Role-Based Access Control as a Programming Challenge

Yanhong A. Liu*

This programming challenge description focuses on a small but rich set of problems from an important practical application domain, Role-Based Access Control (RBAC). The goal is to allow the use of a wide variety of essential programming constructs to first specify the problems clearly and then solve the problems efficiently, as much as possible.

- Role-Based Access Control (RBAC) is a security policy framework for controlling user access to resources based on roles [3, 9]. It is extremely important for reducing the cost of policy administration, especially in large organizations.
- The problems include updates, for actions and transactions, and queries, for checking, analysis, optimization, and planning, in the presence of constraints, naturally organized into a set of components for ease of use by the applications.

The RBAC programming challenge is described in the next two pages.

Among the five RBAC components described, functionalities of the first four are created based on the ANSI standard for RBAC [4, 1] but reduced to contain only the most essential concepts and improved to avoid discovered anomalies [8, 6]. Functionalities in the last component are created to correspond to role mining [2] and generalize from user-role reachability [10].

- As a programming challenge, any subset of self-contained components and functionalities can be used, and the rest can be made optional.
- Additional RBAC components and functionalities can also be added, for example, for sessions and for dynamic separation of duty (DSD) constraints in the ANSI standard [4, 1], for role mining with probabilistic models [5], and for trust management [7] (also called distributed RBAC) in decentralized systems.
- Furthermore, one may add a verification component for proving or checking the constraints, a Graphical User Interface (GUI) component, a particular RBAC policy for an RBAC system, and a test component for correctness and performance testing.

This programming challenge is created for the Workshop on Logic and Practice of Programming (LPOP) at the Federated Logic Conference (FLOC), Oxford, UK, July 18, 2018. The emphasis is on clearly expressing the problem logic first before improving the program efficiency. Any languages and systems can be used.

*Author's contact: Computer Science Department, Stony Brook University, Stony Brook, New York. Email: liu@cs.stonybrook.edu

RBAC programming challenge

We consider Role-Based Access Control (RBAC) with 5 components:

Core RBAC
Hierarchical RBAC
Core RBAC with Static Separation of Duty (SSD) constraint (a.k.a. Constrained RBAC)
Hierarchical RBAC with SSD constraint
Administrative RBAC

Core RBAC keeps several sets including the following:

`USERS`: set of users
`ROLES`: set of roles
`PERMS`: set of permissions
`UR`: set of user-role pairs
`PR`: set of permission-role pairs

with constraints:

`UR` is subset of `USERS * ROLES`
`PR` is subset of `PERMS * ROLES`

update functions for each set, subject to the constraints above:

`AddUser`, `DeleteUser`, `AddRole`, `DeleteRole`, `AddPerm`, `DeletePerm`
`AddUR`, `DeleteUR`, `AddPR`, `DeletePR`, where
each `Add` has pre-conditions: the element is not in and no constraints will be violated, and
each `Delete` has the pre-condition that the element is in, and maintains the constraints by updates if needed

and query functions including the following:

`AssignedRoles(user)`: the set of roles assigned to `user` in `UR`
`UserPermissions(user)`: the set of permissions assigned to the roles assigned to `user`
`CheckAccess(user, perm)`: whether some role is assigned to `user` and is granted `perm`

Hierarchical RBAC extends CoreRBAC and keeps also a role hierarchy:

`RH`: set of pairs of roles, called ascendant and descendant roles,
where an ascendant role inherits permissions from a descendant role

with constraints:

`RH` is subset of `ROLES * ROLES`, and `RH` is acyclic

update functions for `RH`, subject to the constraints above:

`AddInheritance(asc, desc)`, `DeleteInheritance(asc, desc)`, where
each update has the same kinds of pre-conditions as updates in CoreRBAC

and query functions including the following:

`Trans()`: the transitive closure of role hierarchy unioned with the reflexive role pairs
`AuthorizedRoles(user)`: the set of roles of `user` and their transitive descendant roles

Core RBAC with SSD extends CoreRBAC and keeps also a set of SSD items, where
each item has: a name, a set of roles, and a cardinality

with constraints:

all roles in all SSD items are in `ROLES`
for each SSD item, its cardinality is greater than 0 and less than the number of its roles
for each user, for each SSD item, the number of assigned roles (`AssignedRoles`) of the user that are in the item's set of roles is at most the item's cardinality

update functions, subject to the constraints above:

`CreateSsdSet(name, roles, c)`: add SSD item having `name`, `roles`, and cardinality `c`
`DeleteSsdSet(name)`: delete SSD item having `name`
`AddSsdRoleMember(name, role)`: add `role` to roles of SSD item having `name`
`DeleteSsdRoleMember(name, role)`: delete `role` from roles of SSD item having `name`
`SetSsdSetCardinality(name, c)`: set `c` to be cardinality of SSD item having `name`, where
each update has the same kinds of pre-conditions as updates in CoreRBAC, except that all updates have also pre-conditions that no constraints will be violated

and query functions including the following:

`SsdRoleSets()`: the set of names of SSD items
`SsdRoleSetRoles(name)`: the set of roles in SSD item having `name`
`SsdRoleSetCardinality(name)`: the cardinality of SSD item having `name`

Hierarchical RBAC with SSD extends both Hierarchical RBAC and Core RBAC with SSD and combines all from both except that the SSD constraint uses `AuthorizedRoles` in place of `AssignedRoles`

Administrative RBAC could extend each of the previous 4 components; we consider extending the last, HierarchicalRBACwithSSD, with optimization and planning functions:

`MinRoleAssignments`:
find `ROLES'`, `UR'`, and `PR'` with the smallest total size of `UR'` and `PR'`
such that each user has the same permission through `AuthorizedRoles` as before
`MinRoleAssignmentsWithHierarchy`:
find `ROLES'` , `UR'`, `PR'`, and `RH'` with the smallest total size of `UR'`, `PR'`, and `RH'`
such that each user has the same permissions through `AuthorizedRoles` as before
`GetRolesPlan(user, roles, acts)`:
find a sequence of actions, i.e., updates, in `acts` that allows `user` to get `roles`
`GetRolesShortestPlan(user, roles, acts)`:
find a shortest sequence of actions, i.e., updates, in `acts` that allows `user` to get `roles`

and an operation:

`GetRoles(user, roles, acts)`:
perform a sequence of actions in `acts` that allows `user` to get `roles` if possible

Any subset of updates can be used as `acts`. All constraints must hold after each update.

Acknowledgment

Many thanks to Scott Stoller for helpful references and discussions and to Marc Denecker and Scott Stoller for helpful comments and suggestions.

References

[1] ANSI INCITS. Role-Based Access Control. ANSI INCITS 359-2004, American National Standards Institute, International Committee for Information Technology Standards, Feb. 2004.

[2] A. Ene, W. G. Horne, N. Milosavljevic, P. Rao, R. Schreiber, and R. E. Tarjan. Fast exact and heuristic methods for role minimization problems. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*, pages 1–10, 2008.

[3] D. Ferraiolo and R. Kuhn. Role-based access control. In *Proceedings of the 15th NIST-NSA National Computer Security Conference*, pages 554–563, Blatimore, Maryland, 1992.

[4] 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 Systems Security*, 4(3):224–274, 2001.

[5] M. Frank, J. M. Buhmann, and D. A. Basin. Role mining with probabilistic models. *ACM Transactions on Information and System Security*, 15(4):1–28, 2013.

[6] N. Li, J.-W. Byun, and E. Bertino. A critique of the ANSI standard on role-based access control. *IEEE Security and Privacy*, 5(6):41–49, 2007.

[7] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust-management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130, 2002.

[8] Y. A. Liu and S. D. Stoller. Role-based access control: A corrected and simplified specification. In *Department of Defense Sponsored Information Security Research: New Methods for Protecting Against Cyber Threats*, pages 425–439. Wiley, 2007.

[9] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[10] S. D. Stoller, P. Yang, C. Ramakrishnan, and M. I. Gofman. Efficient policy analysis for administrative role based access control. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 2007.