Role-Based Access Control (RBAC) Lab

Lab Description

The learning objectives of this lab are for students to discover the advantage of Role-Based Access Control (RBAC) over other access control mechanisms, and to implement the RBAC principle to enhance system security. Role based access control, as introduced in 1992 by Ferraiolo and Kuhn, has become the predominant model for advanced access control because it reduces the complexity and cost of security administration in large applications. Most information technology vendors have incorporated RBAC into their product line, and the technology is finding applications in areas ranging from health care to defense, in addition to the mainstream commerce systems for which it was designed. RBAC has also been implemented in `Fedora Linux` and Trusted Solaris. In this lab, students will develop a simplified RBAC system for `Minix`. Our simplification is based on the RBAC standard proposed by NIST [1].

Lab Tasks

In traditional `Unix` systems, there is a special type of programs called `Set-UID` programs. When the owners of these programs are root (i.e. `Set-Root-UID` programs), these programs are privileged programs; namely, a user can gain additional privileges by running these programs. In the following, we use `Set-UID` to refer to these privileged programs, rather than using `Set-Root-UID`. In this lab, we would like to use access control to protect these `Set-UID` programs, such that only users who have been granted the permissions to execute a `Set-UID` program are allowed to run the program. We will use RBAC to conduct the access control; namely, the permissions are assigned to roles, and a user needs to be in an appropriate role to be able to execute any `Set-UID` program. Your system should have the following components and functionalities:

(A) Core RBAC. Core RBAC includes five basic data elements called users (USERS), roles (ROLES), objects (OBS), operations (OPS), and permissions (PRMS). In this lab, OBS includes all the `Set-UID` programs, and OPS includes just the `execute` operation; therefore, each permission is a tuple $(execute, ob \in OBS)$. Core RBAC also includes sessions (SESSIONS), where 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. In this lab, we make a simplification, limiting each user to be associated with only one session at any time. You can use user's login session as the RBAC session; however, if the same user has two login sessions (login twice from two different windows), we still treat them as one RBAC session.

Based on the basic RBAC data elements, you should implement the following functionalities:

- **Creation and Maintenance of Roles:** Another simplification we make is that all roles are fixed after system boots up. In other words, you do not need to worry about adding/deleting roles. However, you cannot hard-code roles. Administrators should be able to add/delete roles, and the modification will take effects after system reboots. However, 10 bonus points will be given to the implementation that does not make this simplification.

- **Creation and Maintenance of Relations:** The main relations of Core RBAC are (a) user-to-role assignment relationship (UA), and (b) permission-to-role assignment relation (PA). Functions to create and delete instances of UA relations are `AssignUser` and `DeassignUser`. For PA the required functions are `GrantPermission` and `RevokePermission`. Note that these functions can only be performed by administrators with appropriate roles, and these relations can be modified during the run time.
- **Activate/Deactivate Roles:** When a user's session starts, a default set of active roles for the user will be used for such a session. The composition of this set can then be altered by the user during the session by adding or deleting roles. Functions relating to the adding and dropping of active roles are `AddActiveRole` and `DropActiveRole`.

(B) Role Hierarchies. Role hierarchies define an inheritance relation among roles. NIST standards defines two types of hierarchies: the general role hierarchies and limited role hierarchies. In this lab, you should implement the general role hierarchies. In terms of hierarchies creation and maintenance, to simplify the lab, we assume that hierarchies are static and cannot be modified on the fly. To change the hierarchies and enable them, one needs to reboot the machine. With this simplification, you can put the entire role hierarchy in a configuration file, which will be loaded into the system during the system bootup. Therefore, modification of the hierarchies can be achieved by directly modifying the configuration file. We will give 10 bonus points to the implementation that does not make such a simplification.

(C) Separation of Duty. 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. NIST RBAC standard defines two types of separation of duty relations: *Static Separation of Duty (SSD)* and *Dynamic Separation of Duty (DSD)*. SSD enforces the separation-of-duty constraints on the assignment of users to roles; for example, membership in one role may prevent the user from being a member of one or more other roles, depending on the SSD rules enforced. DSD allows a user to be assigned conflicted roles, but ensures that the conflicted roles cannot be activated simultaneously. In this lab, your system should support both SSD and DSD. Similar to the previous components, we also assume that the SSD and DSD constraints are static, and any change to the constraints may require system to reboot. However, 10 bonus points will be given to the implementation that does not make this simplification.

Design Issues

We will prepare a document to describe the entire process of executing a program. Using this document, students should be able to find the places where they can add this extra access control.

Submission

You need to submit a detailed lab report to describe what you have done and what you have observed. You also need to demonstrate your system.

References

[1] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and system Security*, 4(3):224–274, August 2001.