Learning Objectives for Today (1 minute)
RBAC motivation (15 minutes)
RBAC model (15 minutes)
RBAC exercise (15 minutes)
Philosophy
→ The Gostak
Learning Objectives

→ Observe basics of RBAC model
→ Think about limitations of RBAC
The system authenticates a principal or subject via:

- something you know
- something you have
- something you are

You are then granted an identifier as your token. A “login session” is created.
User ID

[loconsto@csl ~]$ id
uid=2446(loconsto) gid=1061(profs) groups=1061(profs),1000,1001
[loconsto@csl ~]$
The access control matrix specifies which objects you can perform actions on. The reference monitor intercepts your request to perform an action on an object, checks the ACM, and either permits or denies the action.
read(2) →
  vfs_read()
  → rw_verify_area()
  → → security_file_permission()
  → → → file_permission()
  → → → → inode_permission()
  → → → → → generic_permission()
/**
 * generic_permission - check for access rights on a Posix-like filesystem
 * @inode: inode to check access rights for
 * @mask: right to check for (%MAY_READ, %MAY_WRITE, %MAY_EXEC)
 * @check_acl: optional callback to check for Posix ACLs
 *
 * Used to check for read/write/execute permissions on a file.
 * We use "fsuid" for this, letting us set arbitrary permissions
 * for filesystem access without changing the "normal" uids which
 * are used for other things..
 */

int generic_permission(struct inode *inode, int mask, 
        int (*check_acl)(struct inode *inode, int mask))
{
    int ret;

    /*
     * Do the basic POSIX ACL permission checks.
     */
    ret = acl_permission_check(inode, mask, check_acl);
    if (ret != -EACCESS)
        return ret;

    /*
     * Read/write DACs are always overridable.
     * Executable DACs are overridable if at least one exec bit is set.
     */
    if (!(mask & MAY_EXEC) || execute_ok(inode))
        if (capable(CAP_DAC_OVERRIDE))
            return 0;

    /*
     * Searching includes executable on directories, else just read.
     */
    mask &= MAY_READ | MAY_WRITE | MAY_EXEC;
    if (mask == MAY_READ || (S_ISDIR(inode) && !(mask & MAY_WRITE)))
        if (capable(CAP_DAC_READ_SEARCH))
            return 0;

    return -EACCESS;
}
Users come and go. Do we want to keep updating (and verifying) the ACM for frequent events like this?

Besides, people’s roles in an organization often speak to their different capabilities and responsibilities.

So this notion of roles becomes a more useful real-world primitive.
Role-based Access Control rewrites the subjects of the ACM so that they are *domains* or roles.

In this way, the map of users to roles can be changed and updated, but the central ACM permissions remain stable. And this more logically describes the rules within an organization.

e.g., I don’t care to create a rule saying Locasto can read this grad application; instead I rely on creating a rule that everyone with the PROF role can read any grad application.

This seems like a big conceptual and practical win, especially if we add things like role inheritance, etc.
Issues with RBAC in Practice

In large organizations, still the potential for churn in the role map as new tasks are defined and old tasks retired. For large maps, this is still difficult to think about coherently, so as humans we fall back to “share this folder”. UX / usability fail. Overpermissions / underprovisioning of roles based on “default” role inheritance.

Roles are still a fuzzy concept: how do you define a role? In terms of the actions it can execute. Adding an action permission actually changes the semantics of the role.

People don’t fit neatly into a role. People have multiple roles to get the permissions they need to do work.

You’ve moved the security problem around to the People-Role map, but what are your security and assurance practices with respect to this map?

Probably still need good auditing. Consider the Unix case of everyone in “wheel” or logged in as “root”. What admin did what?
Let’s take a look at something that isn’t quite RBAC, but still provides a way to give access based on a label with similar semantics to a “role”: the Unix group.