Access Control

Lesson Introduction

- Understand the **importance of access control**
- **Explore ways** in which access control can be implemented
- Understand **how access control** is implemented

Controlling Accesses to Resources

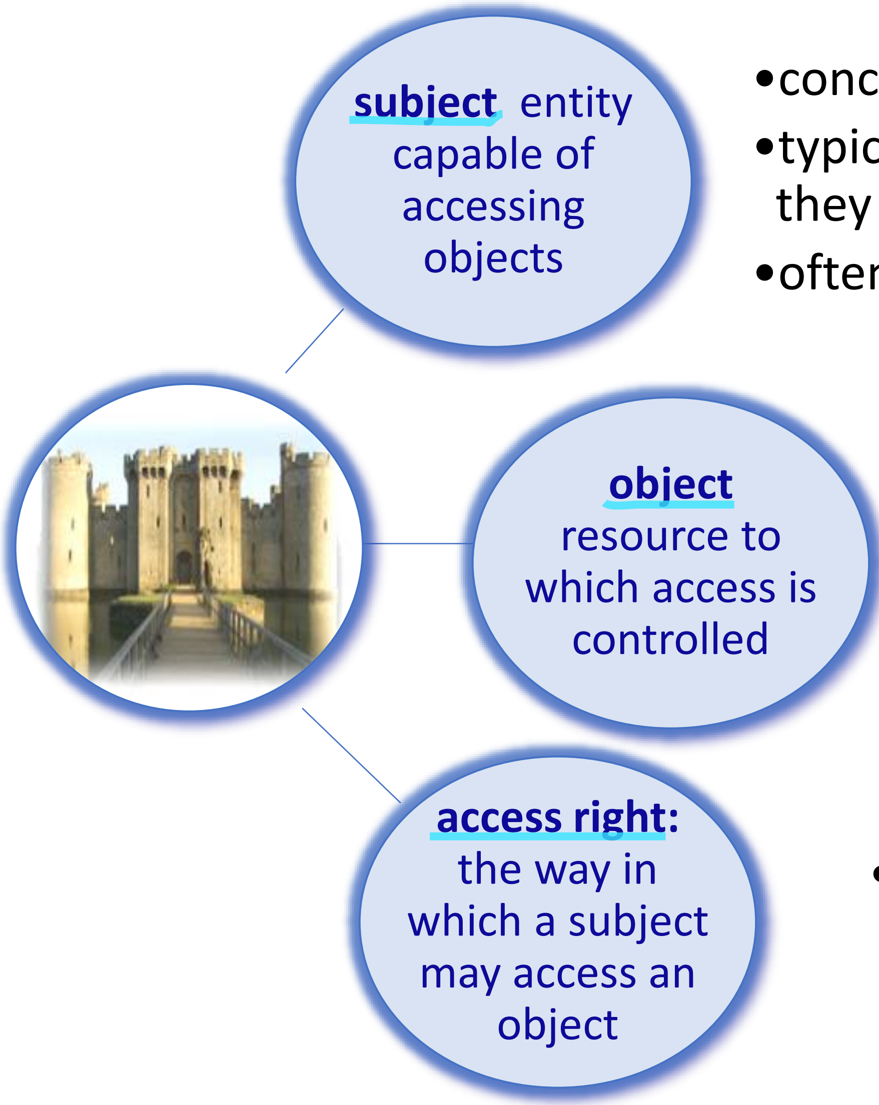
- **TCB** (reference monitor) sees a request for a resource, how does it decide whether it should be granted?

    - **Example:** Should John’s process making a request to read a certain file be allowed to do so?

Controlling Accesses to Resources

- **Authentication** establishes the source of a request (e.g., John's UID)
- **Authorization** or access control answers the question if a certain source of a request (User ID) is allowed to read the file
- **Subject who owns a resource (creates it) should be able to control access to it (sometimes this is not true)**

Access Control Basic Elements


subject entity capable of accessing objects
object resource to which access is controlled
access right: the way in which a subject may access an object

- concept equates with that of process
- typically held accountable for the actions they initiate
- often have three classes: owner, group, world

- entity used to contain and/or receive information
- protection depends on the environment in which access control operates

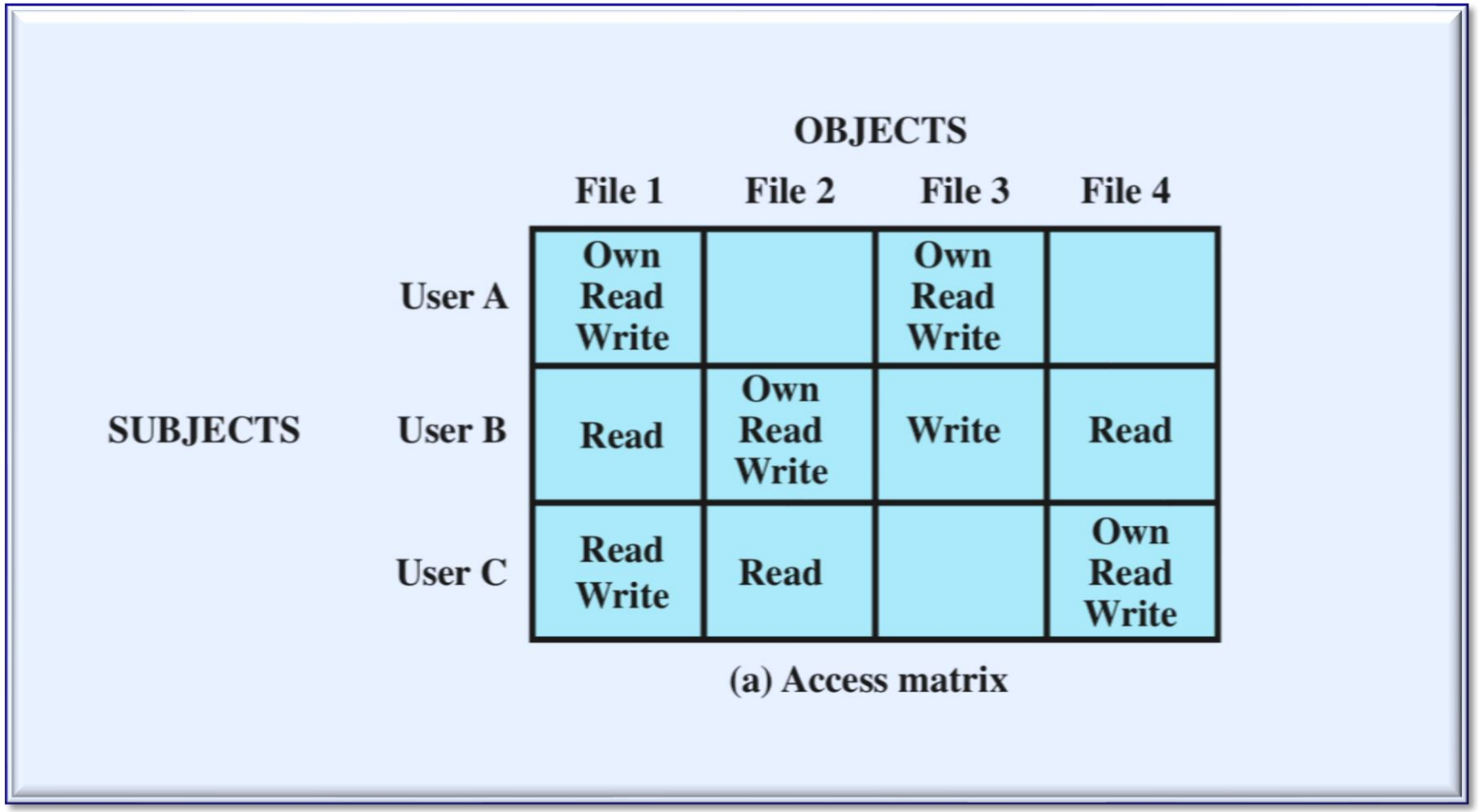
- e.g. read, write, execute, delete, create, search

Access Control Matrix (ACM)

- An access control matrix (ACM) **abstracts the state relevant to access control.**
- Rows of ACM correspond to users/subjects/groups
- Columns correspond to resources that need to be protected.
- **ACM defines who can access what**
    - ACM [U,O] define what access rights user U has for object O.

Access Matrix

Own: has maximum control over the file, you can execute, read, rewrite …
You can also give the ownership to someone else.



OBJECTS

SUBJECTS	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

(a) Access matrix

Implementing Access Control

- Access control matrix is large
- How do we represent it in the system?
  - $\boxed{\text{Column}}$ for object Oi is [(ui1, rights1), (ui2, rights2),…]
    - Called access control list or ACL
    - Associated with each resource
  - For user ui, a $\boxed{\text{row}}$ in the matrix is [(oi1, rights1), (oi2,rights2,….].
    - Called a capability-list or C-list.
    - Such a C-list stored for each user

Implementing Access Control

$$\begin{array}{c|ccc} & X & Y & Z \\ \hline A & rwx & r & \\ B & & rw & rx \\ C & & rw & rx \end{array}$$

Different ways of looking at a matrix

ACLs For each object

X → [(A,rwx)]

Y → [(A,r)(B,rw)(C,rw)]

Z → [(B,rx)(C,rx)]

C-lists For each subject (user)

A → [(X,rwx)(Y,r)]

B → [(Y,rw)(Z,rx)]

C → [(Y,rw)(Z,rx)]

ACL and C-Lists Implementation:

ACL

- **Where should an ACL be stored?**
  - In trusted part of the system
  - Consists of access control entries, or, ACEs
  - Along with other object meta-data description of data
  - For example, file meta-data has a bunch of information where this can go as well
  - Checking requires traversal of the ACL

ACL and C-Lists Implementation:

C-list

We want to be associated with that individual. C-list looks at a people, what capabilities do I have?

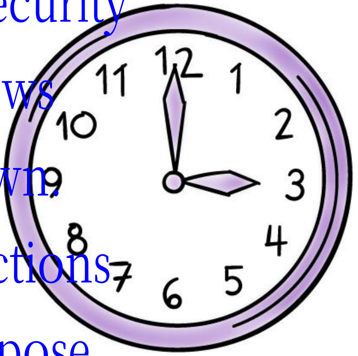
- **Where do C-lists go?**
  - A capability is a reference/handle for a resource
  - User catalogue of capabilities defines what a certain user can access
  - Can be stored in objects/resources themselves
  - Sharing requires propagation of capabilities
    - Capability comes from the resource owner

ACL and C Lists Implementation:

ACL vs. C-list

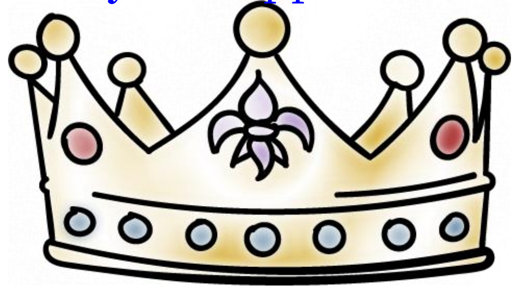
We want to make sure users only have access to what they are supposed to have access to.

It is bad that security restrictions slows the system down. Security restrictions should not impose unnecessary burden on the system.



Efficiency

C-list



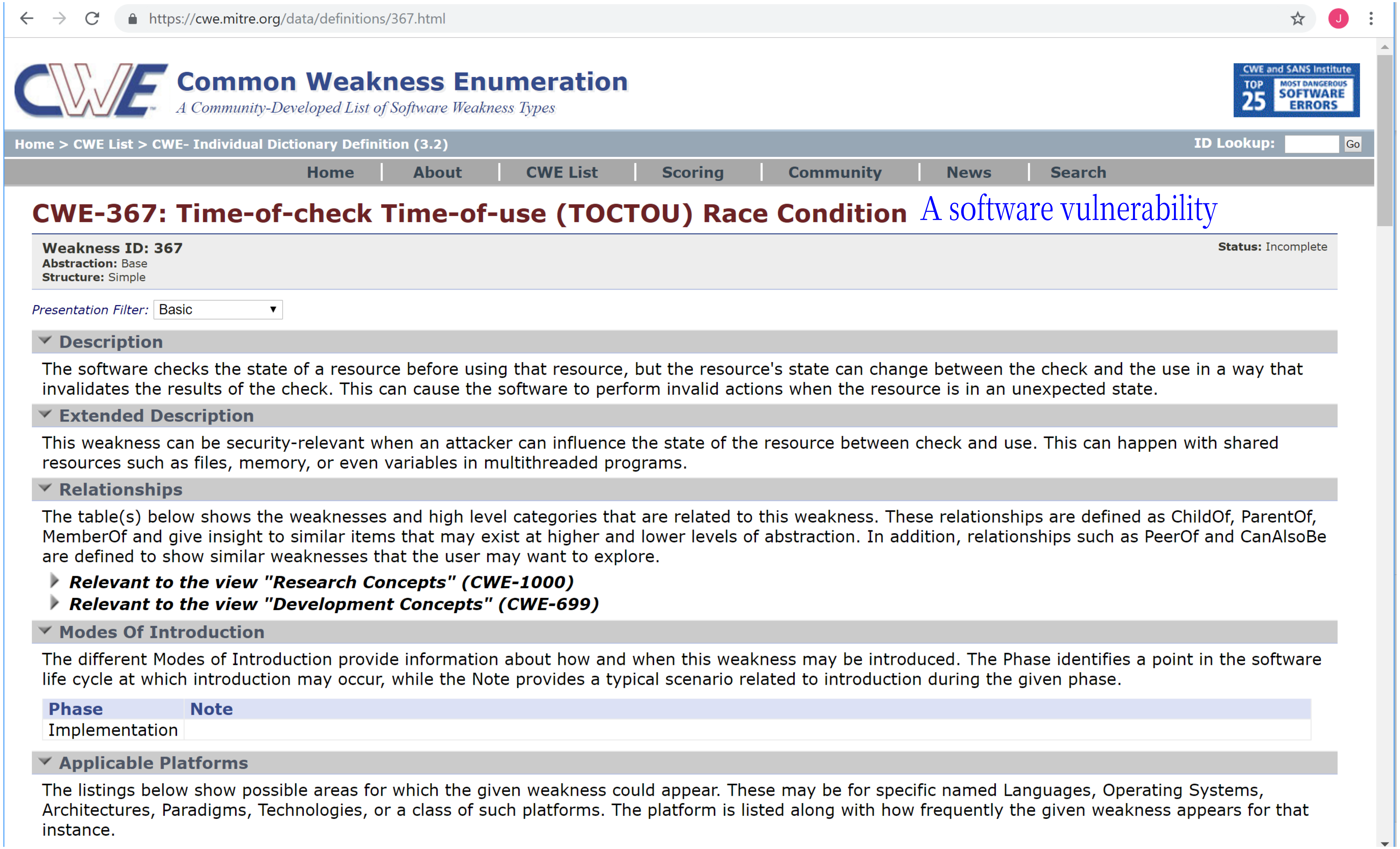
Accountability

ACL

Correctness of athorization in terms of the object.

Revocation

ACL


https://cwe.mitre.org/data/definitions/367.html
Common Weakness Enumeration
A Community-Developed List of Software Weakness Types
CWE and SANS Institute TOP 25 MOST DANGEROUS SOFTWARE ERRORS
Home > CWE List > CWE- Individual Dictionary Definition (3.2)
ID Lookup:
Go
Home
About
CWE List
Scoring
Community
News
Search

CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition

A software vulnerability

Weakness ID: 367
Abstraction: Base
Structure: Simple

Status: Incomplete

Presentation Filter: Basic

Description

The software checks the state of a resource before using that resource, but the resource's state can change between the check and the use in a way that invalidates the results of the check. This can cause the software to perform invalid actions when the resource is in an unexpected state.

Extended Description

This weakness can be security-relevant when an attacker can influence the state of the resource between check and use. This can happen with shared resources such as files, memory, or even variables in multithreaded programs.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that the user may want to explore.

Relevant to the view "Research Concepts" (CWE-1000)
Relevant to the view "Development Concepts" (CWE-699)

Modes Of Introduction

The different Modes of Introduction provide information about how and when this weakness may be introduced. The Phase identifies a point in the software life cycle at which introduction may occur, while the Note provides a typical scenario related to introduction during the given phase.

Phase	Note
Implementation	

Applicable Platforms

The listings below show possible areas for which the given weakness could appear. These may be for specific named Languages, Operating Systems, Architectures, Paradigms, Technologies, or a class of such platforms. The platform is listed along with how frequently the given weakness appears for that instance.

Time to Check vs. Time to Use (TOCTOU)

A time-to-check-time-to-use vulnerability arises when access check is performed separately from when a file is read or written. TOCTOU vulnerability arises when file permissions change after an open() call completes for the file and before it is closed.

How can this be prevented?

Access Control Policies

The original form of access control

- Discretionary access control (DAC)
  - Controls access based on the identity of the requestor and on access rules (authorizations) stating what requestors are (or are not) allowed to do
- Mandatory access control (MAC)
  - Controls access based on comparing security labels with security clearances

Students & Instructors have access to canvas of a class.

Student and Instructor are roles.

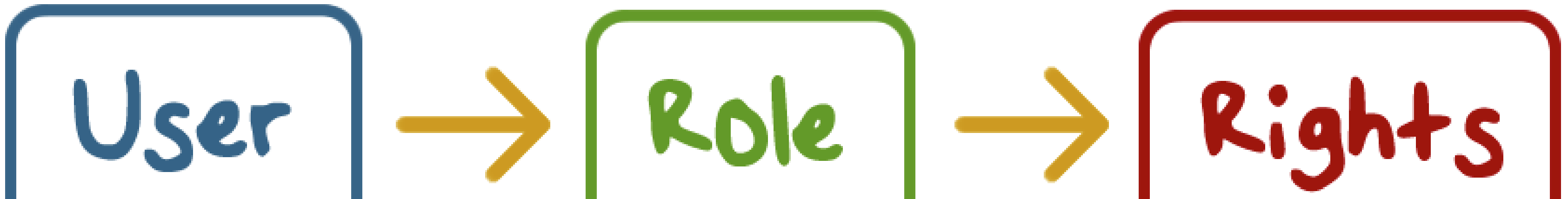
- Role-based access control (RBAC)
  - Controls access based on the roles that users have within the system and on rules stating what accesses are allowed to users in given roles
- Attribute-based access control (ABAC)
  - Controls access based on attributes of the user, the resource to be accessed, and current environmental conditions

Discretionary Access Control

- Scheme in which an entity may enable another entity to access some resource
  - often provided using an access control matrix
  - each entry in the matrix indicates the access rights of a particular subject for a particular object
  - Could we also use a C-list?

Potential Problem: you have so many individuals and you have to look at each individual and decide whether or not he/she has access control.

Role-Based Access Control (RBAC)



Role: abstraction of a group of individual

- In enterprise setting, **access may be based on job function or role of a user**
  - Payroll manager, project member etc.
  - Access rights are associated with roles
- **Users authenticate themselves** to the system
- **Users then can activate one or more roles** for themselves

RBAC Benefits

- Policy need not be updated when a certain person with a role leaves the organization
- New employee should be able to activate the desired role
- **Revisiting least privilege**
  - User in one role has access to a subset of the files
  - Switch roles to gain access to other resources

The Limitations of RBAC

- Today, defining authorization policies based solely on a user's role is not good enough.
- The context surrounding that user, their data, and the interaction between the two are also important to provide access to
  - the right user,
  - at the right time,
  - in the right location,
  - and by meeting regulatory compliance.

Drives the need for an attribute-based system.

- That means evolving an existing Role Based Access Control model to an Attribute Based Access Control (ABAC) model.

The Limitations of RBAC

- ***Limitation #1***: **Lack of Context**
  - Due to the **static** **nature** of *Role Based Access Control*, RBAC is **unable** to model **policies** that depend on **contextual** details:
    - Time-of-day, location, relationship between users, etc.
  - In other words, RBAC has no way of determining the relationships between users and using that information to make policy decisions. can be solved by splitting roles into "sub-roles", but this tend to go back to discretionary access control.
  - At its best, RBAC was originally designed to answer just one question:

    What access does a user have based on their assigned role(s)?

The Limitations of RBAC

- ***Limitation #2***: **Role Explosion**
  - RBAC is limited to defining access permissions by **role**
  - An ever-increasing *number of* **users** requires an exponentially increasing *number of* **roles** to accommodate *various permission combinations*
  - Each user often needs unique access rights
    - One user may be assigned several roles
  - A **one-size-fits-all** solution often results in too much (or too little) access

The Limitations of RBAC

- ***Limitation #3***: **Toxic Combinations**
  - Various roles assigned to a given user could contain conflicting data.
    - One user may have a role allowing him to create a purchase order, and another allowing him to approve it.
    - Such assignments posse a significant business risk if not managed properly.

The Limitations of RBAC

- ***Limitation #4***: **Management Nightmares**
  - Between growing numbers of users, and exponentially more roles, role engineering becomes an increasingly difficult task.
  - Administrators have to constantly be on top of changes to users and to roles, and ensure that role assignment combinations are current, accurate, and not conflicting with other roles a user might be assigned.
  - Any attempts to audit or certify such an environment would likewise be fraught with management nightmares.

The Limitations of RBAC

- Authorization must answers:
  - **What is a user's role(s)?**
  - **Who or what is that user related to?**
  - **What resources should he/she have access to?**
  - **What can they do with this data?**
  - **Where is data being accessed from?**
  - **At what time?**
- By defining the **context** between ***users*** and ***attributes***
  - robust policies can be defined specifying that though a user may create purchase orders, and approve them, he/she is only able to approve purchase orders to which that user is not assigned.
- Knowing a user's role is no longer enough to ensure safe and secure access control.
- The context and the relationship information between various entities involved in the system are required.

Evolving RBAC with ABAC

- Attribute Based Access Control allows an enterprise to extend existing roles using attributes and policies.
- By adding context, **authorization decisions** can be made **based** on:
  - Role of the user
  - Who or what that user is related to
  - What that user needs access to
  - Where that user needs access from
  - When that user needs access
  - How that user is accessing that information
- ***ABAC*** does this by using **policies** built upon the individual **attributes** using **natural language**.

Attribute based access control

- For example, a policy may be written as follows:
  - "Doctors can view medical records of any patient in their department and update any patient record that is directly assigned to them, during working hours and from an approved device."
- By creating a **policy** that is easy to understand, with context around a user and what he/she should have access to, ***access control*** becomes **far more robust**.
- Similar to RBAC in the sense that it also adopts a policy driven approach.
- Uses attributes of subjects, objects, and the environment (instead of roles).
  - **Attributes** are used to express rich policies

Attribute based access control

The **subject** who is demanding access to an asset.
Attributes: roles, group memberships, the user's department / company, management level,
ce

The **action** that the user wants to perform are "read" and/or "write".
In some situations, the action may be described by a combination of attributes. For example in online banking, you may have "action type= deposit", and "amount = 1000,-".

n **context**".

- A sentence like this includes four grammatical building blocks:
  - a subject
  - an action
  - a resource
  - the environment in which the request is made

smart phone, etc.).

Access Control
Lesson Summary

Discretionary focuses on individual;
Role-based assigns individual to a role;
Attribute-based specifies context.

- Fundamental requirement when **resources need to be protected**
- An **access control matrix** captures who can access what and the manner in which it can be done
- **ACLs and C-lists** are ways for implementing access control
- Getting **access control policy right is challenging**