The RCL2000 Language for Specifying Role-Based Authorization Constraints

Gail-Joon Ahn

ABSTRACT

- This presentation includes
  - The first formal (and intuitive) language for role-based authorization constraints
  - A formal semantics for this language
  - Demonstration of the expressive power of the language
  - Characterization of role-based constraints into prohibition and obligation constraints

SEPARATION OF DUTY (1)

- SOD is fundamental technique for preventing fraud and errors
- Related Work
  - Enumerate several forms of SOD
  - Little work on specifying SOD in a comprehensive way

SEPARATION OF DUTY (2)

PURCHASING MANAGER

ACCOUNTING PAYABLE MANAGER
PROHIBITION

- Separation of Duty constraints

OBLIGATION

- Every faculty member must be assigned to at least one departmental committee

RESEARCH PLAN

- Need to specify these constraints
  - Language
- Show the meaning of expression
  - Formal semantics
- Expressive power of the language
  - Well-known constraints and simulations
- Analysis of the work
  - Characterization

BIG PICTURE

Constraint Specification
Constraint Analysis
Constraint Enforcement

WHO IS THE USER

- Security Researcher
- Security Policy Designer
- Security Architect

RCL 2000

- RCL 2000 (Role-based Constraints Language 2000)
- Specification Language
  - to formally express constraints in role-based systems
- Most components are built upon RBAC96
BASIC ELEMENT
(from RBAC96)

\( \diamond U : \text{a set of users} \)
\( \diamond R : \text{a set of roles} \)
  \( R \subseteq R \times R : \text{role hierarchy} \)
\( \diamond \text{OBJ} : \text{a set of objects} \)
\( \diamond \text{OP} : \text{a set of operations} \)
\( \diamond P = \text{OP} \times \text{OBJ} : \text{a set of permissions} \)
\( \diamond S : \text{a set of sessions} \)

UA : a many-to-many user-to-role assignment relation

PA : a many-to-many permissions-to-role assignment relation

SYSTEM FUNCTIONS
(from RBAC96)

\( \diamond \text{user} : R \rightarrow 2^U \)
\( \diamond \text{roles} : U \cup P \cup S \rightarrow 2^R \)
\( \diamond \text{sessions} : U \rightarrow 2^S \)
\( \diamond \text{permissions} : R \rightarrow 2^P \)
\( \diamond \text{operations} : R \times \text{OBJ} \rightarrow 2^\text{OP} \)
\( \diamond \text{object} : P \rightarrow 2^{\text{OBJ}} \)

CR : all conflicting role sets
CU : all conflicting user sets
CP : all conflicting permission sets

NON-DETERMINISTIC FUNCTIONS (beyond RBAC96)

\( \diamond \text{introduced by Chen and Sandhu (1995)} \)
\( \diamond \text{oneelement (OE)} \)
  \( \text{oneelement}(X) = x_i, \text{where } x_i \in X \)
\( \diamond \text{allother (AO)} \)
  \( \text{allother}(X) = X - \text{OE}(X) = X - \{x_i\} \)
  \( \Rightarrow \text{should occur along with OE function} \)

SYNTAX
EXAMPLES OF CONSTRAINT EXPRESSION

Conflicting roles cannot have common users
\[ |\text{roles}(\text{OE}(U)) \cap \text{OE}(\text{CR})| \leq 1 \]

Conflicting users cannot have common roles
\[ |\text{roles}(\text{OE}(\text{OE}(\text{CU}))) \cap \text{roles}(\text{AO}(\text{OE}(\text{CU})))| = \emptyset \]

Users cannot activate two conflicting roles
\[ |\text{roles}(\text{sessions}(\text{OE}(U))) \cap \text{OE}(\text{CR})| \leq 1 \]

Users cannot activate two conflicting roles in a single session
\[ |\text{roles}(\text{OE}(\text{sessions}(\text{OE}(U)))) \cap \text{OE}(\text{CR})| \leq 1 \]

FORMAL SEMANTICS

- Reduction Algorithm
  - to convert a constraint expression to a restricted form of first order predicate logic (RFOPL)
- Construction Algorithm
  - to construct a constraint expression from RFOPL

REDUCTION ALGORITHM

\[ \text{OE}(\text{OE}(\text{CR})) \cap \text{roles}(\text{OE}(U)) \Rightarrow \text{AO}(\text{OE}(\text{CR})) \cap \text{roles}(\text{OE}(U)) = \emptyset \]

1. \[ \text{OE}(\text{OE}(\text{CR})) \cap \text{roles}(\text{OE}(U)) \Rightarrow (\text{OE}(\text{CR}) \cap (\text{OE}(\text{OE}(\text{CR})))) \cap \text{roles}(\text{OE}(U)) = \emptyset \]

2. \[ \forall \text{cr} : \text{OE}(\text{CR}) \cap \text{roles}(\text{OE}(U)) \Rightarrow (\text{cr} \cap (\text{OE}(\text{CR}))) \cap \text{roles}(\text{OE}(U)) = \emptyset \]

3. \[ \forall \text{cr}, \forall \text{cr} : \exists \text{roles}(\text{OE}(U)) \Rightarrow (\text{cr} \cap (\text{r})) \cap \text{roles}(\text{OE}(U)) = \emptyset \]

4. \[ \forall \text{cr}, \forall \text{cr}, \forall \text{u} : \exists \text{roles}(\text{U}) \Rightarrow (\text{cr} \cap (\text{r})) \cap \text{roles}(\text{U}) = \emptyset \]

RFOPL STRUCTURE

- sequence part : predicate
- \[ \forall r \in R, \forall u \in U : \exists \text{roles}(u) \]
- \[ \forall x \notin x_1, \forall x \notin x_2, \forall x \notin x_3 : \exists \text{roles}(u) \]
- \[ \forall x \notin x_1, \forall x \notin x_2, \forall x \notin x_3 : \exists \text{roles}(u) \]

CONSTRUCTION ALGORITHM

\[ \forall \text{cr} : \exists \text{cr}, \forall \text{u} : \exists \text{roles}(\text{U}) \Rightarrow (\text{cr} \cap (\text{r})) \cap \text{roles}(\text{U}) = \emptyset \]

1. \[ \forall \text{cr}, \forall \text{cr} : \exists \text{roles}(\text{U}) \Rightarrow (\text{cr} \cap (\text{r})) \cap \text{roles}(\text{U}) = \emptyset \]

2. \[ \forall \text{cr} : \exists \text{cr} : \exists \text{roles}(\text{U}) \Rightarrow (\text{cr} \cap (\text{r})) \cap \text{roles}(\text{U}) = \emptyset \]

3. \[ \text{OE}(\text{OE}(\text{CR})) \cap \text{roles}(\text{OE}(U)) \Rightarrow (\text{OE}(\text{CR}) \cap (\text{OE}(\text{OE}(\text{CR})))) \cap \text{roles}(\text{OE}(U)) = \emptyset \]

4. \[ \text{OE}(\text{OE}(\text{CR})) \cap \text{roles}(\text{OE}(U)) \Rightarrow \text{AO}(\text{OE}(\text{CR})) \cap \text{roles}(\text{OE}(U)) = \emptyset \]

SOUNDNESS AND COMPLETENESS

- Theorem 1 Given RCL2000 expression \( \alpha \), \( \alpha \) can be translated into RFOPL expression \( \beta \). Also \( \alpha \) can be reconstructed from \( \beta \).

\[ C(R(\alpha)) = \alpha \]

- Theorem 2 Given RFOPL expression \( \beta \), \( \beta \) can be translated into RCL2000 expression \( \alpha \). Also \( \beta \) which is logically equivalent to \( \beta \) can be reconstructed from \( \alpha \).

\[ R(C(\beta)) = \beta' \]
SEPARATION OF DUTY
CONSTRAINTS

- Identify new SOD properties
  - Role-centric
  - User-centric
  - Permission-centric

ROLE-CENTRIC SOD
CONSTRAINT EXPRESSION

- Static SOD
  - Conflicting roles cannot have common users
  - \( U = \{u_1, u_2, \ldots, u_n\}, \quad R = \{r_1, r_2, \ldots, r_n\}, \quad CR = \{cr_1, cr_2\} : \quad cr_1 = \{r_1, r_2, r_3\}, \quad cr_2 = \{r_4, r_5, r_6\} \)
  - \( |\text{roles}(OE(U)) \cap \text{OE}(CR)| \leq 1 \)

PERMISSION-CENTRIC SOD
CONSTRAINT EXPRESSION

- SSOD-CP
  - \( |\text{permissions}(\text{roles}(OE(U))) \cap \text{OE}(CP)| \leq 1 \)
- Variations of SSOD-CP
  - SSOD-CP
    - \( |\text{permissions}(\text{OE}(R)) \cap \text{OE}(CP)| \leq 1 \)

USER-CENTRIC SOD
CONSTRAINT EXPRESSION

- SSOD-CU (User-centric)
  - SSOD-CR \( \land |\text{user}(\text{OE}(CR)) \cap \text{OE}(CU)| \leq 1 \)

DYNAMIC SOD

- User-based DSOD
  - \( |\text{roles}(\text{sessions}(OE(U))) \cap \text{OE}(CR)| \leq 1 \)
- User-based DSOD with CU
  - \( |\text{roles}(\text{sessions}(OE(OE(CU)))) \cap \text{OE}(CR)| \leq 1 \)
- Session-based DSOD
  - \( |\text{roles}(\text{sessions}(OE(U))) \cap \text{OE}(CR)| \leq 1 \)
- Session-based DSOD with CU
  - \( |\text{roles}(\text{sessions}(OE(OE(CU)))) \cap \text{OE}(CR)| \leq 1 \)

CASE STUDIES

- Lattice-based access control
- Chinese Wall policy
  - Ravi Sandhu (1992)
- Discretionary access control
  - Sandhu and Munawer (1998)
LATTICE-BASED ACCESS CONTROL

Subject s can write object o only if $\lambda(s) \leq \lambda(o)$
Subject s can read object o only if $\lambda(o) \leq \lambda(s)$

Constraints on UA: Each user is assigned to exactly two roles $x_R$ and $L_W$

AR = \{ar_1, ar_2\}
\* ar_1 = \{HR, HW\}, ar_2 = \{LR, LW\}
ASR = \{asr_1, asr_2\}
\* asr_1 = \{HR, LW\}, asr_2 = \{LR, LW\}

Constraint on UA:
\* $\text{roles} (OE(U)) = OE(ASR)$
Constraint on sessions:
\* $\text{roles} (OE(\text{sessions}(OE(U)))) = OE(AR)$

PROHIBITION CONSTRAINTS

\* Forbid the RBAC component from doing (or being) something which is not allowed to do (or be)
\* Separation of duty constraints

OBLIGATION CONSTRAINTS

\* Force the RBAC component to do (or be) something
\* LBAC-RBAC, Chinese Wall-RBAC simulation

CONSTRAINTS CHARACTERIZATION

SIMPLE PROHIBITION CONSTRAINTS

\* Type 1
\* $|expr| \leq 1$
\* Type 2
\* $expr = \emptyset$ or $|expr| = 0$
\* Type 3
\* $|expr_1| < |expr_2|$
### SIMPLE OBLIGATION CONSTRAINTS

| Type 1 | $\text{expr} = 0$ or $|\text{expr}| > 0$ |
| Type 2 | $\text{Set X} = \text{Set Y}$ |
| Type 3 | Obligation constraints $\Rightarrow$ obligation constraints |
| Type 4 | $|\text{expr}| = 1$ or $|\text{expr}| \leq 1$, $|\text{expr}| > 0$ |

### CONTRIBUTIONS

- Developed the first formal and intuitive language for role-based authorization constraints
- Provided a formal semantics for this language
- Demonstrated the expressive power of the language by:
  - Specifying well-known separation of duty constraints
  - Identifying new role-based SOD constraints
  - Showing how to specify constraints identified in the simulations of other policies in RBAC
- Characterized role-based constraints into prohibition and obligation constraints

### FUTURE WORK

- Extension of RCL 2000
  - Applying it the formalization of some realistic security policies
- Implementation Issue
  - Tool for checking syntax and semantic as well as visualization of specification
- Enforcement of constraints