UNIVERSITÀ DELLA CALABRIA

il Campus per eccellenza

# Access control

Mario Alviano

**Main References**

Computer Security: Principles and Practice – Chapter 4

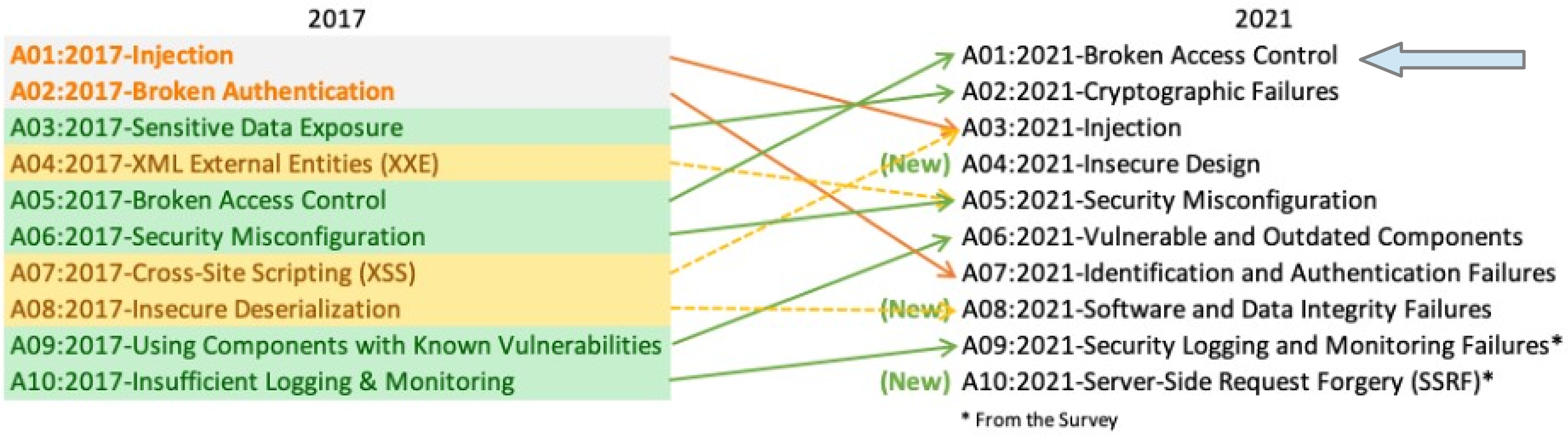
Bug Bounty Bootcamp – Chapter 10

https://portswigger.net/web-security/access-control

https://portswigger.net/web-security/access-control/idor

# OWASP Top Ten

*A broad consensus about the most critical security risks to web applications*



2017

A01:2017-Injection
A02:2017-Broken Authentication
A03:2017-Sensitive Data Exposure
A04:2017-XML External Entities (XXE)
A05:2017-Broken Access Control
A06:2017-Security Misconfiguration
A07:2017-Cross-Site Scripting (XSS)
A08:2017-Insecure Deserialization
A09:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring

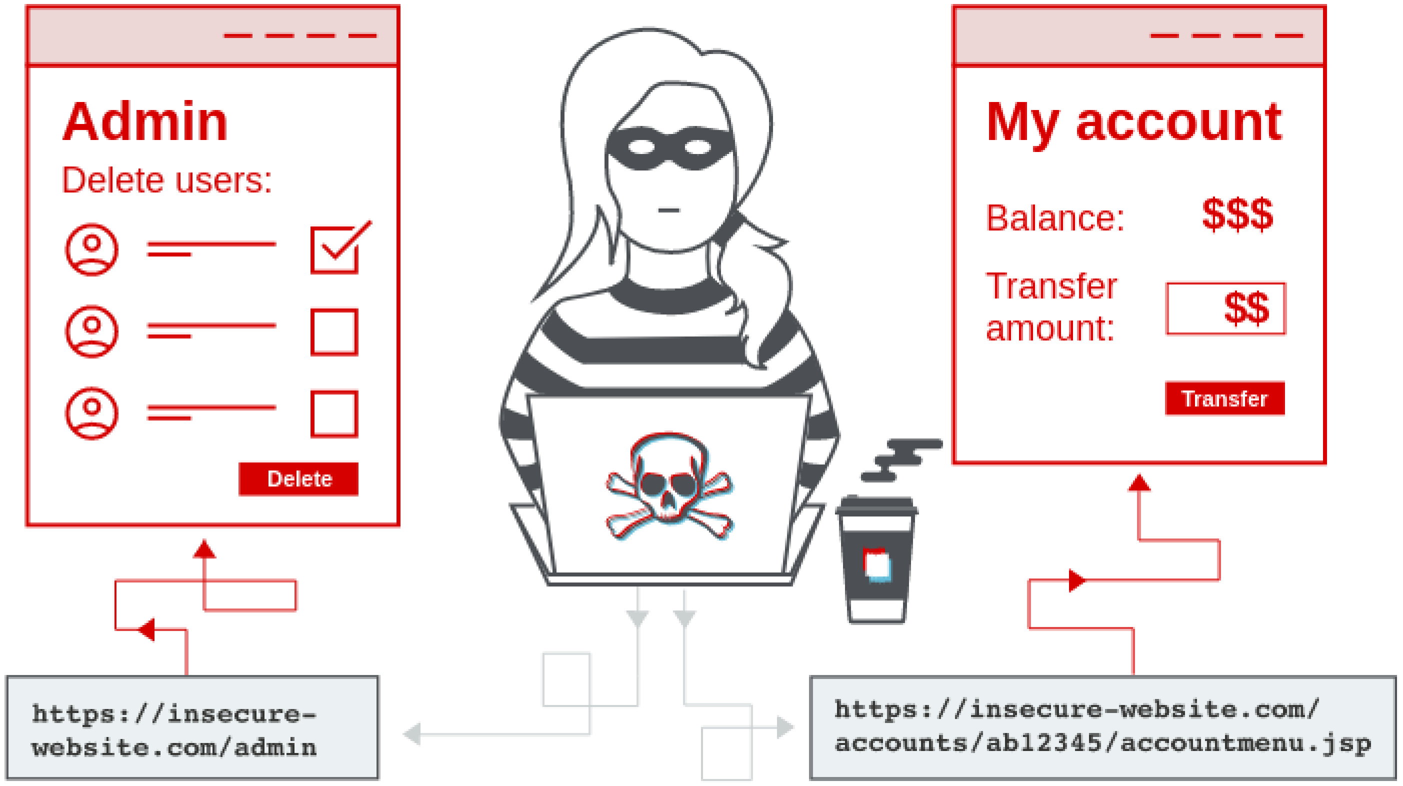
2021

A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
(New) A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
(New) A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures*
(New) A10:2021-Server-Side Request Forgery (SSRF)*

* From the Survey

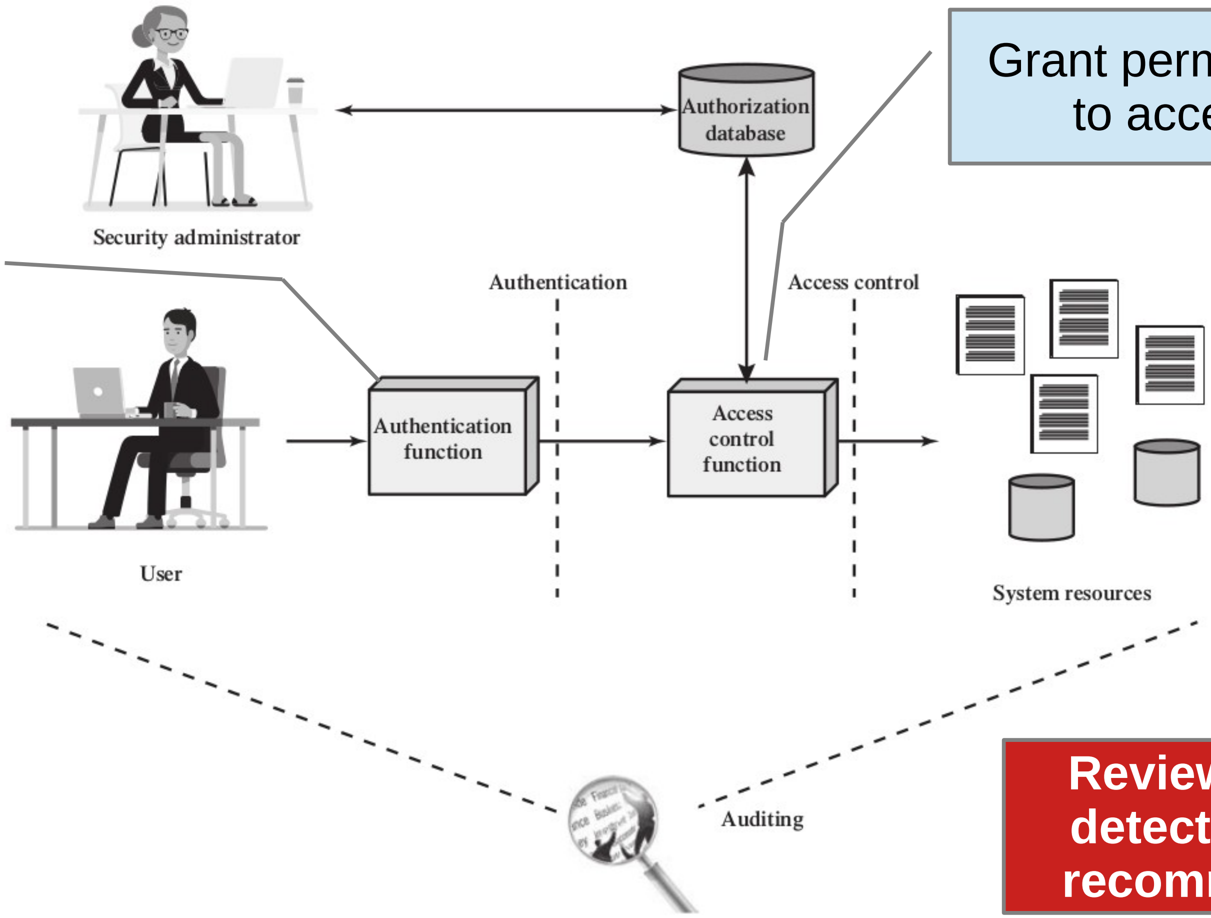
$$\begin{tikzcd}
\text{A01:2017} \arrow[r] & \text{A03:2021} \\
\text{A02:2017} \arrow[r] & \text{A07:2021} \\
\text{A03:2017} \arrow[r] & \text{A02:2021} \\
\text{A04:2017} \arrow[r, dashed] & \text{A05:2021} \\
\text{A05:2017} \arrow[r] & \text{A01:2021} \\
\text{A06:2017} \arrow[r] & \text{A05:2021} \\
\text{A07:2017} \arrow[r, dashed] & \text{A03:2021} \\
\text{A08:2017} \arrow[r, dashed] & \text{A08:2021} \\
\text{A09:2017} \arrow[r] & \text{A06:2021} \\
\text{A10:2017} \arrow[r] & \text{A09:2021}
\end{tikzcd}$$

## Access Control (or Authorization, AuthZ)


Admin
Delete users:
Delete
My account
Balance: $$$
Transfer amount: $$
Transfer
https://insecure-website.com/admin
https://insecure-website.com/accounts/ab12345/accountmenu.jsp

The application of constraints on who (or what) can perform actions or access resources

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

**Review and monitor, detect breaches and recommend changes**

## Access control policies

A set of constraints made of triples,
subjects, objects, access rights

**Subject**
- a user, a group, or a role
- the owner of the object
- a user with some attributes

**Object**
- a resource of the system
- can also be another user
- the policy constraints themselves

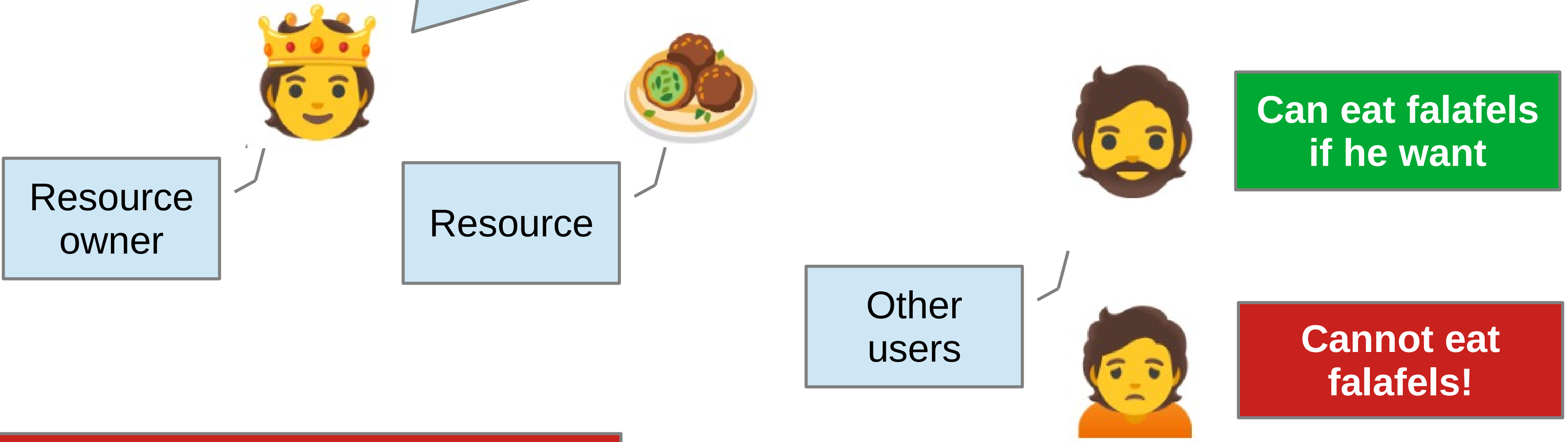
**Access right**
- read
- write
- execute
- delete
- create
- search

**Policies are not mutually exclusive… they are often combined (which is not necessarily a good idea)**

# Discretionary access control (DAC)

The owner of each resource states who can have access to that resource, and what can be done

I, the owner of these falafels, grant permission to BeardMan 🧔 to eat them!



**Can eat falafels if he want**

**Cannot eat falafels!**

**Really classic… eg. UNIX filesystem**

## DAC via Access Matrices

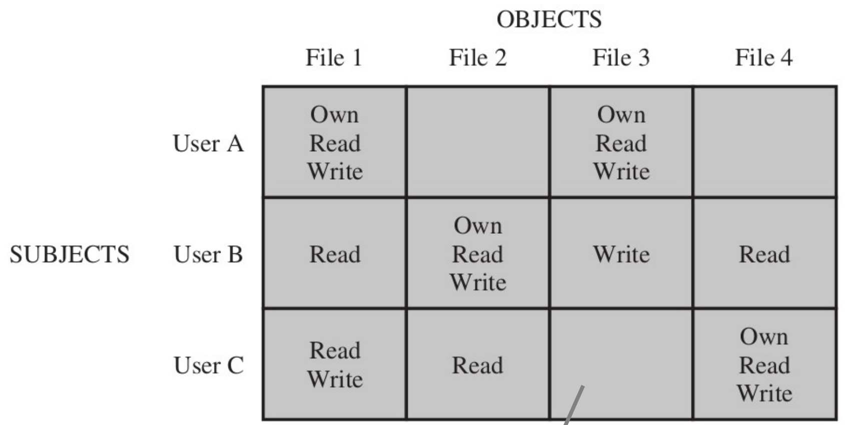
One column for each file

| | | OBJECTS | | | |
|---|---|---|---|---|---|
| | | File 1 | File 2 | File 3 | File 4 |
| SUBJECTS | User A | Own Read Write | | Own Read Write | |
| | User B | Read | Own Read Write | Write | Read |
| | User C | Read Write | Read | | Own Read Write |

One row for each user

Access rights in each cell

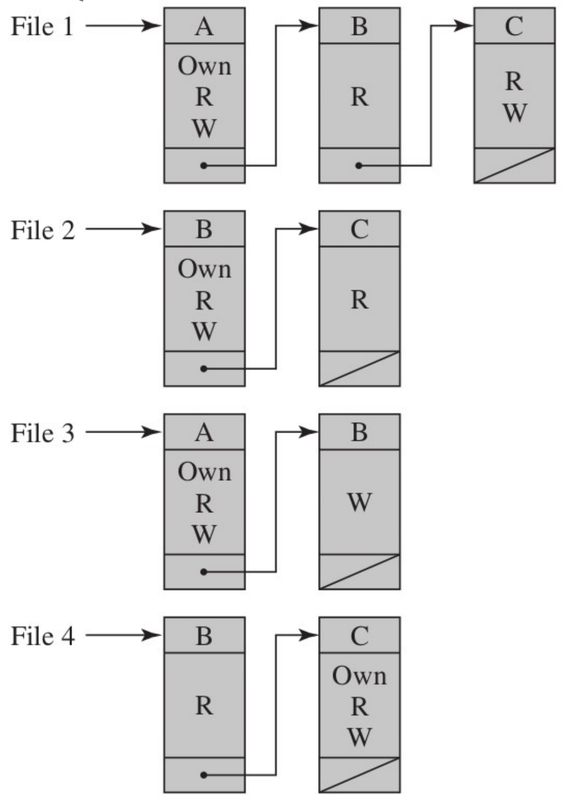
## Access Matrix via Access Control Lists (ACLs)



| | | OBJECTS | | | |
|---|---|---|---|---|---|
| | | File 1 | File 2 | File 3 | File 4 |
| | User A | Own Read Write | | Own Read Write | |
| SUBJECTS | User B | Read | Own Read Write | Write | Read |
| | User C | Read Write | Read | | Own Read Write |

It's a sparse matrix!

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



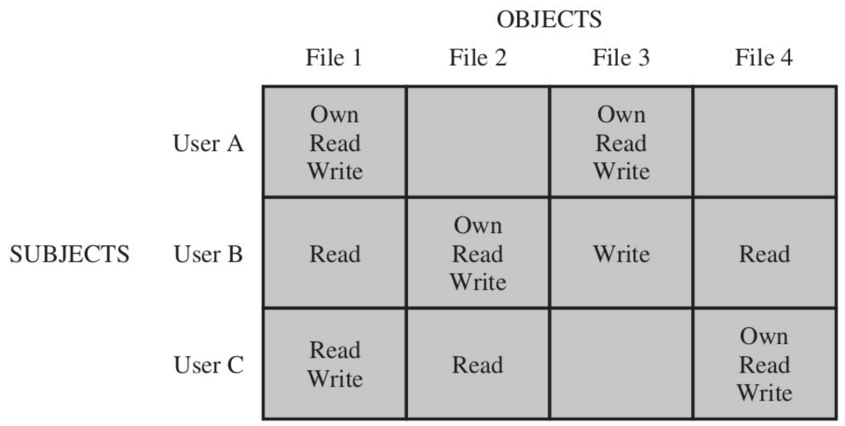
**ACLs decompose an access matrix by columns**

*Good to determine which subjects have which rights on a specific resource.*

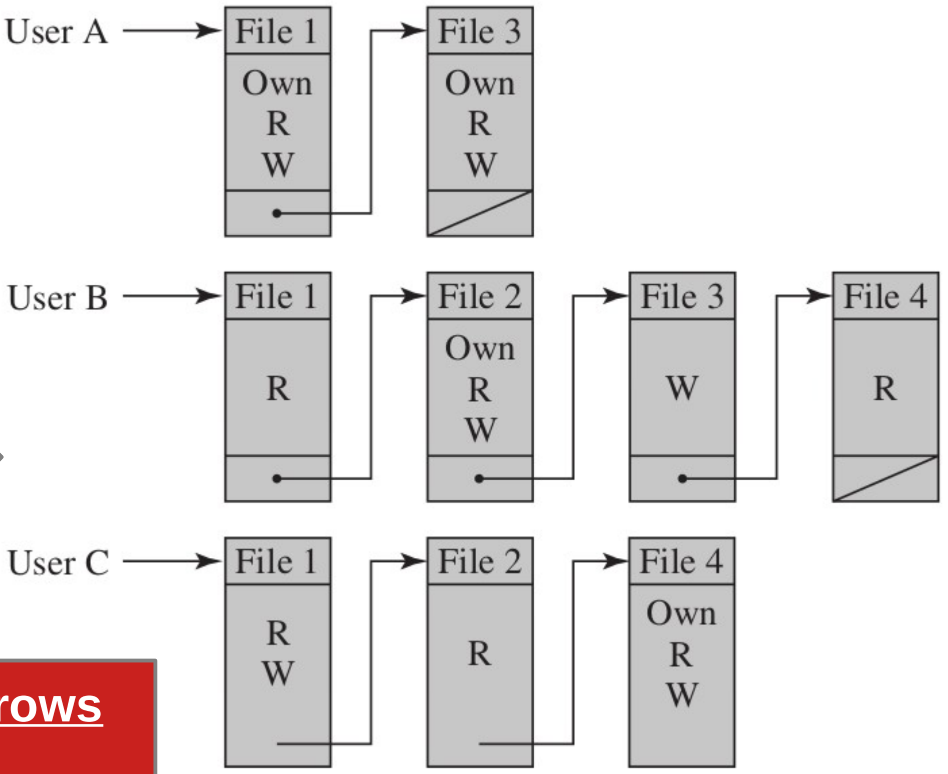
Bad to determine the access rights of a specific subject.

# Access Matrix via Capabilities Tickets

User A is the owner of File 1, he can read and write it.
Moreover, A is the owner of File 2...

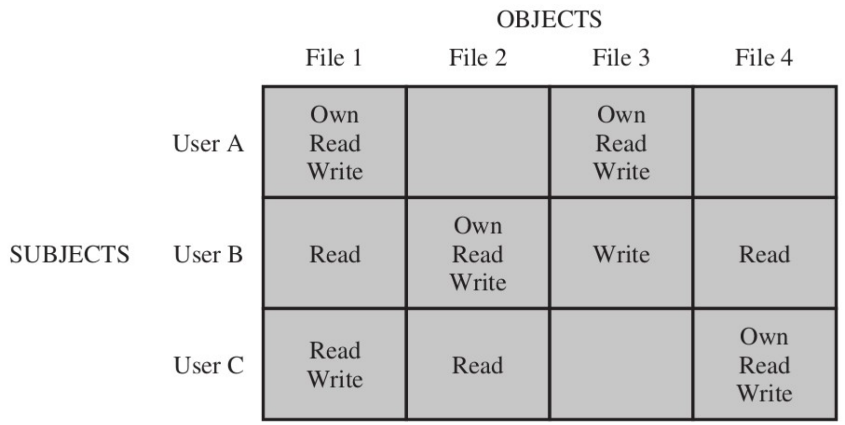


| | | OBJECTS | | | |
|---|---|---|---|---|---|
| | | File 1 | File 2 | File 3 | File 4 |
| | User A | Own Read Write | | Own Read Write | |
| SUBJECTS | User B | Read | Own Read Write | Write | Read |
| | User C | Read Write | Read | | Own Read Write |



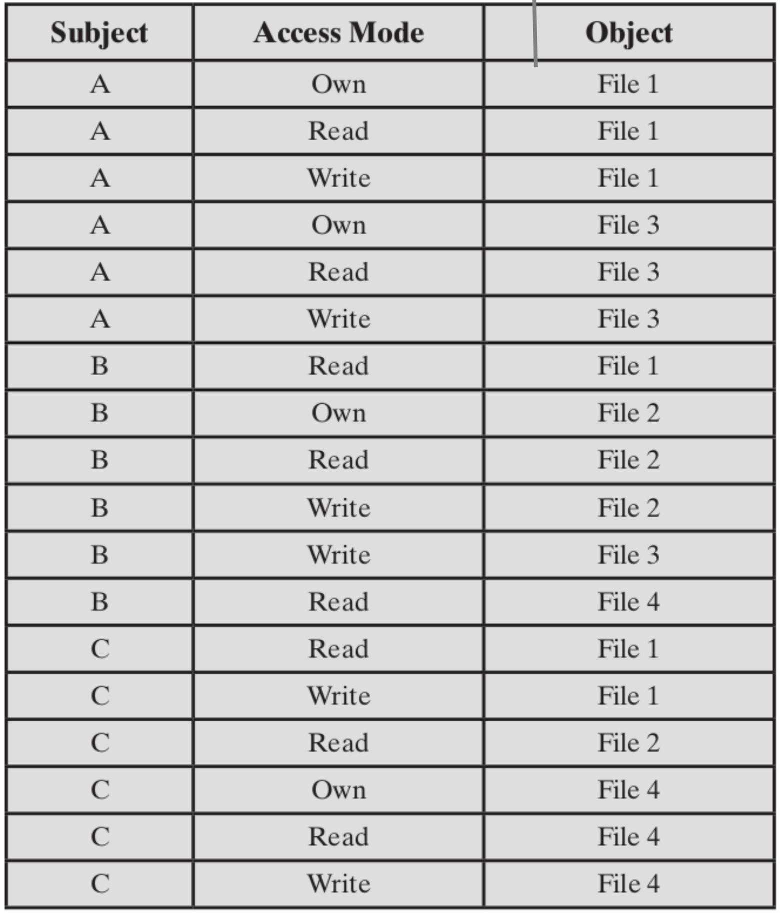
**<u>Capabilities decompose an access matrix by rows</u>**
Bad to determine which subjects have
which rights on a specific resource.
*Good to determine the access rights of a specific subject.*

# Access Matrix via Authorization Tables



| | | OBJECTS | | | |
|---|---|---|---|---|---|
| | | File 1 | File 2 | File 3 | File 4 |
| SUBJECTS | User A | Own Read Write | | Own Read Write | |
| | User B | Read | Own Read Write | Write | Read |
| | User C | Read Write | Read | | Own Read Write |

One row for each access right of each subject on each 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 |

**Just represent triples!**
Filter by subject to obtain a capability list.
Filter by object to obtain an ACL.

## Mandatory access control (MAC)

Each resource is assigned a security label (critical level), and entities are assigned security clearances (access level)

User **King**, security clearance of level 3

**Soup**, critical level 3

User **BeardMan**, security clearance of level 2

**Falafels**, critical level 2

User **NoFace**, security clearance of level 1

**CanFood**, critical level 1

**Emerged for military security. Computer systems needs more flexibility.**

## Role-based access control (RBAC)

Roles assigned to entities,
rules stating what each role can access

User King, roles **PremiumUser**, **RegisteredUser**

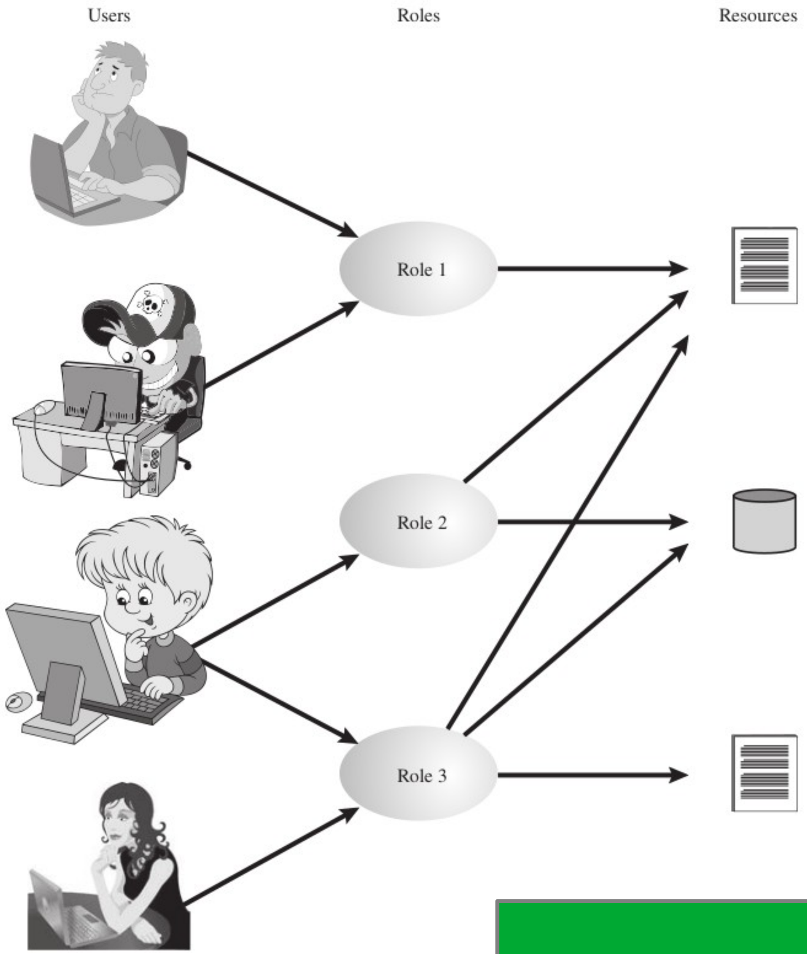
User BeardMan, role **RegisteredUser**

Unauthenticated user (can be seen as a role)

**Soup**, PremiumUser can eat it

**Falafels**, RegisteredUser can eat it

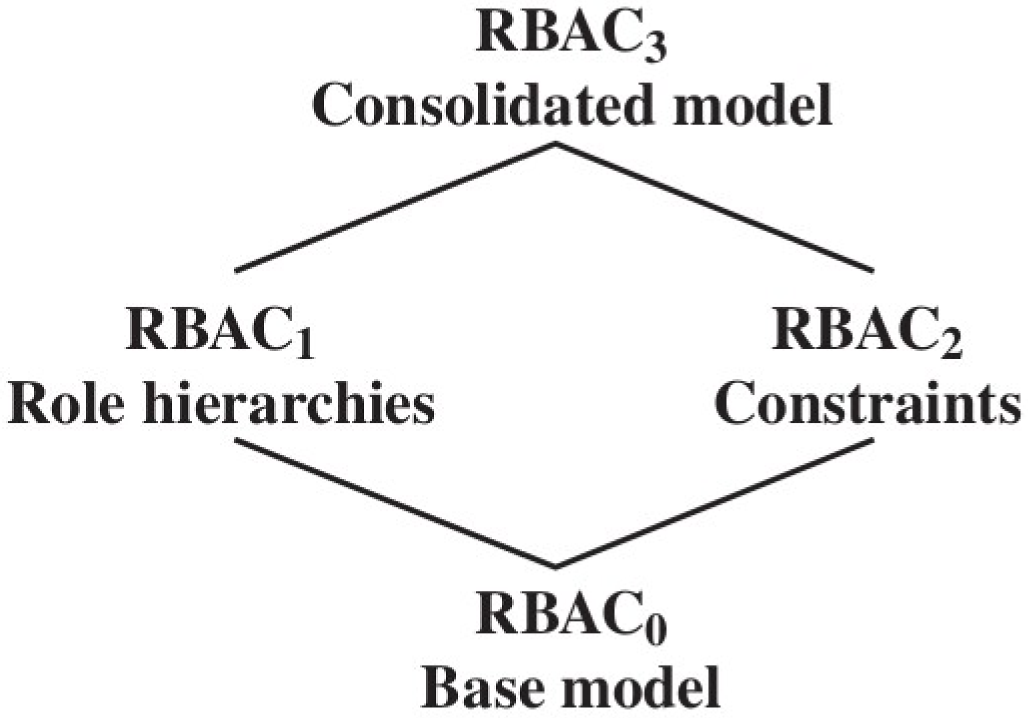
**CanFood**, anyone can eat it



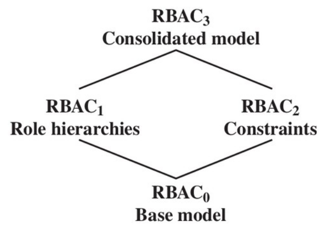
**It’s simple… yet powerful!**

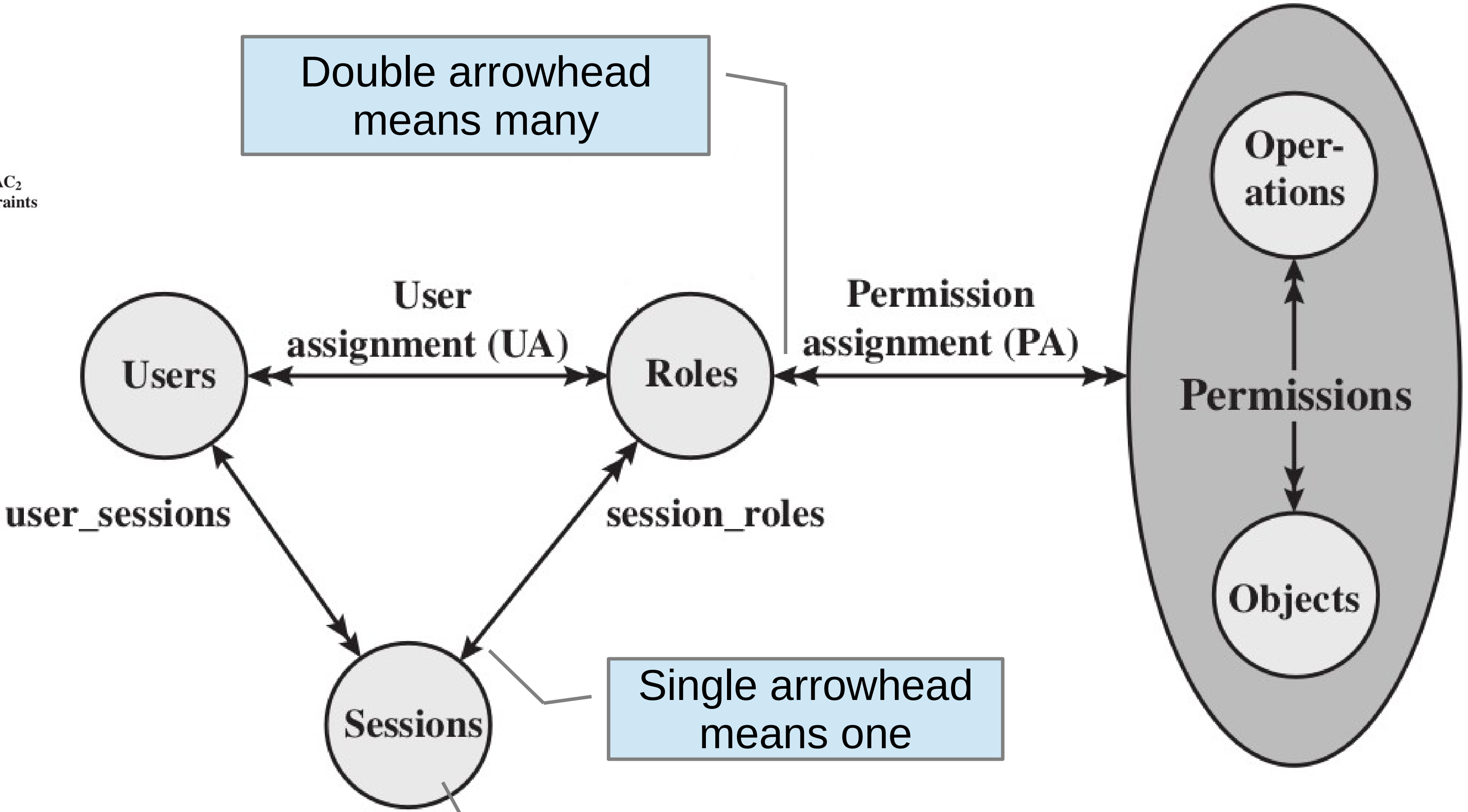
Users and their association with roles may change frequently.
*The set of roles is relatively static!*
**Assigning access rights to roles results into a more stable policy.**

## RBAC Reference Models



## RBAC$_0$



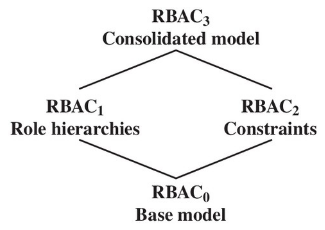


Double arrowhead means many

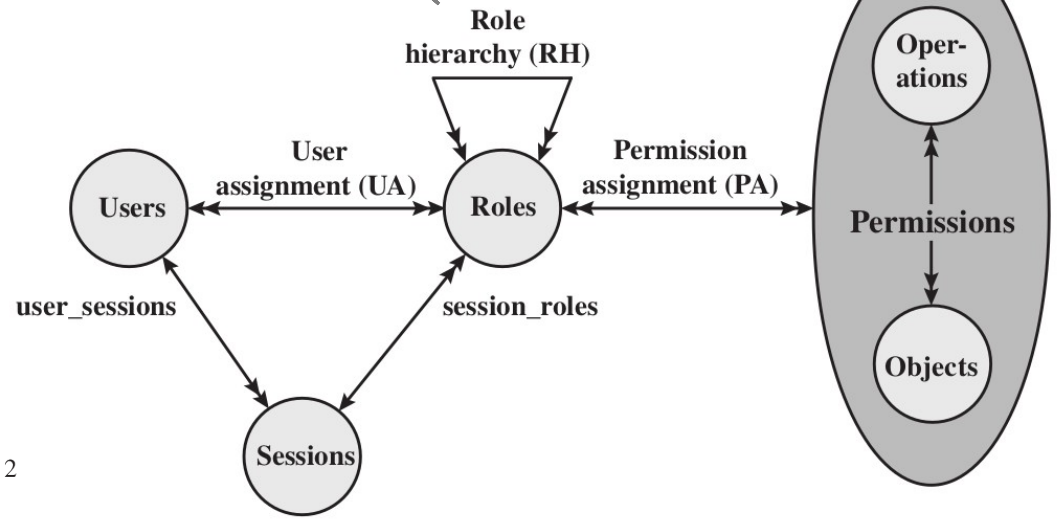
Single arrowhead means one

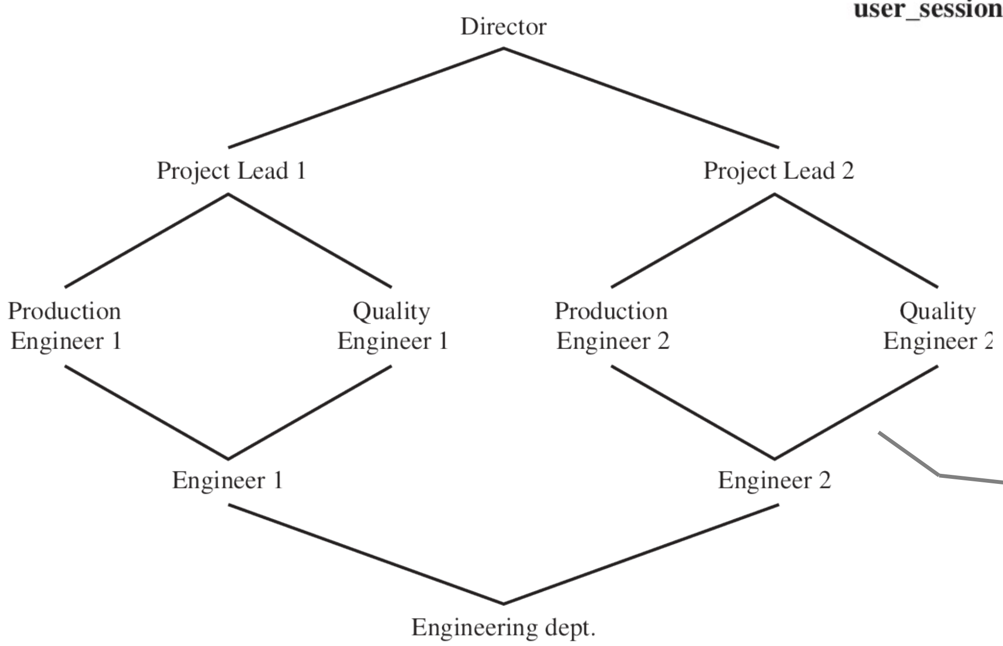
**Least privilege**
The user establishes a session with only the roles needed for a particular task

## RBAC$_1$



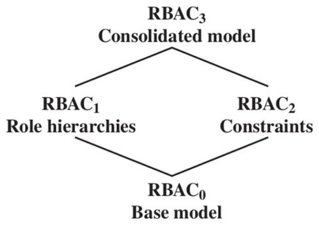
Add role hierarchy to inherit access rights





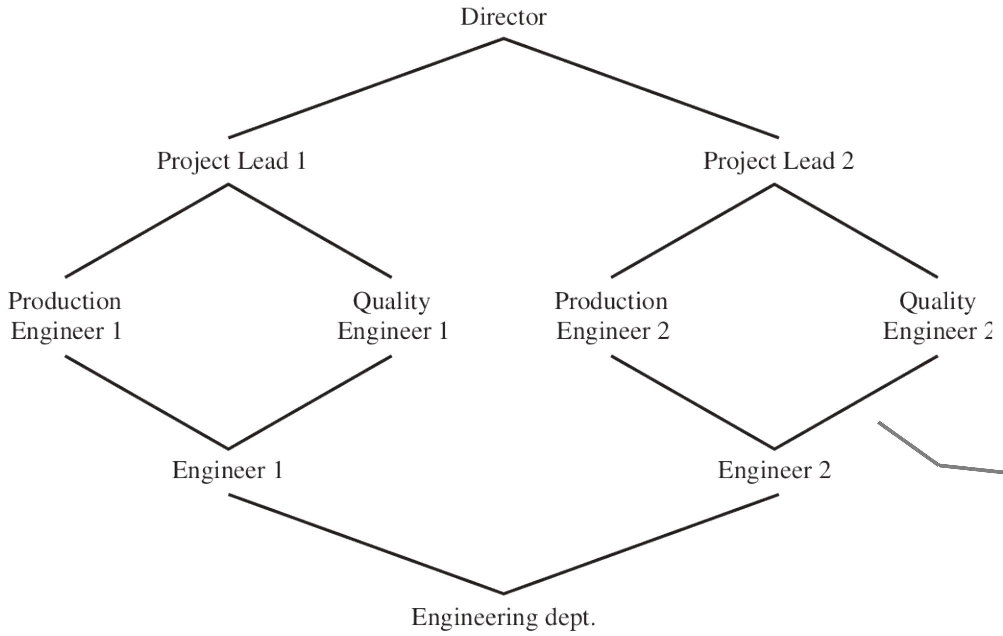
RH usually reflects hierarchical structure of roles in an organization

## RBAC$_2$



Add constraints on access rights assignment

- Mutually exclusive roles
  - cannot lead different projects
- Cardinality bounds on roles
  - at most one project leader (in each project)
- Prerequisite roles
  - to lead a project, one needs to be production and quality engineer

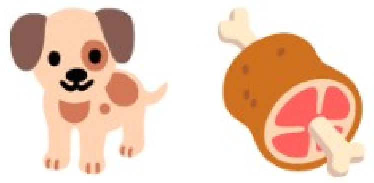


RH usually reflects hierarchical structure of roles in an organization

## Attribute-based access control (ABAC)

Access based on attributes
of entities and resources

User Dog, attribute
**vegetarian = False**



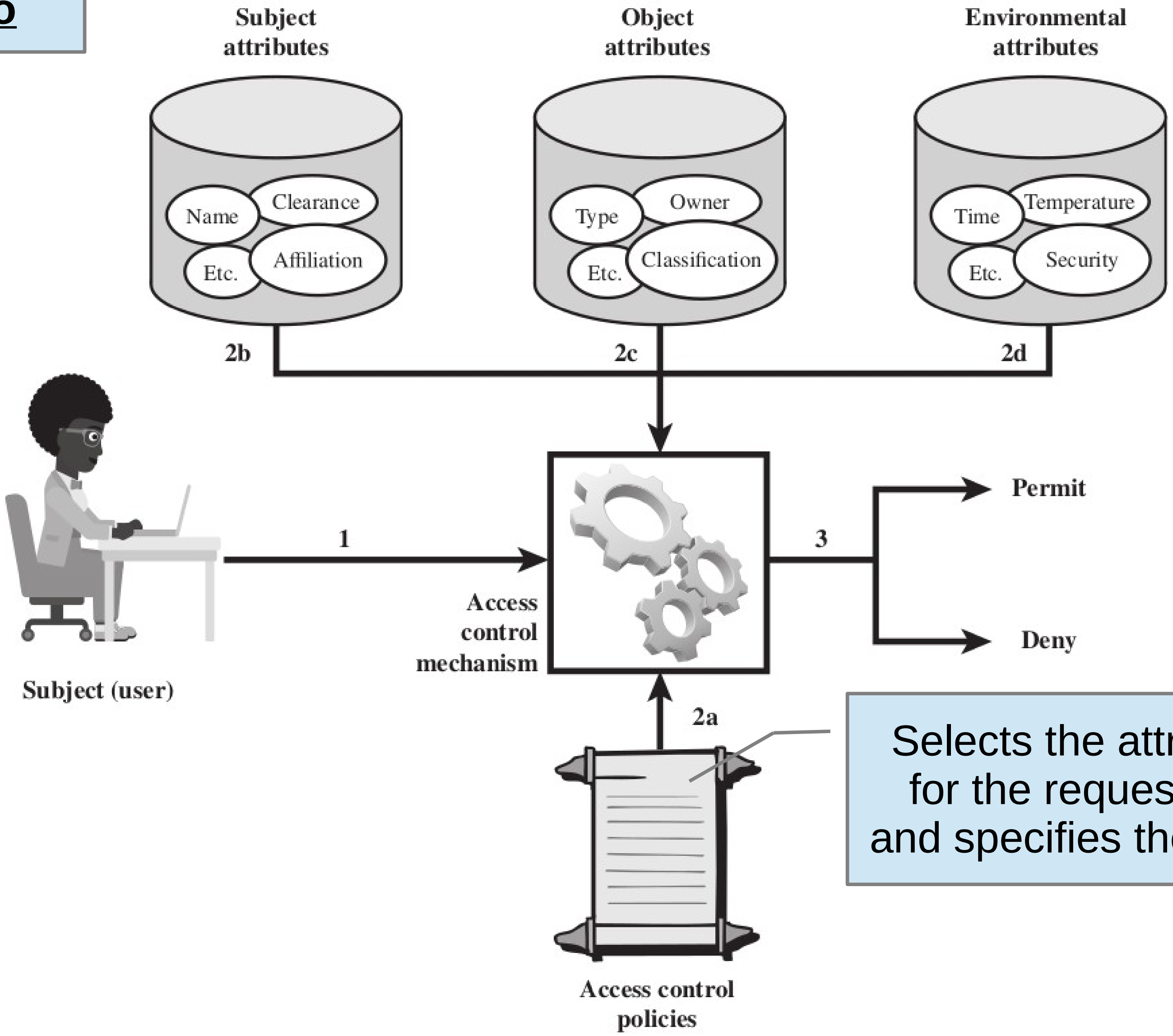
Resource Meet, attribute
**suitable_for_vegetarian = False**

User Cow, attribute
**vegetarian = True**

Resource GreenSalad, attribute
**suitable_for_vegetarian = True**

Really powerful, but expensive!
OK for web services, where the latency helps.

## ABAC scenario



Selects the attributes of interest for the requested access right and specifies the expected values

## Broken Access Control

A user can access some resource or perform some action that they are not supposed to be able to access.

**Vertical privilege escalation**
Gain access to not permitted functionalities

**Horizontal privilege escalation**
Gain access to resources of another user

## Example: Insecure Direct Object Reference (IDOR)

A missing access control on a resource that
can be accessed by directly referencing the object ID.

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

Your user ID

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

```
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!

## Prevention

- Obfuscation alone will not help
- Secure defaults
  - Deny access by default
  - Authorize only administrators by default
- Don’t mix different access control mechanisms
- Have tests for access controls

# Questions