A Framework for Implementing Role-based Access Control Using CORBA Security Service

Konstantin Beznosov and Yi Deng
{beznosov,deng}@cs.fiu.edu

Center for Advanced Distributed Systems Engineering
School of Computer Science
Florida International University

October 28, 1999
Overview

- CORBA access control model
- Definition of CORBA protection state configuration
- Framework for implementing RBAC models using CORBA Security Service
- Example configurations of CORBA protection state that support RBAC models
Problem Statement

- RBAC is getting popular and recognized by the industry and the government
  - Implementations of RBAC concepts in: Oracle, NetWare, Java, DG/UX, object-oriented systems, object-oriented databases, MS Windows NT, enterprise security management systems.
  - proposed rules on security from the DHHS include RBAC

- Significant financial investments in CS in commercial and government organizations

- It is important to foresee if CS will fully support RBAC models

- No work in the research community that has explored the potential of CS for support of RBAC reference models
Solution Overview

- Define a configuration of CORBA protection system

- Re-define RBAC models in the language of CORBA protection system

- Identify what needs to be implemented for support of $\text{RBAC}_0$-$\text{RBAC}_3$ besides CORBA security service

- Provide a check-list for users of CORBA Security Service implementations
CS: Control Points

Client

Target Object

client application access decision

target application access decision

client-side invocation access decision

target-side invocation access decision

request

ORB

request
CS: User Authentication

User Sponsor

Authenticate

Principal Authenticator

Create

Credentials Attributes

Request

ORB

Client
CS: Access Control Model
CORBA Protection State Configuration

Thirteen-tuple \((A, IM, O, R, D, C, RRM, DS, IDM, GRM, effective\_rights, combine, interface\_operation)\):

\(A\) – the set of privilege attributes.

\(IM\) – the set of operations uniquely identified by interfaces.

\(O\) – the set of distinguishable interface instances.

\(R\) – the set of rights.

\(D\) – the set of access policy domains.

\(C = \{\text{all, any}\}\) – the set of rights combiners.

\(RRM\) – required rights matrix: \([IM, \text{Rights}] \subseteq R, [IM, \text{Combinator}] \in C\).
CORBA Protection State Configuration (cont’d.)

\[ DS = \{i, d\} \] – the set of delegation states.

\[ IDM \] – the matrix of domain membership for interface instances.
\[ [D, O] \subseteq \{T,F\}, \quad [d, o] = T \implies o \in d. \]

\[ GRM \] – granted rights matrix. \( [A, D] \subseteq R. \)

**effective_rights:** \( D \times 2^A \rightarrow 2^R, \) a function mapping a set of privilege attributes in a domain to a set of effective rights.

**combine:** \( 2^D \times 2^R \rightarrow 2^R, \) a function mapping sets of rights for every domain to a set of effective rights.

**interface_operation:** \( M \times O \rightarrow IM, \) a function mapping an operation name \( m \) and an interface instance \( o \) into an interface operation.
## Correspondence between RBAC and CORBASEC Notations

<table>
<thead>
<tr>
<th>RBAC</th>
<th>CS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Meaning</strong></td>
<td><strong>Notation</strong></td>
</tr>
<tr>
<td>Users</td>
<td>U</td>
</tr>
<tr>
<td>Roles</td>
<td>R</td>
</tr>
<tr>
<td>Role</td>
<td>r</td>
</tr>
<tr>
<td>Permissions</td>
<td>P</td>
</tr>
<tr>
<td>permission</td>
<td>p</td>
</tr>
<tr>
<td>Sessions</td>
<td>S</td>
</tr>
<tr>
<td>Session</td>
<td>s</td>
</tr>
</tbody>
</table>
Original RBAC$_0$ Definition

- $U$, $R$, $P$, and $S$ (users, roles, permissions and sessions respectively)
- $PA \subseteq P \times R$, a many-to-many permission to role assignment relation
- $UA \subseteq U \times R$, a many-to-many user to role assignment relation
- $user : S \rightarrow U$, a function mapping each session $s_i$ to the single user $user(s_i)$
- $roles : P \rightarrow 2^R$, a function mapping each session $s_i$ to a set of roles $roles(s_i) \subseteq \{ r \mid (user(s_i), r) \in UA \}$ and session $s_i$ has the permissions $\bigcup_{r \in roles(s_i)} \{ p \mid (p, r) \in PA \}$
RBAC\textsubscript{0} Definition in the Language of CS

- \(U, A, R, P\) (users, attributes of type role, rights, and principals, respectively)

- \(PA \subseteq R \times A\), a many-to-many assignment of granted rights to security attributes of type role relation.

- \(UA \subseteq U \times A\), a many-to-many user to security attributes of type role assignment relation

- \(user : P \rightarrow U\), a function mapping each principal \(p_i\) to the single user \(user(p_i)\), constant for the principal lifetime, and

- \(roles : P \rightarrow 2^A\), a function mapping each principal \(p_i\) to a set of privilege attributes of type role \(roles(p_i) \subseteq \{a \mid (user(p_i), a) \in A\}\) and principal \(p_i\) has the granted rights \(\bigcup_{a \in roles(p_i)}\{r \mid (r, a) \in PA\}\)
To Support RBAC₀

1. comply with CS standard

2. provide a means to administer UA relation

3. provide a means for users to select through UserSponsor a set of roles with which they would like to activate the new principal

4. implement PrincipalAuthenticator which creates principal credentials containing privilege attributes of type role according to relation UA

5. implement PrincipalAuthenticator which creates principal credentials containing one and only one privilege attribute of type AccessId
Original RBAC\(_1\) Definition

- \(U, R, P, S, PA, UA\), and user are unchanged from RBAC\(_0\)

- \(RH \subseteq R \times R\) is a partial order on \(R\) called the role hierarchy or role dominance relation, also written as \(\geq\), and

- \(roles: S \rightarrow 2^R\) is modified from RBAC\(_0\) to require \(roles(s_i) \subseteq \{ r \mid (\exists r' \geq r) [ (users(s_i), r') \in UA ] \}\) (which can change with time) and session \(s_i\) has the permissions \(\bigcup_{r \in roles(s_i)} \{ p \mid (\exists r'' \leq r)[ (p, r'') \in PA ] \}\)
**RBAC\(_1\) Definition in CS Language**

RBAC\(_1\) is RBAC\(_0\) with role hierarchies. RBAC\(_1\) implemented in CS is formally defined as follows:

- **U, A, R, P, PA, UA and user** are unchanged from RBAC\(_0\).

- **RH \(\subseteq A \times A\)** is a partial order on R called the role hierarchy, written as \(\geq\)

- **roles : P \(\rightarrow 2^A\)** is modified from RBAC\(_0\) to require \(\text{roles}(p_i) \subseteq \{a | (\exists a' \geq a) [(\text{users}(p_i), a') \in UA]\}\) and principal \(p_i\) has the granted rights \(\bigcup_{a \in \text{roles}(p_i)} \{r | (\exists a'' \leq a) (r, a'') \in PA\}\)
Implementing $\text{RBAC}_1$

- *roles* implemented and enforced by a *Principal Authenticator*
  - A user provides a set of roles to *UserSponsor*

- The *PrincipalAuthenticator* creates new credentials of the principal
  - Credentials have roles requested by the user provided that they satisfy the definition of function *roles* for $\text{RBAC}_1$

- A valid implementation of $\text{RBAC}_1$
  - Allows a user to specify any role junior to those the user is a member of
To Support RBAC$_1$

1. Implement RBAC$_0$

2. Provide a means to administration the role hierarchy relation $RH$

3. Implement $PrincipalAuthenticator$ which creates principal credentials containing privilege attributes of type role according to relations $UA$, $RH$ as well as function $roles$
To Support RBAC$_2$

1. Implement RBAC$_0$, and

2. Implement support of constraints on UA relation user administrator tools, and

3. Implement PrincipalAuthenticator with support of constraints on functions user and roles, and

4. Enable enforcement of constraints on PA relation by security administration tools.
**RBAC\(_3\): RBAC\(_1\) + RBAC\(_2\) + RH constraints**

To support RBAC\(_3\):

1. Implement RBAC\(_1\)
2. Implement RBAC\(_2\).
3. Implement possible additional constraints on the role hierarchy.
Example Role Hierarchy

Director (DIR)

Project Lead 1 (PL1)
- Production Engineer 1 (PE1)
- Quality Engineer 1 (QE1)
  - Engineer 1 (E1)

Project Lead 2 (PL2)
- Production Engineer 2 (PE2)
- Quality Engineer 2 (QE2)
  - Engineer 2 (E2)

Engineering Department (ED)

Project 1
- Employee (E)

Project 2

Interfaces

<<Interface>>
Employee

• get_name()
• assign_to_project()
• unassign_from_project()
• add_experience()
• get_experience()
• fire()

<<Interface>>
EngineeringProject

• make_changes()
• review_changes()
• inspect_quality()
• report_problem()
• close_problem()
• create_new_release()
• get_description()
• close()
Hypothetical Access Control Policies

1. Only colleagues can lookup employee experience.

2. Everyone in the engineering department can get a description of and report problems regarding any project.

3. Engineers working on the projects can make changes and review changes.

4. Quality engineers can inspect their project quality.

5. Production engineers can create new releases.

6. Project leaders can close problems and add experience to the records of the employees in the project.

7. The director can manage employees ([un]assign from/to projects and fire) and close engineering projects.
Multiple Domain Solution

Company (C)

Engineering Department (ED)

Engineering Project 1 (EP1)

Engineering Project 2 (EP2)
Configuration of a System Protection State

A, O, C, DS, effective_rights, combine are the same as in the single domain solution.

\[ IM = \{ \text{Employee::get\_name, Employee::assign\_to\_project, Employee::unassign\_from\_project, Employee::add\_experience, Employee::get\_experience, Employee::fire, EngineeringProject::inspect\_quality, EngineeringProject::make\_changes, EngineeringProject::report\_problem, EngineeringProject::review\_changes, EngineeringProject::close, EngineeringProject::close\_problem, EngineeringProject::get\_description} \}. \]

\[ R = \{ \text{gn, atp, ufp, ae, ge, f, mc, rc, iq, rp, cp, cnr, gd, c} \}. \]

\[ D = \{ \text{C, ED, EP1, EP2} \} \]
## Required Rights Matrix (RRM)

<table>
<thead>
<tr>
<th>Operations</th>
<th>Rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee::get_name</td>
<td>gn</td>
</tr>
<tr>
<td>Employee::assign_to_project</td>
<td>atp</td>
</tr>
<tr>
<td>Employee::unassign_from_project</td>
<td>ufp</td>
</tr>
<tr>
<td>Employee::add_experience</td>
<td>ae</td>
</tr>
<tr>
<td>Employee::get_experience</td>
<td>ge</td>
</tr>
<tr>
<td>Employee::fire</td>
<td>f</td>
</tr>
<tr>
<td>EngineeringProject::get_description</td>
<td>gd</td>
</tr>
<tr>
<td>EngineeringProject::inspect_quality</td>
<td>iq</td>
</tr>
<tr>
<td>EngineeringProject::make_changes</td>
<td>mc</td>
</tr>
<tr>
<td>EngineeringProject::review_changes</td>
<td>rc</td>
</tr>
<tr>
<td>EngineeringProject::report_problem</td>
<td>rp</td>
</tr>
<tr>
<td>EngineeringProject::close_problem</td>
<td>cp</td>
</tr>
<tr>
<td>EngineeringProject::create_new_release</td>
<td>cnr</td>
</tr>
<tr>
<td>EngineeringProject::close</td>
<td>c</td>
</tr>
</tbody>
</table>
Interface Instance Domain Membership
### Interface Instance Domain Membership Matrix (IDM)

<table>
<thead>
<tr>
<th>Interface</th>
<th>Domains</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instance</strong></td>
<td><strong>C</strong></td>
</tr>
<tr>
<td>e</td>
<td>✓</td>
</tr>
<tr>
<td>ed</td>
<td>✓</td>
</tr>
<tr>
<td>e1</td>
<td>✓</td>
</tr>
<tr>
<td>pe1</td>
<td>✓</td>
</tr>
<tr>
<td>qe1</td>
<td>✓</td>
</tr>
<tr>
<td>pl1</td>
<td>✓</td>
</tr>
<tr>
<td>e2</td>
<td>✓</td>
</tr>
<tr>
<td>pe2</td>
<td>✓</td>
</tr>
<tr>
<td>qe2</td>
<td>✓</td>
</tr>
<tr>
<td>pl2</td>
<td>✓</td>
</tr>
<tr>
<td>dir</td>
<td>✓</td>
</tr>
<tr>
<td>prj1</td>
<td>✓</td>
</tr>
<tr>
<td>prj2</td>
<td>✓</td>
</tr>
</tbody>
</table>
## Granted Rights Matrix (GRM)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Rights Domains</th>
<th>C</th>
<th>ED</th>
<th>EP1</th>
<th>EP2</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>gn</td>
<td>ge</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ed</td>
<td>-</td>
<td>gd, rp</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>e1</td>
<td>-</td>
<td>-</td>
<td>mc, rc</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>pe1</td>
<td>-</td>
<td>-</td>
<td>crn</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>qe1</td>
<td>-</td>
<td>-</td>
<td>iq</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>pl1</td>
<td>-</td>
<td>-</td>
<td>cp, ae</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>e2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>mc, rc</td>
<td>-</td>
</tr>
<tr>
<td>pe2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>crn</td>
<td>-</td>
</tr>
<tr>
<td>qe2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>iq</td>
<td>-</td>
</tr>
<tr>
<td>pl2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>cp, ae</td>
<td>-</td>
</tr>
<tr>
<td>dir</td>
<td>atp, ufp, f, c</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Conclusions

- Implementations compliant with CS specification can support RBAC\textsubscript{0–3}.
  - Additional functionality non-specified by CS is required.
    * RBAC\textsubscript{1}: Implementations of PrincipalAuthenticator interface and UserSponsor need to be aware of roles and their hierarchies.
    * Support of constraints (RBAC\textsubscript{2}): a PrincipalAuthenticator has to enforce corresponding constraints.
      - Tools to administer user-to-role and role-to-rights relations are also required.

- We set up a framework for implementing as well as for assessing implementations of RBAC models using CS.
  - It provides directions for CS developers to realizing RBAC in their systems.
  - It gives criteria to users for selecting such CS implementations that support models from RBAC\textsubscript{0–3} family.