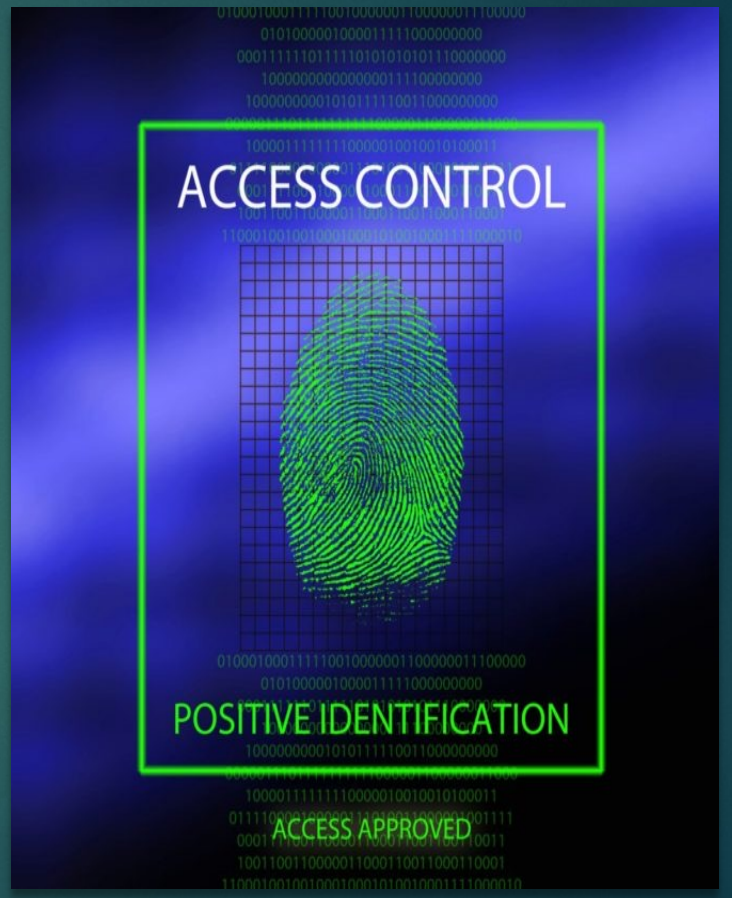
Access Control

- Access control and authorization mean the same thing
- It is about mediating access to resources based on a user specific policy
- Resources such as users / processes / files & data etc.
- Meta-operations that are overlooked – Read, Write, Execute, Create, Delete and File Ownership



What can you access

- Network access – can you connect to a system / service
- Physical Access – Laptops, Desktops, USBs, Servers
- Restricted functions – Transactions, Configuration changes

Also note: Resource access does not mean only files and database functionality – also API's, memory, storage media…anything used in processing

Example of Access Controls

- Session lock after a period of inactivity
- Account management
- Mapping of users' rights to business and process requirements
- Limits on the number of concurrent sessions
- Session termination after a period of inactivity, total time of use or time of day
- Implementation of the principle of least privilege for granting access
- Restriction of access after a certain time of day

Vulnerabilities

- Insecure IDs
- Failure to restrict URL access
- Path Traversal
- File Permission
- Client-side Caching

Insecure IDs

- Most applications have some form of identifier to reference users, roles, functions etc..
- Attacker might be able to guess or enumerate these IDs. If this is not validated….
- Attacker now has access to areas and function they would be normally prohibited from if normal checks were in place.
- Example : http://somesite.com/app?uid=1
- Attacker could alter above ID to impersonate another user and therefore access control is broken

Failure to restrict URL access

- Attacker can access resources not associated with their account
- Admin functionality can be accessed with authorization.
  - Example 1: http://somesite.com/user_panel
  - Example 2: http://somesite.com/admin_panel
- Unauthorized user should never be allowed access Example 2.
- Simple checks in code for user role prevents this

Path Traversal

- Allows an attacker to access files and directories stored outside web root folder.
- Attacker can manipulate variables that reference files
- Adding dot-dot-slash (../) sequences or absolute paths an attacker can access arbitrary files stored on file system.
- Example : http://somesite.com/getMyPic.jsp?myPic=../../../../etc/passwd
- This will return the file stored on the system

File permissions

- Attacker can access files stored locally on the Web server that should not be publicly accessible.
- Configurations files, default files, scripts, etc.
- Only files specifically intended to be presented to web users should be marked as readable by the application using the OS permissions mechanism.
- Most files directories should not be readable.
- Very few if any files should be executable

Client-side caching

- Many users access shared terminals in public access points.
- Issue arises when browsers cache web pages.
- Attackers can then gain access to inaccessible site locations and data.
- Developers should use multiple mechanisms to mitigate this:
  - HTTP Headers, meta tags
- Example: http header
  - Cache-Control: no-store, no-cache, must-revalidate
  - Cache-Control: post-check=0, pre-check=0
- Example: meta tag

```
<meta http-equiv="cache-control" content="max-age=0" />
<meta http-equiv="cache-control" content="no-cache" />
<meta http-equiv="expires" content="0" />
<meta http-equiv="expires" content="Tue, 01 Jan 1980 1:00:00 GMT" />
<meta http-equiv="pragma" content="no-cache" />
```

Access Control Models

- Discretionary access controls (DAC)

  Identify / Need to know basis
- Mandatory access controls (MAC)

  Sensitivity of the information contained in objects.
- Role-based access controls (RBAC)

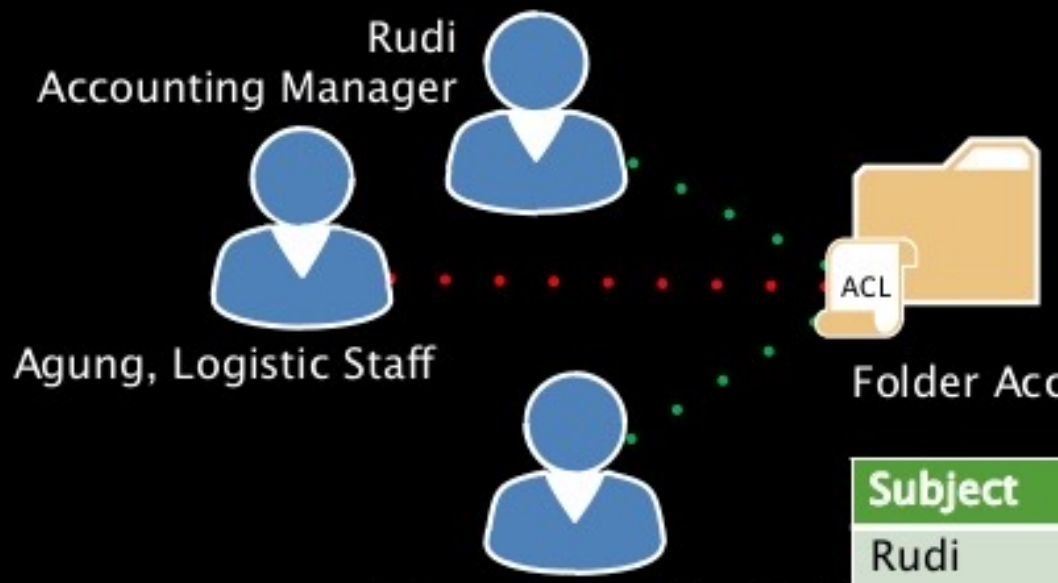
  Based on roles played by users and groups in organizational functions.
- Attribute-based access control (ABAC)

  Contextual attributes such as time of day / location

Discretionary Access Control (DAC)

- Means restricting access to information based on the identify of users and members of certain groups
- Access decisions are typically based on the authorizations granted to a user based on credentials
- Pros: Easy to use / Easy to administer / Aligns to the principle of least privileges / object owner has total control over access granted.
- Cons: Maintain documentation of roles and accesses / Tendency for scope creep to happen

DISCRETIONARY ACCESS CONTROL (DAC)



Subject	Permission
Rudi	Full Control
Johny	R W D C

Mandatory Access Control (MAC)

- Assigns sensitivity labels on information and compares this to the level sensitivity a user is operating at.
- MAC is appropriate for extremely secure systems including military applications / mission critical data applications
- Pros:

  Access to an object is based on the sensitivity of that object.

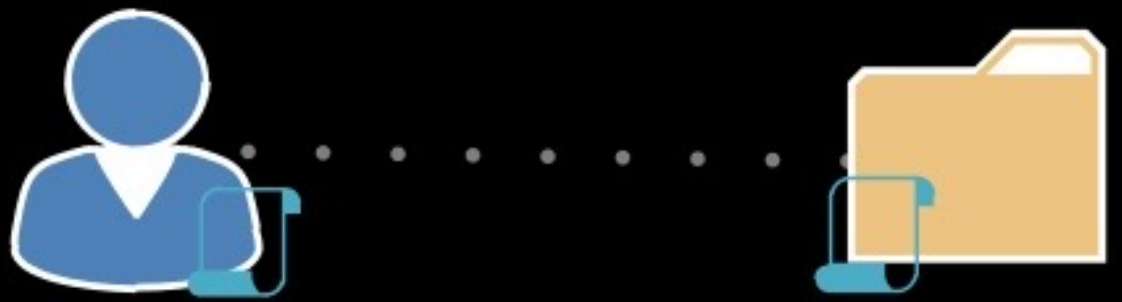
  Access is based on need to know.

  Only the Admin can grant access.
- Cons:

  Difficult and expensive to implement.

  Not Agile

MANDATORY ACCESS CONTROL (MAC)



Ken Watanabe, Intelligent Analysis
Clearance Level 2

Project Pegasus
Data Classification **Top Secret**

Clearance Level	Classification
Level 5	Top Secret, Secret, Classified, UnClassified
Level 4	Secret, Classified, UnClassified
Level 3	Classified, UnClassified
Level 2	UnClassified

Role Based Access Control (RBAC)

- Allows security administrators the ability to determine who can perform what actions where/when/from where
- Access decisions are based on the users' roles, e.g., doctor / nurse / attendant / patients all have different permissions
- Pros:

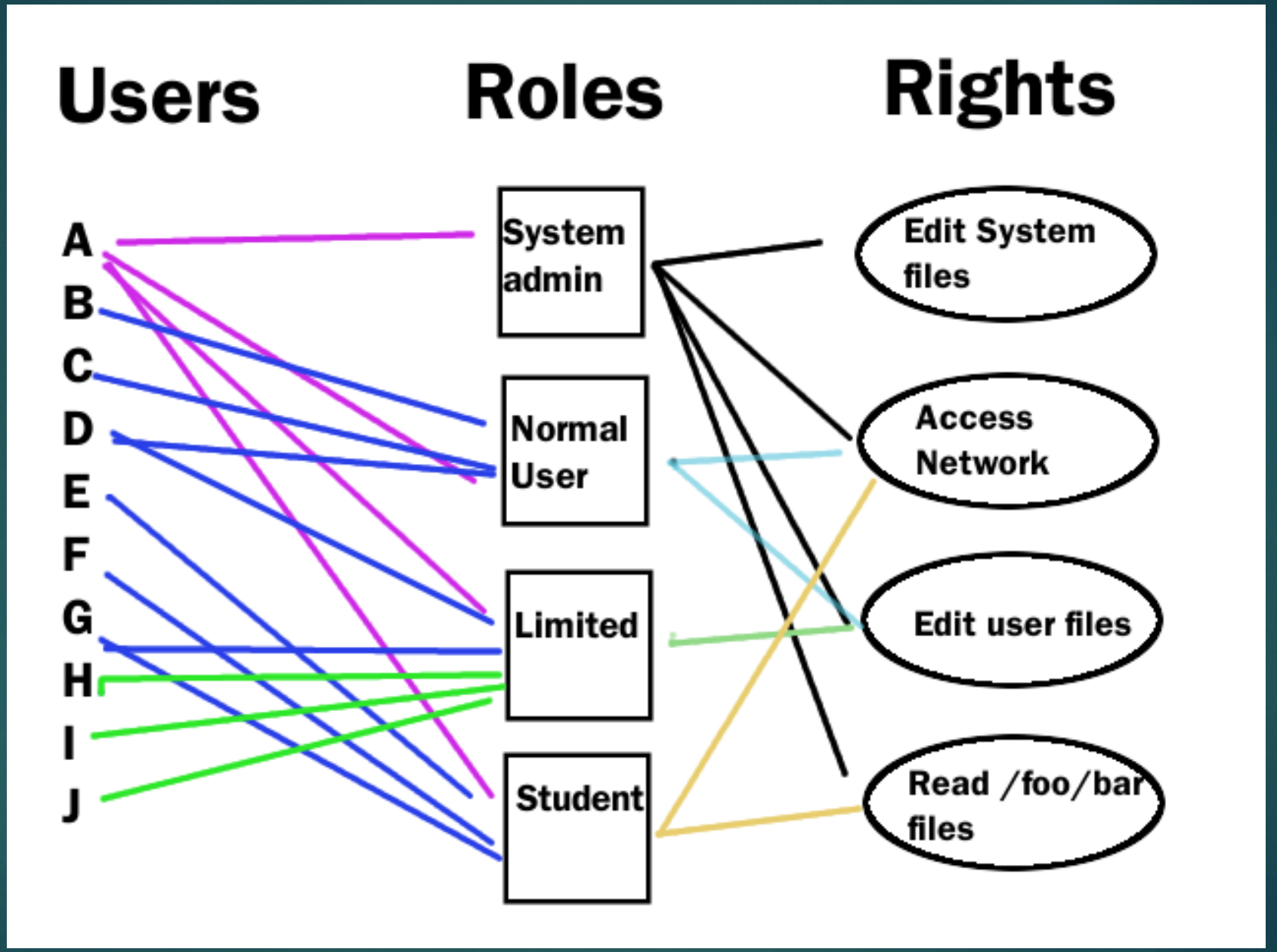
  Easy to use / Administer

  Built into most frameworks

  Aligns with security principles (segregation / duties of least privileges)
- Cons:

  Roles and access needs documentation

  Errors can occur when a user changes their role (more or less access)


Users
Roles
Rights
A
B
C
D
E
F
G
H
I
J
System admin
Normal User
Limited
Student
Edit System files
Access Network
Edit user files
Read /foo/bar files

Attribute Based Access Control (ABAC)

- Also referred to as Permission Based Access Control (PBAC)
- A permission may be represented simply as a String based name, for example "READ" / "WRITE" / "EXECUTE"
- In some (PBAC) where they provide fine-grained domain object level access, permissions are grouped into classes
- In such a system a "Document" class may be defined with with the permissions "Read", "WRITE" and "DELETE".
- A "SERVER" class may be defined with the permissions "START", "STOP" and "REBOOT"

Reminder tips

- Check authorization on every page of the application.
- Note that users can supply their own URLs, including unexpected ones. Check them.
- Do not inadvertently become a proxy for services behind a firewall.
- Only allow short amounts of time for token sessions.
- Destroy tokens on the server side on user logout.
- Remove all demo/debug code before going live with an application.
- Change/verify defaults in third party code or applications. E.g., be default, the MySQL root account does not require a password when connecting from the local host.
- Do not use higher privileges than are necessary. Web servers should run using a low-privilege account. Web applications should access database resources using an account with minimal privileges required by the application.

Reminder tips

- Limit file permissions on web files.
- Do not create session id's based solely on user input. Force a “sufficiently random” session id. Avoid any guessable id's such as a time stamp or other sequential formula.
- Do not use timestamps as a token. Timestamps can be used in generating a token, but timestamps alone are easily guessable and may not be unique. Timestamps should be used for expiring tokens.
- Verify the session. Do not trust IP addresses. DSL/Broadband users may have their IP addresses change at unpredictable times. Dial-up users are also likely to have an odd IP address (they still might exist!).
- Do not embed database or other IDs or passwords in application code.