Role-Based Access Control (RBAC)

1 Motivation

With many capabilities and privileges in a system, it is difficult to manage them, such as assigning privileges to users, changing assignments, keeping track of the assignments for all users, ensuring that the assignments are not causing security problems, etc. These tasks are particular difficult in a dynamic environment, where users' privileges can change quite frequently. Role-Based Access Control (RBAC) can ease the management task.

2 RBAC: Role Based Access Control

- Traditional access control: Users/groups and access rights.
- Elements of RBAC: Users, Roles, Access Rights
- RBAC concept:
  - With role-based access control, access decisions are based on the roles that individual users have as part of an organization. Users take on assigned roles (such as doctor, nurse, teller, manager). The process of defining roles should be based on a thorough analysis of how an organization operates and should include input from a wide spectrum of users in an organization.
  - Access rights are grouped by role name, and the use of resources is restricted to individuals authorized to assume the associated role. For example, within a hospital system the role of doctor can include operations to perform diagnosis, prescribe medication, and order laboratory tests; and the role of researcher can be limited to gathering anonymous clinical information for studies.
  - The use of roles to control access can be an effective means for developing and enforcing enterprise-specific security policies, and for streamlining the security management process.
- Roles:
  - Example: Doctor, nurse, teller, manager.
  - Role hierarchies:
    - Role hierarchies are a natural way of organizing roles to reflect authority, responsibility, and competency.
    - Match the natural structure of an enterprise.
    - A role hierarchy defines roles that have unique attributes and that may contain other roles.
    - One role may implicitly include the operations that are associated with another role.
    - Example: in the healthcare situation, a role Specialist could contain the roles of Doctor and Intern. This means that members of the role Specialist are implicitly associated with the operations associated with the roles Doctor and Intern without the administrator having to explicitly list the Doctor and Intern operations. Moreover, the roles Cardiologist and Rheumatologist could each contain the Specialist role.
  - Role-Role relations:
    - Mutually exclusive: the same user is not allowed to take on both roles.

    - Inheritance: one role inherits permissions assigned to a different role.
    - These relations can be used to enforce security policies that include separation of duties and delegation of authority.
  - User-Role relationship: Assigning roles to users.

- Access rights
  - Role-Permission relationships: Access rights are grouped by role name.
  - For example, the role of doctor can include operations to perform diagnosis, prescribe medication, and order laboratory tests; the role of researcher can be limited to gathering anonymous clinical information for studies.
  - NIST Studies:
    - Permissions assigned to roles tend to change relatively slowly compared to changes in user membership of roles.
    - Assignment of users to roles will typically require less technical skill than assignment of permissions to roles.
    - Conclusion: have a predefined role-permission relationship. For example, NIST is defining roles and operations suitable for the IRS environment, Veterans Administration, etc. The process of defining roles should be based on thorough analysis of how an organization operates.

- Rules for the association of operations with roles.
  - In addition to the association of access right with roles, RBAC can also set extra rules to regulate the use of those access rights.
  - RBAC provides administrators with the capability to regulate who can perform what actions, when, from where, in what order, and in some cases under what relational circumstances.
  - Example 1: Organizations can establish the rules for the association of operations with roles.
    - For example, a healthcare provider may decide that the role of clinician must be constrained to post only the results of certain tests but not to distribute them where routing and human errors could violate a patient's right to privacy.
  - Example 2: A teller and an accounting supervisor in a bank.
    - Teller: read/write access to records.
    - Supervisor: perform correction (also need read/write access).
    - Rules 1: Supervisor cannot initiate deposits or withdrawals, but can only perform corrections after the fact.
    - Rule 2: Teller can only initiate deposits or withdrawals, but cannot perform corrections once the transaction has been completed.
  - Example 3: Operations can also be specified in a manner that can be used in the demonstration and enforcement of laws or regulations.
    - For example, a pharmacist can be provided with operations to dispense, but not to prescribe, medication.
  - Example 4:
    - Several employees may act in a manager role.

    * Rule: the role can be granted to only one employee at a time.

- Supporting three well-known security principles
  - Least privilege:
    * A user can be given no more privilege than is necessary to perform the job. This concept of least privilege requires identifying the user's job functions, determining the minimum set of privileges required to perform that function, and restricting the user to a domain with those privileges and nothing more.
  - Separation of duties: mutually exclusive
    * For example, requiring an accounting clerk and account manager to participate in issuing a check. These two roles must be mutually exclusive.
  - Data abstraction:
    * Permission can be defined at a higher level, rather than on read/write/ execute. For example, permissions can be defined on credit, debit for an account object.
    * This is in contrast to the more conventional and less intuitive process of attempting to administer lower-level access control mechanisms directly (e.g., access control lists, capabilities) on an object-by-object basis.
- Comparison with Groups
  - A group is typically treated as a collection of users and not as a collection of permissions (still ACL).
  - A role is both a collection of users on one side and a collection of permissions on the other (capability).
- Is RBAC a DAC or MAC?
  - DAC: individual user defines policies.
  - MAC: system defines mandatory policies.
  - RBAC:
    * Some treat it as an independent type of access control.
    * RBAC is policy-neutral by itself.
    * Particular configurations of RBAC can have a strong mandatory flavor, while others can have strong discretionary flavor.

3 NIST RBAC Standard

- Components
  - Core RBAC
  - Hierarchical RBAC
  - Static Separation of Duty Relations
  - Dynamic Separation of Duty Relations
- Core RBAC (Figure 1)

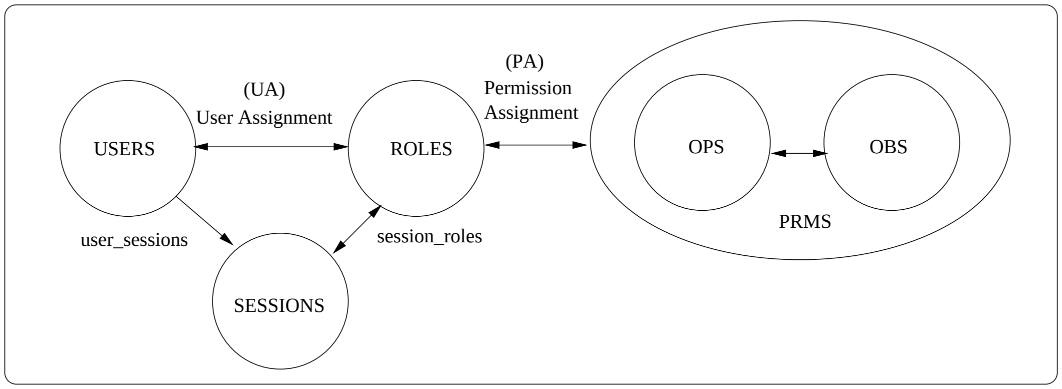

Figure 1: Core RBAC

- Users, roles, objects (OBS), operations (OPS), permissions (PRMS), and sessions.
- Sessions: each session is a mapping between a user and an activated subset of roles that are assigned to the user. Each session is associated with a single user and each user is associated with one or more sessions.
- The following figure depicts the relationships among the elements of Core RBAC (arrows depict one-to-many relationship, double-arrows depict many-to-many relationship).

- Hierarchal RBAC
  - Role hierarchies define an inheritance relation among roles. A role can inherit the permissions of another role.
  - General role hierarchies:
    * Role hierarchy can be an arbitrary partial order
    * Allow multiple inheritances.
  - Limited role hierarchies:
    * Role hierarchy can only be tree structures.
    * Disallow multiple inheritances.

- Constrained RBAC
  - Constrained RBAC adds separation of duty (SOD) relations to the RBAC model.
  - Separation of duty relations are used to enforce conflict of interest policies that organizations may employ to prevent users from exceeding a reasonable level of authority for their positions.
  - To minimize the likelihood of collusion, individuals of different skills or divergent interests are assigned to separate tasks. The motivation is to ensure that fraud and major errors cannot occur without deliberate collusion of multiple users.
  - Static Separation of Duty (SSD)
    * Enforce constraints on the assignment of users to roles.
    * Users cannot be assigned mutually exclusive roles.
  - Dynamic Separation of Duty (DSD)
    * Enforce constraints on the roles that can be activated within or across a user's sessions.

    * Users can be assigned mutually exclusive roles, but these roles cannot be active at the same time.
    * Example: a user may be authorized for both the roles of Cashier and Cashier Supervisor, where the supervisor is allowed to acknowledge corrections to a Cashier's open cash drawer. If the individual acting in the role Cashier attempted to switch to the role Cashier Supervisor, DSD would require the user to drop the Cashier role, and thereby force the closure of the cash drawer before assuming the role Cashier Supervisor.

4 Role Engineering

- What is role engineering?
  A: Applying engineering principals and techniques to define roles that implement a security policy. The defined roles should reflect the nature of the enterprise or organization.
- How to conduct role engineering?
  - Bottom-up Approach:
  - Top-down Approach:

5 Solaris RBAC Implementation (Solaris 8)

- Roles:
  - A role is a special identity that can be assumed by assigned users only.
  - Just like a user account, with a home directory, groups, password, etc.
  - Also stored at /etc/passwd, /etc/shadow.
  - Users can assume roles from the command line by using the `su` command and supplying the role password.
  - Users cannot login directly to a role
    * This is better for auditing (users cannot assume another role within one role).
  - Unlike the NIST standard, roles are not assigned to users, they can only be assumed by users via the `su` command.
  - There are no predefined roles shipped with Solaris 8. It is up to the customer site to decide what types of roles should be set up.
  - There are three roles that can be readily configured:
    * Primary administrator
    * System administrator: add users, configuring
    * System operator
- Authorizations (Permissions):
  - An authorization is a discrete right granted to a user or role.
  - Authorizations are capabilities.
  - Examples:

    - `solaris.admin.usermgr.read`: read but not write access to user configuration files.
    - `solaris.admin.usermgr.pswd`: change a user's password.
    - `solaris.admin.printer.delete`: delete a printer.
    - `solaris.admin.usermgr.grant`: delegate any of the authorizations with the same prefix to other users.
  - Authorizations are stored at /etc/security/auth_attr. Currently, it is impossible to add new authorizations.

- Right Profiles (Role-Permission Relationship):
  - A package that can be assigned to a role or user. It may consist of:
    - **Authorizations**: these are the capabilities.
    - **Commands with security attributes**: these give the users permissions to run commands with privileges (e.g. with root privileges).
      - Security attributes are the setuid functions for setting real or effective user IDs (UIDs) and group IDs (GIDs) on commands.
      - The preferred approach is to assign effective IDs, rather than real IDs. However, since shell scripts and other programs may require a real UID of root, real IDs must be available in the security attributes as well
    - **Supplementary right profiles**: these provide nested right profiles.
  - Examples: Primary Administrator
    - Intended for the most powerful role on the system
    - `solaris.*` authorization effectively assigns all of the authorizations provided by the Solaris software.
    - `solaris.grant` authorization lets a role assign any authorization to any rights profile, role, or user.
    - The `*:uid=0`; `gid=0` provides the ability to run any command with `uid=0` and `gid=0`.

```
Autorizations: solaris.*, solaris.grant
Commands:      *:uid=0; gid=0
```

  - Example: Operator: provide the ability to do backups and printer maintenance.

```
Supplementary rights profiles: Printer Management, Media Backup, All.
```

  - Example: Printer Management: A typical profile intended for a specific task area.

```
Authorizations: solaris.admin.printer.delete,
                solaris.admin.printer.modify,
                solaris.admin.printer.read
Commands: /usr/sbin/accept:euid=lp, /usr/ucb/lpq:euid=0,
             /etc/init.d/lp:euid=0, /usr/bin/lpstat:euid=0,
             /usr/lib/lp/lpsched:uid=0,
             /usr/sbin/lpfilter:euid=lp, ...
```

  - Example: Basic Solaris User Rights Profile: By default, this profile is assigned to all users through the policy.conf file.

    ```
    Authorizations: solaris.profmgr.read, solaris.jobs.usr,
                    solaris.admin.usermgr.read,
                    solaris.admin.logsvc.read, ...
    Supplementary rights profiles: All
    ```

- Database Supporting RBAC
  - `user_attr`: associates users and roles with authorizations and rights profiles.

    ```
    root::::auths=solaris.*,solaris.grant;\\
            profiles=All, Web Console Management
    lp::::profiles=Printer Management
    adm::::profiles=Log Management
    ```

  - `auth_attr`: define authorizations and their attributes.
  - `prof_attr`: defines profiles, lists the profile's assigned authorizations. This is the authorizations part of a profile
  - `exec_attr`: identifies the commands with security attributes assigned to specific rights profiles. This is the commands part of a profile.

6 SELinux RBAC Implementation

Primarily developed by the US National Security Agency, SELinux has been integrated into version 2.6 series of the Linux kernel. SELinux provides a hybrid of concepts and capabilities drawn from mandatory access controls, mandatory integrity controls, role-based access control (RBAC), and type enforcement architecture. We only focus on the RBAC part here.

- User Assignment: assigning roles to users: SELinux creates associate an identity to every process on the system. This identity is different from the UID of a process. When the UID of a process changes, the identity does not change, unless the change is specified in the policy. SELinux puts a very restrict control on the change of identities. In this way, access control decisions can be based on the correct identity. In the following example, `system_u` is an identity created by SELinux.

  ```
  user system_u roles system_r;
  user root roles { user_r sysadm_r };
  ```

- Permission Assignment: assigning permissions to users: SELinux uses a type-enforcement (TE) model for its underlying access control, so the actual permissions depend on types. Without getting into a complicated discussion here, just simply consider these types (e.g., `kernel_t, sysadm_t`) as permissions.

  ```
  role system_r types { kernel_t initrc_t getty_t klogd_t };
  role sysadm_r types { sysadm_t run_init_t };
  ```

- Transition rules: specifies authorised transitions between roles: switching to a new role has to following these rules. These rules prevents the switchings which are considered unsafe.

```
allow system_r { user_r sysadm_r };
allow user_r sysadm_r;
```

- RBAC-related commands:
  - Switch to a new role:

    ```
    % newrole -r user_r
      Authentication is needed (password)
    ```

  - check the current role:

    ```
    % id -Z
    ```

7 SUDO : An ad hoc implementation of RBAC

- `sudo` (superuser do):
  - `sudo` allows a permitted user to execute a command as the superuser or another user, as specified in the sudoers file.
  - If the invoking user is root or if the target user is the same as the invoking user, no password is required. Otherwise, sudo requires the users to authenticate themselves with a password by default.
  - Once a user has been authenticated, a timestamp is updated and the user may then use sudo without a password for a short period of time (e.g. 5 minutes).
  - `sudo` determines who is authorized user by consulting the file `/etc/sudoers`.
  - If a user who is not listed in the sudoers file tries to run a command via sudo, mail is sent to the proper authorities.
  - `sudo` can log both successful and unsuccessful attempts to syslog, a log file, or both.
  - `sudo` operates on a per-command basis; it is not a replacement for the shell.
- Figure 2 depicts an example `/etc/sudoers`: list of permissions assigned to users or groups of users.
- Examples
  - To get a file listing of a protected directory as user `operator`:
    ```
    dgb% sudo -u operator ls /usr/local/protected
    ```
    Note: `sudo` by default runs the command as root.
  - To kill a process
    ```
    dgb% sudo kill 1003
    ```

```
User_Alias  Faculties = millert, mikef, dowdy
User_Alias  Admins = alice, bob

Runas_Alias OP = root, operator

Host_Alias  ALPHA = apollo, nyx :\
            BETA = ulyssess zeus :\

Cmnd_Alias  PRINTING = /usr/sbin/lpc, /usr/bin/lprm

# on the alphas, john may su to anyone but root and flags are
# not allowed; on the betas, john can print.
john  ALPHA = /usr/bin/su [!-]*, !/usr/bin/su *root* :\
      BETA = PRINTING

# Faculties may change passwords for anyone but root on
# the BETA machines.
Faculties   BETA = /usr/bin/passwd [A-z]*, !/usr/bin/passwd root

# Admins may run anything on the ALPHA and BETA machines as any
# user listed in the Runas_Alias "OP" (i.e.: root and operator)
Admins      ALPHA=(OP) ALL : BETA=(OP) ALL

# dgb is allowed to run /bin/ls as operator,
# but /bin/kill and /usr/bin/lprm as root.
dgb   ALPHA = (operator) /bin/ls, (root) /bin/kill, /usr/bin/lprm
```

Figure 2: An example of `sudo` configuration file: `/etc/sudoers`

- To edit the index.html file as user www:
  `% sudo -u www vi  www/htdocs/index.html`

- To shutdown a machine:
  `% sudo shutdown -r +15 "quick reboot"`

- Discussion 1: Is `sudo` an implementation of RBAC?
  - Roles: `Faculties, Admins`.
  - Permissions:
    e.g. `ALPHA = /usr/bin/su [!-]*`.
  - User_Role Relationship:
    e.g. `"User_Alias   Faculties = millert, mikef, dowdy"`.
  - Role_Permission Relationship:
    e.g. `"Faculties   BETA = /usr/bin/passwd [A-z]*, ..."`.

- Discussion 2: Is `sudo` a general implementation or RBAC?
  - Permissions are defined as "ability to run commands with the root (or other users) privileges". Therefore, they are built upon commands.
  - This is more like an RBAC-based `Set-UID` mechanism. Namely, it adds an extra layer of access control on who can/cannot execute `Set-UID` programs. This extra layer of access control is based on RBAC.
  - The fundamental functionalities of RBAC are not supported, including enabling, disabling, delegating, revoking capabilities, etc.
  - There are potential security risks (see the next item).
- Security Consideration
  - `"sudo su"` or `"sudo sh"`: will `sudo` log all the subsequent commands run from the new shell? Will `sudo`'s access control affect them?
    - **No**. Since `sudo` is just a user-space implementation, it is difficult for `sudo` to log and enforce access control on the subsequent commands. A general RBAC implementation should take care of this.
    - Therefore, care must be taken when giving users access to commands via `sudo` to verify that the command does not inadvertently give the user an effective root shell.
  - Is it safe to allow a normal user to run `"sudo vi /xyz"` or `"sudo emacs /xyz"` (`vi` and `emacs` will run with the root privilege)? `/xyz` is simply a data file that can only be modified by root.
    - **No**. You can launch a shell from within the `vi` and `emacs` programs. The new shell runs as root, and is not subjected to `sudo`'s access control.
  - Like `Set-UID` programs, `sudo` needs to deal with the issues caused by environment variables, search path, dynamic and library loading.
  - Running shell scripts via `sudo` can expose the same vulnerabilities that make setuid shell scripts unsafe.