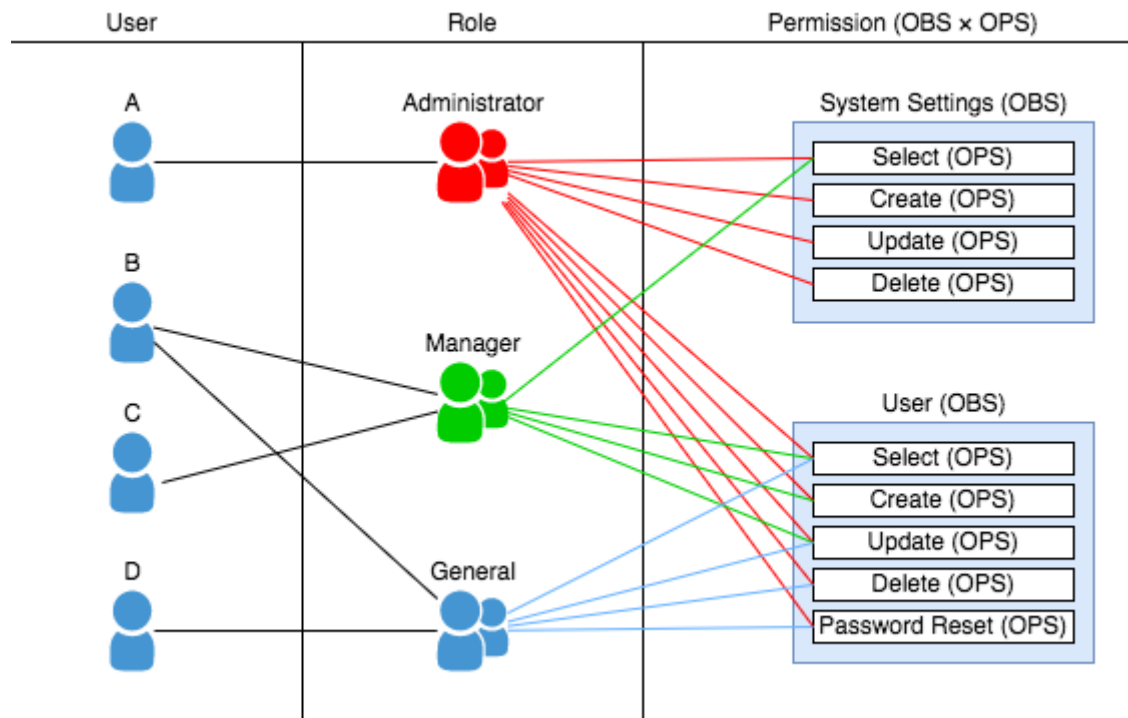
It was born for military security, but computer systems need more flexibility so it is not so used at all.

Role-based access control (RBAC policy)

In this case:
- **each entity (subject) is assigned to one or more role**
- **there are rules that state which resource a specific role can access**

In general, it is flexible because users and their association with roles may change frequently.
- for example, if I don't pay for Spotify sub then I will pass to a normal user from the premium

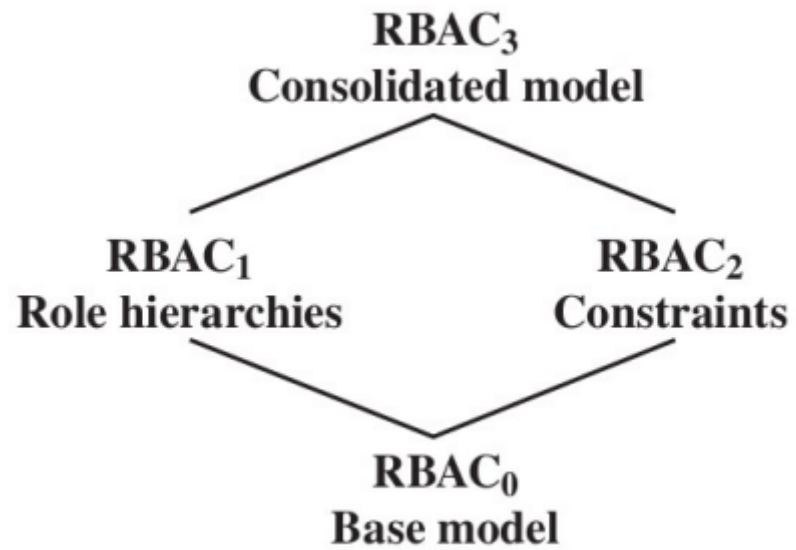**The set of roles is relatively static**
- example in Spotify if I have an account I can listen to parts of songs, if I'm premium I can listen and download all songs etc

Assigning access rights to roles results in a more stable policy.
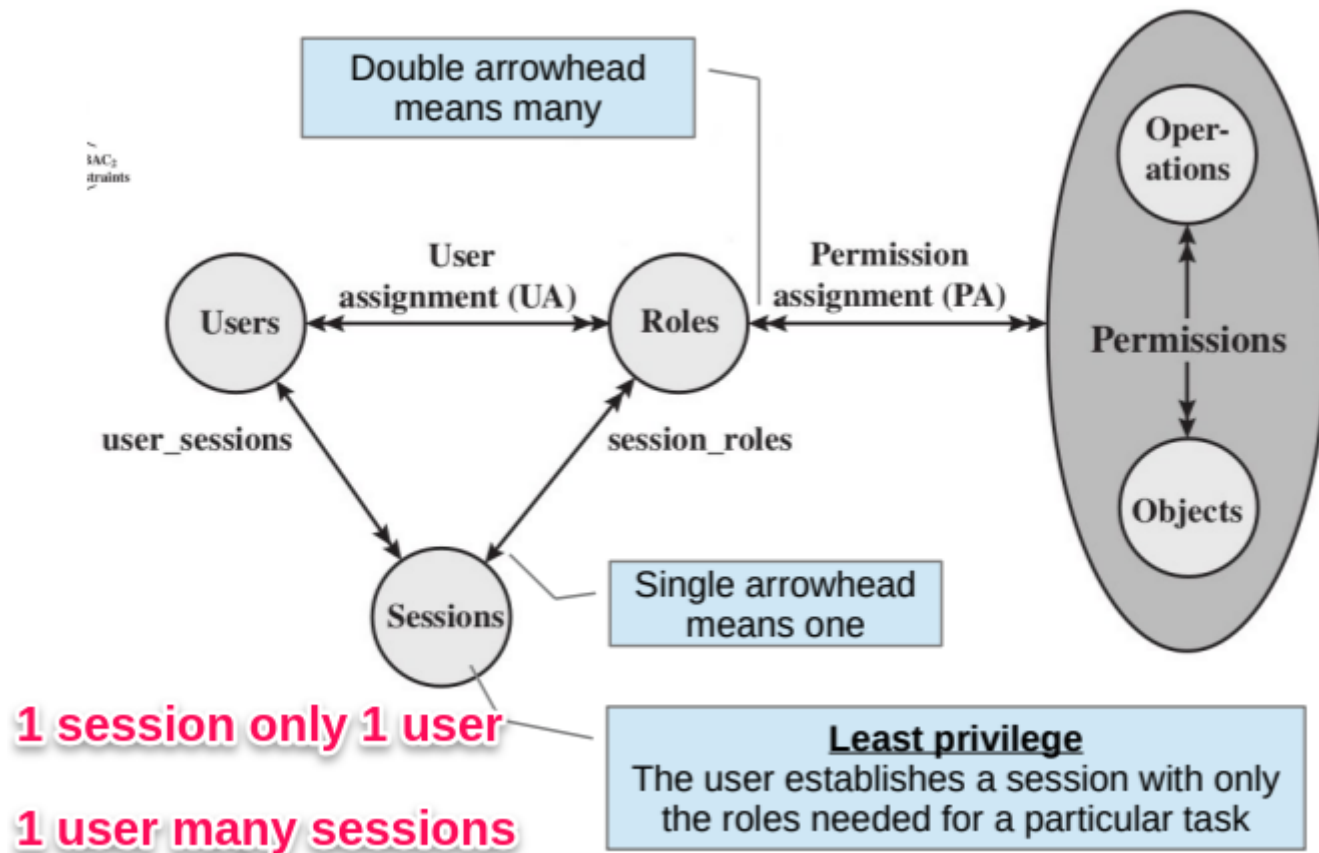
Role-based access control Reference models

A variety of functions and services can be included under the general RBAC approach.

To clarify the various aspects of RBAC, it is useful to define a set of abstract models of RBAC functionality.

$$RBAC_3$$
Consolidated model
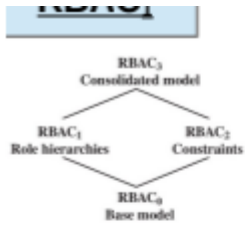
$$RBAC_1$$
Role hierarchies

$$RBAC_2$$
Constraints

$$RBAC_0$$
Base model

RBAC0

Double arrowhead
means many

User
assignment (UA)

Permission
assignment (PA)

Users

Roles

Permissions

Oper-
ations

Objects

user_sessions

session_roles

Sessions

Single arrowhead
means one

1 session only 1 user

1 user many sessions

Least privilege
The user establishes a session with only
the roles needed for a particular task

RBAC1 (role hierarchies)
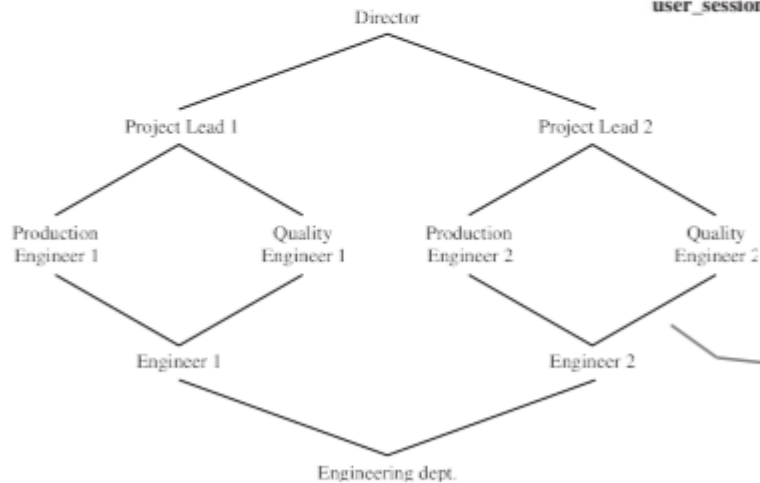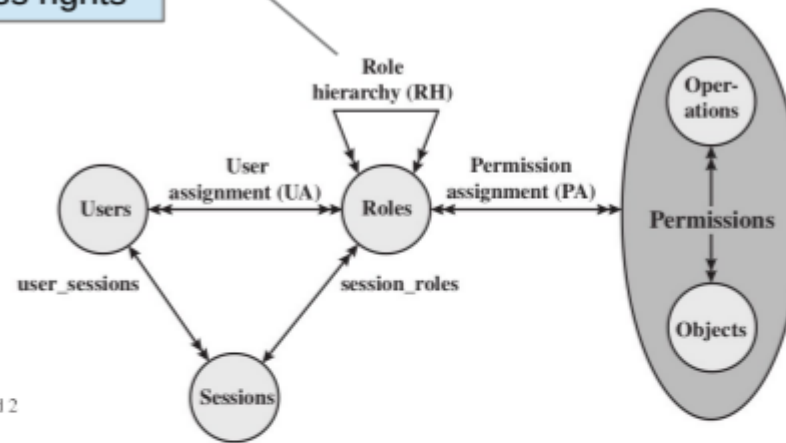
the director has all the privileges of the substant roles

RBAC2 (constraints)

Here we can add constraints on access to the right assignment

For example, we could have

- mutually exclusive roles
  - for example is not possible to lead different projects
- cardinality bounds on roles
  - we could have at most one project lead on a single project

- prerequisite roles
  - we could say that to be a director we must be a project leader and so on

In this case is not necessarily a hierarchy but I can simulate it. For example: to be a director I must be a project leader and so on.

Attribute-based access control (ABAC)

The access here is based on attributes that are assigned to resources and entities (subjects).

We can have some different kinds of attributes, boolean, strings, and so on.
For example:
- I have a premium account and a score
- if my score is >= x then I can win a prize and so on

It is really powerful but also really expensive. But it is ok for web services where the latency can help us.

ABAC scenario

An access by a subject to an object proceeds according to the following steps:

1. **A subject requests access to an object. This request is routed to an access control mechanism.**
2. **The access control mechanism is governed by a set of rules (2a) that are defined by a preconfigured access control policy. Based on these rules, the access control mechanism assesses the attributes of the subject (2b), object (2c), and current environmental conditions (2d) to determine authorization.**
3. **The access control mechanism grants the subject access to the object if access is authorized, and denies access if it is not authorized.**
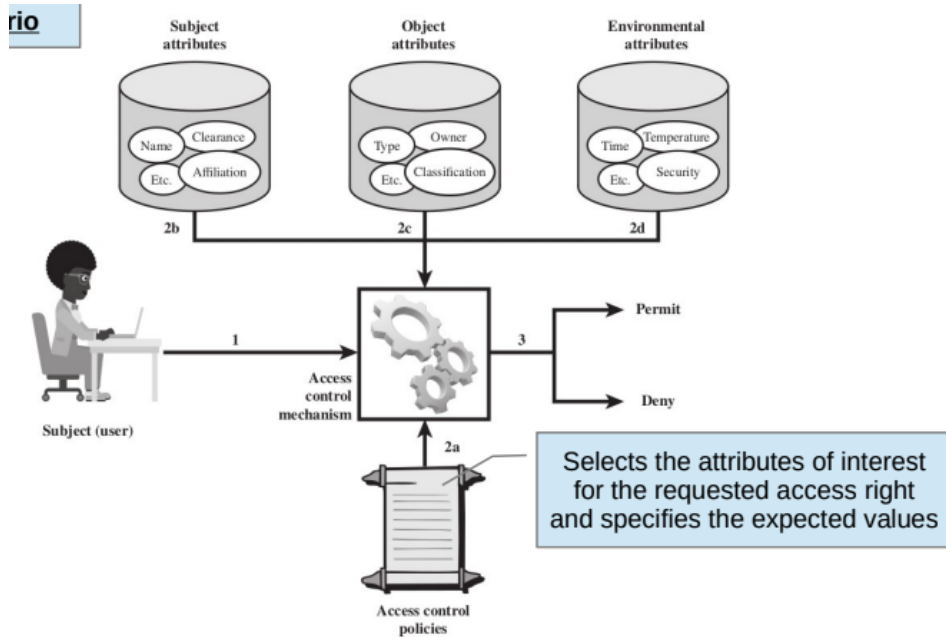
Broken access control

In Broken access control, a user can access some resources or perform some actions that they are not supposed to be able to access.

Vertical privilege escalation

In a vertical privilege escalation, a user can gain access to non-permitted functionalities.
For example, if a non-administrative user can gain access to an admin page where they can delete user accounts, then this is vertical privilege escalation.

Unprotected functionality

At its most basic, vertical privilege escalation arises when an application does not enforce any protection for sensitive functionality.
For example, if the website does not protect the admin section a user could have access to admin functionalities only by visiting:

```
https://insecure-website.com/admin
```

In some cases, sensitive functionality is concealed by giving it a less predictable URL.

This is an example of so-called "security by obscurity".

Example:

```
https://insecure-website.com/administrator-panel-yb556
```

- but it could be leaked in other website sections

Parameter-based access control methods

Some applications determine the user's access rights or role at login, and then store this information in a user-controllable location. This could be:
- A hidden field.
- A cookie.
- A preset query string parameter.

The application makes access control decisions based on the submitted value. For example:

```
https://insecure-website.com/login/home.jsp?admin=true
https://insecure-website.com/login/home.jsp?role=1
```

This approach is insecure because a user can modify the value and access functionality they're not authorized to, such as administrative functions.

Broken access control resulting from platform misconfiguration

<mark>Some applications restrict access to specific URLs and HTTP methods based on the user's role.</mark>
For example, an application might configure a rule as follows:

```
DENY: POST, /admin/deleteUser, managers
```

This rule denies access to the `POST` method on the URL `/admin/deleteUser`, for users in the manager's group. V
Some application frameworks support various non-standard HTTP headers that can be used to override the URL in the original request, such as `X-Original-URL` and `X-Rewrite-URL`.
For example:

```
POST / HTTP/1.1 X-Original-URL: /admin/deleteUser
```

Horizontal privilege escalation

<mark>Horizontal privilege escalation occurs if a user is able to gain access to resources belonging to another user, instead of their own resources of that type. (on the same level)</mark>

For example, if an employee can access the records of other employees as well as their own, then this is horizontal privilege escalation.

```
https://insecure-website.com/myaccount?id=123
```

- here we could change the ID and navigate to other users' account

In some applications, the exploitable parameter does not have a predictable value.

But we could in some cases predict them, for example looking at users' public info on the websites etc.

Insecure direct object references

IDORs occur if an application uses user-supplied input to access objects directly and an attacker can modify the input to obtain unauthorized access.

https://example.com/messages?user_id=1234

Try user_id=1233... if it works there is IDOR!

Your user ID

- so I can access to other user information

Sometimes there are checks on the authentication of a user but not on the authorization.

Hunting for IDORs

1. create 2 different accounts
2. discover features (check features that can be used on the website)
   1. Pay special attention to functionalities that return user information or modify user data
3. capture the requests of these feature
4. change the ID or the parameters

```
POST /change_password

(POST request body)
user_id=1234&new_password=12345
```

When you change
your password

```
POST /change_password

(POST request body)
user_id=1233&new_password=12345
```

Try to change another user password...

https://example.com/uploads?file=user1234-01.jpeg

When you upload
the first file

```
USER_ID-FILE_NUMBER.FILE_EXTENSION
```
Easy to guess pattern

```
user1233-01.jpeg
```
Try this!

Access control vulnerabilities in multi-step processes

Many websites implement important functions over a series of steps. This is common when:

- **A variety of inputs or options need to be captured.**
- **The user needs to review and confirm details before the action is performed.**

Sometimes, a website will implement rigorous access controls over some of these steps, but ignore others.

For example:

- The minecraft website had a bug in which we could pass from the order phase directly to the order completion without passing through the payment step.

Referer-based access control

Some websites based access controls on the `Referer` **header submitted in the HTTP request.**
The `Referer` **header can be added to requests by browsers to indicate which page initiated a request.**

For example, an application robustly enforces access control over the main administrative page at `/admin`, but for sub-pages such as `/admin/deleteUser` only inspects the `Referer` header.
If the `Referer` header contains the main `/admin` URL, then the request is allowed.

This means that an attacker can forge direct requests to sensitive sub-pages by supplying the required `Referer` **header, and gain unauthorized access.**

Location-based access control

Some websites enforce access controls based on the user's geographical location.
An attacker can just use a VPN or manipulate client-side geolocation.

How to prevent access control vulnerabilities

Access control vulnerabilities can be prevented by taking a defense-in-depth approach and applying the following principles:
- **Obfuscation alone will not help**
- **Authorize only administrator by default**
  - **deny any other deafult access on resources**
- **Use a single application-wide mechanism for enforcing access controls. (if possible)**
- **Write tests for access controls to ensure they work as designed.**