The Generalized Temporal Role Based Access Control Model

Security Management
Lecture 7

March 21, 2006
Outline

- Introduction and Motivation
- Overview of the Generalized Temporal RBAC Model
- Expressiveness and Design Considerations
- Related Work
- Conclusion and Future Work
Research Motivation

- Insider attack is a major threat in organizational systems (CSI/FBI Survey)
- Traditional discretionary and mandatory access control (DAC & MAC) approaches have limitations
- Context-based access control is a critical need for emerging applications
“To realize the Department of Defense's vision for the Global Information Grid (GIG), information assurance (IA) requirements include robust identity, authentication and privilege management, policy for dynamic access control, security management, and 'persistence monitoring' or continual monitoring throughout the network, according to Daniel G. Wolf, the director of information assurance for the National Security Agency (NSA).”
Security Management in Multidomain Environment

- Local Policy Base
- Access Control Module
- User's access requests
- Application (e.g., workflow)

- Global Policy Base

- Trust Management

- Lightly-coupled (Federated system)
- Tightly-coupled

- Application (e.g., workflow)
- Access Control Module
- Local Policy Base

- Application (e.g., workflow)
- Access Control Module
- Local Policy Base
Role Based Access Control (RBAC)

- RBAC is a promising approach for addressing diverse security needs
- Access control in organizations is based on “roles that individual users take on as part of the organization”
- A role is “is a collection of permissions”
- Constraints are applied to all the links
NIST Constrained RBAC

Static Separation of Duty

Role Hierarchy (RH)

Users

Roles

Sessions

Operations

Objects

Permissions

user_sessions (one-to-many)

Dynamic Separation of Duty

UA

PA
Advantages of RBAC

- Allows efficient security management
- Supports principle of least privilege
- Separation of duty constraints
- Policy-neutral and provides generality
- Encompasses traditional discretionary and mandatory policies
- Suitable for use in multidomain environments such as digital government, multi-enterprise venture and international coalition
Time-based Access Control Requirement

- Organizational functions and services with temporal requirements
  - A part-time staff is authorized to work only between 9am-2pm on weekdays
  - A day doctor must be able to perform his/her duties between 8am-8pm
  - An external auditor needs access to organizational financial data for a period of three months
  - A video library allows access to a subscriber to view at most three movies every week
  - In an insurance company, an agent needs access to patient history until a claim has been settled
What to model in Generalized Temporal RBAC (GTRBAC)?

- Triggers and Events
- Temporal constraints
  - Roles, user-role and role-permission assignment constraints
  - Activation constraints (cardinality, active duration,..)
- Temporal role hierarchy
- Time-based Separation of duty constraints
States of a Role in GTRBAC

- **Enabled**
  - Transition to **Disabled** via disable
  - Transition to **Active** via enable

- **Disabled**
  - Transition to **Enabled** via enable
  - Transition to **Active** via disable

- **Active**
  - Transition to **Disabled** via disable
  - Transition to **Enabled** via enable

Transition labels: activate, deactivate, enable, disable
Event and Trigger

Simple events
- enable $r$                                           disable $r$
- assign$_u$ $r$ to $u$                               deassign$_u$ $r$ to $u$
- assign$_p$ $p$ to $r$                               deassign$_p$ $p$ to $r$
- activate $r$ for $u$                                deactivate $r$ for $u$

Prioritized event $pr:E$, where $pr \in$ Prios

Status
- Role, assignment status – e.g. enabled($r$); p_assigned($p$, $r$)

Triggers: $E_1, \ldots, E_n, C_1, \ldots, C_k$ $\rightarrow$  $pr:E$ after $\Delta t$, where $E_i$ are events, $C_i$ are status expressions
Example:
  enable DayDoctor $\rightarrow$ enable DoctorInTraining after 1 hour

User/administrator run-time request: $pr:E$ after $\Delta t$
Temporal Constraints: Roles, User-role and Role-permission Assignments

- **Periodic time**
  - \((I, P) : \langle [\text{begin}, \text{end}], P \rangle\) is a set of intervals
  - \(P\) is an infinite set of recurring intervals

- **Calendars:**
  - *Hours, Days, Weeks, Months, Years*

- **Examples**
  - all\(\).\!Weeks + \{2, \ldots, 6\}.\!Days + 10.\!Hours ∪ 12.\!hours
  - Daytime (9am to 9pm) of working days
Temporal Constraints: Roles, User-role and Role-permission Assignments

- **Periodicity: (I, P, pr:E)**
  - ([1/1/2000, \(\infty\]), Daytime, enable DayDoctor)
  - ([1/1/2001, \(\infty\]), \{Mon, Wed\}, assign \(_U\) DayDoctor to Smith)

- **Duration constraint: (D, pr:E)**
  - (Five hours, enable DoctorInTraining)
  - activate DayDoctor for Smith \(\rightarrow\) enable DoctorInTraining after 1 hour
Activation Time Constraints

- **Active role duration**
  - Total duration for role activation
    1. Per role: $D_{\text{active}}[D_{\text{default}}], \text{active}_R_{\text{total}} r$
    2. Per user role: $D_{\text{active}}, u, \text{active}_U_{\text{total}} r$
  - Max active role duration per activation
    1. Per role: $D_{\text{max}}, \text{active}_R_{\text{max}} r$
    2. Per user role: $D_{\text{umax}}, u, \text{active}_U_{\text{max}} r$

- **Cardinality**
  - Total number of role activations
    1. Per role: $N_{\text{active}}[N_{\text{default}}], \text{active}_R_{\text{n}} r$
    2. Per user role: $N_{\text{active}}, u, \text{active}_U_{\text{n}} r$
  - Max number of concurrent activations
    1. Per role: $N_{\text{max}}, [N_{\text{default}}], \text{active}_R_{\text{con}} r$
    2. Per user role: $N_{\text{umax}}, u, \text{active}_U_{\text{con}} r$
Example GTRBAC access policy for a healthcare system

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a. ((\text{DayTime}, \text{enable DayDoctor}), (\text{NightTime}, \text{enable NightDoctor}))</td>
</tr>
<tr>
<td></td>
<td>b. ((\text{M, W, F, assign}_u \text{ Adams to DayDoctor}), ((\text{T, Th, S, Su, assign}_u \text{ Bill to DayDoctor}); ((\text{M, W, F, assign}_u \text{ Alice to NightDoctor}), ((\text{T, Th, S, Su, assign}_u \text{ Ben to NightDoctor}))</td>
</tr>
<tr>
<td></td>
<td>c. ([10\text{am, 3pm, assign}_u \text{ Carol to DayDoctor}))</td>
</tr>
<tr>
<td>2</td>
<td>a. ((\text{assign}_u \text{ Ami to NurseInTraining}); (\text{assign}_u \text{ Elizabeth to DayNurse}))</td>
</tr>
<tr>
<td></td>
<td>b. (c_1 = (6 \text{ hours, 2 hours, enable NurseInTraining}))</td>
</tr>
<tr>
<td>3</td>
<td>a. ((\text{enable DayNurse} \rightarrow \text{enable } c_1))</td>
</tr>
<tr>
<td></td>
<td>b. ((\text{activate DayNurse for Elizabeth } \rightarrow \text{enable NurseInTraining after 10 min}))</td>
</tr>
</tbody>
</table>
Example of Activation Time Constraint

- Video library offers 600 hours of total time per week
- $A$, $B$ and $C$ subscribe for 100 hours each
- $D$ subscribes for 250 hours
- $E$ subscribes for 50 hours

\[
\begin{align*}
&A \quad \text{(Weekly, 300, 100, active}_{R_{\text{total}}} \text{ MV1)} \\
&B \quad \text{(Weekly, 250, active}_{R_{\text{total}}} \text{ MV2)} \\
&C \quad \text{(Weekly, 50, active}_{R_{\text{total}}} \text{ MV3)} \\
&D &- MV1 \\
&E &- MV2 \\
& &- MV3 \\
\end{align*}
\]
GTRBAC Execution Model for Handling Anomalies

- GTRBAC specification can encounter two possible anomalies
  - Simultaneous occurrence of conflicting events
  - Arbitrary triggering of interdependent triggers can create ambiguity

- Conflict resolution
  - Higher priority takes precedence
  - Disabling event takes precedence when the priorities of the conflicting events are the same
    - \textit{disable} \textit{r} takes precedence over \textit{enable} \textit{r}
  - More specific constraint overrides
GTRBAC Execution Model

Event Dependency Analysis

remove undesirable dependencies, policy may be flawed

Run-time action handler

Priority-based conflict resolution

Safe schedule of events

External events (run-time events)

System State
Conflicts in GTRBAC

- GTRBAC specification can generate 3 types of conflicts
  - **Type 1**: between events of same type but opposite nature,
    - e.g., enable $r$ vs. disable $r$
  - **Type 2**: between events of dissimilar types
    - e.g., activate $r$ for $u$ vs. de-assign $r$ to $u$ OR disable $r$
  - **Type 3**: between constraints
    1. $(X, \text{pr}:E)$ vs. $(X', \text{q}:E)$ or $(X', \text{q}:\text{Conf}(E))$
    2. Per-role vs. per-user-role constraints
Handling Conflicts

- **Type 1 and Type 3(a)**
  - Higher priority takes precedence
  - Disabling event takes precedence if priorities are the same
    - e.g., disable $r$ takes precedence over enable $r$

- **Type 2**
  - Activation event has lower precedence

- **Type 3(b)**
  - Per-user-role constraints takes precedence

- **Example**
  - \{H:enable $r_0$, H:disable $r_0$, VH:enable $r_1$, H:disable $r_1$, VH:(s:activate $r_1$ for $u$)\}

  - **After resolution**
    - \{H:disable $r_0$, VH:enable $r_1$, VH:(s:activate $r_1$ for $u$)\}
Ambiguous Event Dependency

- A set of triggers may give rise to ambiguous semantics
- Example:
  - $tr1$: enable $R_1 \rightarrow$ disable $R_2$
  - $tr2$: enable $R_2 \rightarrow$ disable $R_1$
- Let the runtime requests be: \{enable $R_1$; enable $R_2$\},
  1. $tr1$ fires: \{enable $R_1$; disable $R_2$\}
     (Intuitively, $tr1$ blocks $tr2$)
  2. $tr2$ fires: \{enable $R_2$; disable $R_1$\}
     (Intuitively, $tr2$ blocks $tr1$)
- Solution: Detect ambiguity using Labeled dependency graph
Dependency Graph Analysis

- **Labeled Dependency Graph**
  - Directed graph \((N, E)\)
  - \(N\): set of prioritized events that occur in the head of some trigger
  - \(E\): set of triples of the form \((X, l, Y)\)
    - For all triggers \([B \rightarrow p:E]\)
    - For all events \(E'\) in the body \(B\), and for all nodes \(q:E'\) in \(N\)
    - \(<q:E', + , p:E>\)
    - \(<r:conf(E'), -, p:E>\) for all \([r:conf(E')]\) in \(N\) such that \(q \leq r\)

- **Dependency Graph for the Example:**

```
disable R1  ─────── DISABLE ─────── disable R2
```
Safe Set of Triggers

- A set of triggers $T$ is *safe* if its labeled dependency graph has no cycles with label "-".

**Theorem:** If a $T$ is *safe*, then there exists exactly one execution model.

**Complexity of** DAG-based safeness algorithm : $\mathcal{O}(|T|^2)$. 
Role Hierarchy in GTRBAC

- Useful for efficient security management of an organization
- No previous work has addressed the effect of temporal constraints on role hierarchies
- GTRBAC-based temporal role hierarchies allow
  - Separation of permission inheritance and role activation semantics that facilitate management of access control
  - Capturing the effects of the presence of temporal constraints on hierarchically related roles
Axioms

**Axioms**: For all \( r \in \text{Roles}, u \in \text{Users}, p \in \text{Permissions}, s \in \text{Sessions}, \text{ and time instant } t \geq 0, \text{ the following implications hold:}**

1. \( p\_\text{assigned}(p, r, t) \rightarrow can\_\text{be}\_acquired(p, r, t) \)
2. \( u\_\text{assigned}(u, r, t) \rightarrow can\_\text{activate}(u, r, t) \)
3. \( can\_\text{activate}(u, r, t) \land can\_\text{be}\_\text{acquired}(p, r, t) \rightarrow can\_\text{acquire}(u, p, t) \)
Unrestricted Hierarchies
formal definitions

- **Unrestricted I-hierarchy** ($x \geq y$)
  \[
  \forall p, (x \geq y) \land \text{can\_be\_acquired}(p, y, t) \rightarrow \text{can\_be\_acquired}(p, x, t)
  \]

- **Unrestricted A-hierarchy** ($x \succeq y$)
  \[
  \forall u, (x \succeq y) \land \text{can\_activate}(u, x, t) \rightarrow \text{can\_activate}(u, y, t)
  \]

- **Unrestricted IA-hierarchy** ($x \trianglerighteq y$)
  \[
  (x \trianglerighteq y) \rightarrow (x \geq y) \land (x \succeq y)
  \]

- **Consistency Property:**
  Let $<f> \in \{\geq_t, \succeq_t, \trianglerighteq_t\}$ and $<f' > \in \{\geq_t, \succeq_t, \trianglerighteq_t\}/\{<f>\}$. Let $x$ and $y$ be distinct roles such that $x <f> y$; then the condition $\neg(y <f'> x)$ must hold.
Types of role Hierarchy

- Permission-inheritance hierarchy (I-hierarchy)
  - Senior inherits juniors’ permissions
  - User assigned to senior cannot activate juniors

- Role-Activation hierarchy (A-hierarchy)
  - Senior does not inherit juniors’ permissions
  - User assigned to senior can activate junior
  - **Advantage:** SOD constraint can be defined on hierarchically related roles

- Activation Inheritance hierarchy (IA-hierarchy)
  - Senior inherits juniors’ permissions
  - User assigned to senior can activate junior
Types of Role Hierarchy

(a) IA Hierarchy

(b) A Hierarchy

(c) I Hierarchy

Combination of roles that can be activated by $u$:

{(Software Engineer), (Software Engineer, Programmer), (Programmer)}
Weakly Restricted temporal role hierarchy

- **Weakly Restricted**: One role needs to be enabled for the inheritance/activation semantics to apply
  - In $I_W$, a user assigned to SE can inherit $P_P$ in $\tau_2$.
  - In $A_W$, a user assigned to SE can activate P in $\tau_1$. 

Enabling intervals of Software Engineer and Programmer roles
### Restricted Hierarchies

<table>
<thead>
<tr>
<th>Weakly Restricted</th>
<th>Strongly Restricted</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_w$-hierarchy</td>
<td>$I_s$-hierarchy</td>
</tr>
<tr>
<td>$(x \geq_{w,t} y)$</td>
<td>$(x \geq_{s,t} y)$</td>
</tr>
<tr>
<td>$\forall p, (x \geq_{w,t} y) \land enabled(x, t) \land can_be_acquired(p, y, t) \to can_be_acquired(p, x, t)$</td>
<td>$\forall p, (x \geq_{s,t} y) \land enabled(x, t) \land enabled(y, t) \land can_be_acquired(p, y, t) \to can_be_acquired(p, x, t)$</td>
</tr>
<tr>
<td>$A_w$-hierarchy</td>
<td>$A_s$-hierarchy</td>
</tr>
<tr>
<td>$(x \geq_{w,t} y)$</td>
<td>$(x \geq_{s,t} y)$</td>
</tr>
<tr>
<td>$\forall p, (x \geq_{w,t} y) \land enabled(y, t) \land can_activate(u, x, t) \to can_activate(u, y, t)$</td>
<td>$\forall p, (x \geq_{s,t} y) \land enabled(x, t) \land enabled(y, t) \land can_activate(u, x, t) \to can_activate(u, y, t)$</td>
</tr>
<tr>
<td>$IA_w$-hierarchy</td>
<td>$IA_s$-hierarchy</td>
</tr>
<tr>
<td>$(x \geq_{w,t} y)$</td>
<td>$(x \geq_{s,t} y)$</td>
</tr>
<tr>
<td>$(x \geq_{w,t} y) \leftrightarrow (x \geq_{s,t} y) \land (x \geq_{w,t} y)$</td>
<td>$(x \geq_{s,t} y) \leftrightarrow (x \geq_{s,t} y) \land (x \geq_{s,t} y)$</td>
</tr>
</tbody>
</table>
Temporal Role Hierarchy Example

PartTimeDoctor

7am 10am

DayDoctor

NightDoctor

9am 9pm 9am 9pm

Is

PartTimeDoctor

{(3pm-6pm), (7am-10am)}

DayDoctor

(9am-9pm)

NightDoctor

(9pm-9am)
Hierarchy Constraint Expressions

- Hierarchy $h \in \{I, A, IA\}$:
  - Periodicity: $(I, P, \text{enable } h)$,
  - Duration: $([I, P] D, D_h, \text{enable } h)$; That is $(I, P, D_h, \text{enable } h), (D, D_h, \text{enable } h)$ or $(D_h, \text{enable } h)$

Example:
($\geq_t$ is an $I$-hierarchy and $h = (\text{ProjManager} \geq_t \text{ProjEngineer})$)

$\text{enable } r \rightarrow \text{enable } h \text{ after } 10 \text{ min}$
Let $P$ be a permission set for a Software package
- Only 5 user licenses for the package has been obtained
- $P$ is assigned to Programmer role,
- Suppose, we use an activation time cardinality constraint of 5 on Programmer

Let SE be senior of Programmer
- With $I$ or $IA$-hierarchy: The use of $P$ by more than 5 users at a time can be easily violated
- With $A$-hierarchy: The use of $P$ by more than 5 users at a time is controlled

Here the cardinality constraint is said to be permission-oriented
Activation Constraints and Temporal Role Hierarchy

*Requirement:* At the most 5 nurses and 3 doctors on active duty; no restriction on permission use

- Apply cardinality of 5 on Nurse role and 3 on Doctor role

- Let Doctor be senior of Nurse

  - With $A$-hierarchy: 3 doctors and 5 nurses can be active at a time but, doctors will not be able to acquire Nurse’s permissions.
  
  - With $I$ or $IA$-hierarchy: 3 doctors and 5 nurses can be active at a time, and doctors will also acquire Nurse’s permissions.

  Here the cardinality constraint is said to be *user-oriented*
Time-based Cardinality, Dependency and Separation of Duty Constraints

- Generic cardinality expression framework
  - Status predicates to capture all the states of GTRBAC (14)
  - Evaluation function and Projection functions
- Control flow dependency constraints
- Time-based SoD constraints
  - Systematic categorization
  - Various time-based semantics
## Cardinality Constraints Examples

<table>
<thead>
<tr>
<th></th>
<th>Constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$</td>
<td>\Pi_1 \text{eval}(\text{enabled}(r, \text{&quot;t&quot;}))</td>
</tr>
<tr>
<td>2</td>
<td>$</td>
<td>\Pi_1 \text{eval}(\neg \text{enabled}(r, \text{&quot;t&quot;}))</td>
</tr>
<tr>
<td>3</td>
<td>$</td>
<td>\Pi_2 \text{eval}(\text{u_assigned}(&quot;u&quot;, r, \text{&quot;t&quot;}))</td>
</tr>
<tr>
<td>4</td>
<td>$</td>
<td>\Pi_2 \text{eval}(\text{can_activate}(&quot;u&quot;, r, \text{&quot;t&quot;}))</td>
</tr>
<tr>
<td>5</td>
<td>$(\text{Daytime},</td>
<td>\Pi_1 \text{eval}(\text{u_assignedSet}(u, \text{&quot;Nurse&quot;}, t)</td>
</tr>
</tbody>
</table>
Control Flow Dependency Constraints

- Typically used in workflow based systems
- Pre-condition Constraint
  - An event can happen *only if* another has already happened
  - \([I, P], \text{pre, } Y, pr.E \text{ after } \Delta t \text{ for } \Delta d\)
  - E.g., \((Sat, \text{pre, activate Manager for John, enable Employee})\)
- Post-condition Constraint
  - *If* an event happens then another event *must* happen
  - \([I, P], \text{post, } Y, pr.E \text{ after } \Delta t \text{ for } \Delta d\)
  - E.g., \((Sat, \text{post, activate SysAdmin for Smith, enable SysAudit after 30 min})\)
- Precedence
  - If two events happen then one must always precede another
GTRBAC Separation of Duty Constraints

- Important for real world commercial workflow applications and is generally used to prevent fraud
- Categorization of GTRBAC SoDs
  - Role enabling SoDs
  - User-role assignment SoDs
  - Role-permission assignment SoDs
  - Activation time SoDs
  - Possibilistic role activation SoDs
  - Possibilistic permission acquisition SoDs
Role Enabling and User-role Assignment SoDs

- **Role enabling SoD constraints** (*enabled()*):
  - *(I, P, EN, R)*: No two roles in *R* can be *enabled* at the same time
    \[\forall r_1, r_2 \in R, \text{SoD} \land \text{enabled}(r_1, t) \rightarrow \neg\text{enabled}(r_2, t)\]
  - *(I, P, DIS, R)*: No two roles in *R* can be *disabled* at the same time
    \[\forall r_1, r_2 \in R, \text{SoD} \land \neg\text{enabled}(r_1, t) \rightarrow \text{enabled}(r_2, t)\]

- **User assignment SoD constraints** (*u_assigned*)
  - **UAS-SoD_1** *(I, P, UAS_1, U, R)*
    - No two roles in *R* can be assigned to a user in *U* at the same time
  - **UAS-SoD_2** *(I, P, UAS_2, U, R)*
    - No two users in *U* can be assigned a role in *R* at the same time
  - **UAS-SoD_3** *(I, P, UAS_3, U, R)*
    - Different users in *U* cannot be assigned different roles in *R* at the same time
User-role assignment SoDs

- **UAS-SoD_1** does not allow \( c \); **UAS-SoD_2** does not allow \( b \); **UAS-SoD_3** does not allow \( a \)
User-assignment SoDs (Contd.)

- **UAS-SoD_4 = UAS-SoD_2 \land UAS-SoD_3**
  - Roles in \( R \) can be assigned to only one user in \( U \)
  - Example: one user must complete all the sub tasks

- **UAS-SoD_5 = UAS-SoD_1 \land UAS-SoD_3**
  - Users in \( U \) can be assigned only one role in \( R \)
  - Example: A team should be assigned only one consultancy job (e.g., role ConsultantForBankA)

- **UAS-SoD_6 = UAS-SoD_1 \land UAS-SoD_2**
  - A user in \( U \) can be assigned to only one role in \( R \) (and vice versa)
  - Example: A group of consultants should be assigned to different consultancy jobs (e.g., user \( A \) is assigned to role ConsultantForBankA, user \( B \) is assigned to role ConsultantForBankB, etc.)
Other GTRBAC SoDs

- **Activation Time SoDs** (*active()*):
  - SoDs involving active roles and sessions
  - Examples
    - No two roles in \( R \) can be in *active* state in session(s) of a user in \( U \) at the same time
    - No two users in \( U \) can have a role in \( R \) *active* at the same time

- **Possibilistic Activation SoDs** (*can_activate()*):
  - Captures implicit/explicit user assignments (A-hierarchy)
  - Similar to user assignment SoDs and

- **Possibilistic Acquisition SoDs** (*can_be_acquired()*):
  - Captures implicit/explicit permission assignment (I-hierarchy)
  - Example: A user in \( U \) cannot acquire different *permissions* in \( P \) at the same time.
Time-based Semantics of SoD Constraints

- Consider \((I, P, \text{UAS}_1, U, R)\)
  - a user in U cannot be assigned to two roles in R
- \((I, P, \text{UAS}_1, U, R)\) has various forms
  - Weak form: At an instant in \((I, P)\), if a user is assigned to a role in R, at that instant he cannot be assigned to another role in R
  - Strong form: For each interval in \((I, P)\), if there is an instant in which a user is assigned to a role, for no other instant in that interval can he be assigned to another role in R
  - Extended Strong form: At an instant in \((I, P)\), if a user is assigned to a role in R, at no other instant in \((I, P)\) can he be assigned to another role in R
X-GTRBAC
A Policy Specification Language

- An XML conformant specifcation language for GTRBAC
- Allows identity or credential based dynamic assignment of roles to users
- Allows expressing multidomain policies through role mapping
X-GTRBAC
A Policy Specification Language

<!-- Policy Definition--> ::= 
Policy [policy_id = "(value)"]>
   <PolicyName> (name) </PolicyName>
[<!-- XCredType Definition Sheet>]
[<!-- XTemporalConstraint Definition Sheet>]
<!-- XML User Sheet>
<!-- XML Role Sheet>
<!-- XML Permission Sheet>
<!-- XML User-Role Assignment>
<!-- XML Role-Permssion Assignment>
[<!-- XSoD Definition Sheet>]
[<!-- XHierachy Definition Sheet>]
[<!-- Local Policy Definitions-->]
[<!-- Policy Relationship Definitions>]
</Policy>
An XML instance of XUS

```xml
<XUS>
  <Users/>
  <User user_id="j1">
    <UserName>John</UserName>
    <CredType cred_type_id="C100">
      <type_name>Nurse</type_name>
      <CredExpr>
        <age>30</age>
        <field>ophthalmology</field>
        <level>5</level>
        <status>single</status>
      </CredExpr>
    </CredType>
    <MaxRoles>2</MaxRoles>
  </User>
  ....
  <Users/>
</XUS>
```
An XML instance of XRS

```xml
<XRS>
  <Role role_id = "R100">
    <RoleName> Nurse </RoleName>
    <Senior HType = "IA"> Eye_Doctor </Senior>
  </Role>
  <Cardinality> 8 </Cardinality>

  <Role role_id = "R200">
    <RoleName> Eye_Doctor </RoleName>
    <Junior HType = "IA"> Nurse </Junior>
    <Senior HType = "A"> Eye_Surgeon </Senior>
  </Role>
  <Cardinality> 6 </Cardinality>
</XRS>
```
Periodicity and Duration Expressions

```xml
<XTempConstDef>
  <PeriodicTimeExpr pt_expr_id="PTQuarterWeekOne"
    i_expr_id="Year2003">
    <StartTimeExpr>
      <Year>all</Year>
      <MonthSet>
        <Month>1</Month>
        <Month>4</Month>
        <Month>7</Month>
        <Month>10</Month>
      </MonthSet>
      <WeekSet>
        <Week>1</Week>
      </WeekSet>
    </StartTimeExpr>
  </PeriodicTimeExpr>
  <DurationExpr d_expr_id="SixWeeks">
    <cal>Weeks</cal>
    <len>6</len>
  </DurationExpr>
</XTempConstDef>
```
Validation support is provided by Apache Xalan XSLT engine built into JAXP
Policy Design Issue

- GTRBAC constraint set is not minimal
- Constraint design considerations
  - Usability: Clarity of Semantics, Manageability
  - Complexity of specification

<table>
<thead>
<tr>
<th>Complexity parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n.R$</td>
<td>$n$ roles</td>
</tr>
<tr>
<td>$n.S$</td>
<td>$n$ default assignments</td>
</tr>
<tr>
<td>$n.T_r$</td>
<td>$n$ temporal constraints on $(n)$ roles</td>
</tr>
<tr>
<td>$n.T_{ur}$ ($n.T_{rp}$)</td>
<td>$n$ temporal constraints on user- assignment</td>
</tr>
<tr>
<td>$n.A_{ur}$ ($n.A_r$)</td>
<td>$n$ per-user-role (per-role) time constraint</td>
</tr>
<tr>
<td>$n.H$</td>
<td>$n.H$ indicates $n$ hierarchical relations</td>
</tr>
</tbody>
</table>
Activity-equivalence

- GTRBAC Configuration \( C \):
  - \((T, \text{Users, Roles, Permissions, RH}),\) where \( T \) is the set of constraints, RH is a \( A \)-hierarchy.

- Activity-equivalent configurations
  - \( C_1 \approx C_2 \):
    \[
    (u \Rightarrow p) \iff (u \Rightarrow p)
    \]

- \( u \) can acquire \( p \) at \( t \) in \( C_1 \)

- There exists a minimum set of constraint types
GTRBAC Family of Models

Level 0
Minimal Model

Level 1
- \( GTRBAC_{1, A} \) (Per-user-role constraint)
- \( GTRBAC_{1, P} \) (role-permission assignment constraint)
- \( GTRBAC_{1, U} \) (user-role assignment constraint)

Level 2
- \( GTRBAC_{2} \) (All constraints)

Alternative 1: \( n.S + n.T + n.R \)

Alternative 2: \( p.S + s.T + s.R, \) where \( 1 \leq p \leq n2^{n-1}; \ 1 \leq s \leq (2^n - 1) \)
Approach to transformation

- Temporal constraint on user-role assignment is changed to role enabling constraint

- Two approaches
  - Simple approach:
    - \( PE_u \) of each \( u-r \) assignment becomes enabling constraint for \( r \)
  - Minimal Disjoint Set Approach:
    - \( PE_u \) of each \( u-r \) assignment becomes enabling time constraint for a set of new roles
      - First compute Minimal Disjoint Set (MDS) of all \( PE_i \)s
      - Create new roles associated with each element of MDS
        - User \( u \) is assigned to new roles that have elements of MDS that correspond to \( PE_u \)

- Same for the permission role assignment
MDS Approach: Relations on periodic expressions

- **Containment**: PE1 is contained in PE2
  - All instants in PE1 is in PE2
- **Equivalence**: PE1 and PE2 have same time instants
- **Overlapping**: At least one time instant is common to both PE1 and PE2
- **Disjoint**: No common time instants in PE1 and PE2
Minimal Disjoint Set (MDS) & Minimal Subset (MS) of PE over MDS

- Let \( PE = \{PE_1, PE_2, .., PE_n\} \)
  - \( MDS_{PE} = \min_m\{\{PE'_i | 1 \leq i \leq m\} \) such that
    - \( PE'_1, PE'_2, ..., PE'_m \) are pair-wise disjoint
    - \( PE'_1 \cup PE'_2 \cup ... \cup PE'_m = PE_1 \cup PE_2 \cup ... \cup PE_n, \)
    - If \( PE'_i \) contains a time instant of \( PE_j \) then it does not contain a time instant that is not in \( PE_j \)

- Bounds: \( 1 \leq |MDS_{PE}| \leq (2^n - 1) \)

- MS of \( PE_j \in PE \) over the MDS\(_{PE} \)
  - subset of \( MDS_{PE} \) that collectively contains all the time instants of \( PE_j \)
  - Bounds: \( n \leq \sum |MS_{PE_i}| \leq n2^{n-1} \)
Example

- Let $PE = \{PE_A, PE_B, PE_C, PE_D, PE_E\}$, where
  - $PE_A = \{\text{Sun, Mon, Tue, Wed, Thu, Fri}\}$,
  - $PE_B = \{\text{Sun, Tue}\}$,
  - $PE_C = \{\text{Sun, Tue, Thu, Fri}\}$,
  - $PE_D = \{\text{Sun, Mon, Tue, Wed, Sat}\}$,
  - $PE_E = \{\text{Thu, Fri}\}$.
- $MDS$ of $PE$ is
  - $\{PE_1, PE_2, PE_3, PE_4\} = \{\{\text{Sun, Tue}\}, \{\text{Thu, Fri}\}, \{\text{Mon, Wed}\}, \{\text{Sat}\}\}$
- MS values are as follows:
  - $MS$ of $PE_A$ : $\{PE_1, PE_2, PE_3\}$
  - $MS$ of $PE_B$ : $\{PE_1\}$
  - $MS$ of $PE_C$ : $\{PE_1, PE_2\}$
  - $PE_D$ : $\{PE_1, PE_3, PE_4\}$
  - $PE_E$ : $\{PE_2\}$
Algorithm TransformMDS - Replacing temporal constraint on user-role assignment by temporal roles

Input: $C_{in}$  
Output: $C_{out}$  

1. $C_{out} = \{T', \text{Roles}', RH'\} = C_{in} = \{T, \text{Roles, RH}\}$;  
2. FOR each $r \in \text{Roles DO}$  
3. Let $PE = \{PE_1, PE_2, ..., PE_n\} \& U = \{u_1, u_2, ..., u_n\}$ be s.t. $(PE_i, \text{assign } r \text{ to } u_i) \in T'$;  
4. Compute $MDS$ of $PE$; Let the computed $MDS = \{PE'_1, PE'_2, ..., PE'_n\}$;  
5. FOR $i = 1$ to $n$ DO  
6. Compute $MS_{PE_i}$ for $PE_i$  
8. FOR each $PE'_i \in MDS$ DO  
9. Create a unique role $r_i$;  
10. FOR all $u_k \in U$ such that $PE'_i \in MS_{PE_k}$ for $PE_k$ DO  
11. Add default assignment (assign $r_i$ to $u_k$) in $T'$;  
12. Add constraint ($PE'_i, \text{enable } r_i$) in $T'$;  
13. Remove constraint ($PE_i, \text{assign } r \text{ to } u_i$) from $T'$;  
14. Roles' = Roles' $\cup \{r_i\}$;  
15. $RH' = RH' \cup \{r < r_i\}$;
Simple & MDS approaches

- Compute MDS
- Create new roles
- Compute MS
- Assign users to MS elements
- Change triggers
- Change hierarchies

Replace temporal constraints on p-r by that on role-enabling
- Change triggers
- Change hierarchies

Replace temporal constraints on u-r by that on role-enabling
- Change triggers
- Change hierarchies
Replace activation constraints

- Algo: transformMDS
- Algo: transformPR
- Algo: transformUR

\( C_{in} \)

\( C_{out} \approx C_{in} \)

Level 0
Example

(a) Simple

Day Doctor

PE_A = {Sun, Mon, Tue, Wed, Thu, Fri}
PE_B = {Sun, Tue}
PE_C = {Sun, Tue, Thu, Fri}
PE_D = {Sun, Mon, Tue, Wed, Sat}
PE_E = {Thu, Fri}

(b) TransformMDS

Day Doctor

PE_A = {Sun, Mon, Tue, Wed, Thu, Fri}
PE_B = {Sun, Tue}
PE_C = {Sun, Tue, Thu, Fri}
PE_D = {Sun, Mon, Tue, Wed, Sat}
PE_E = {Thu, Fri}

(c) Alternative

Day Doctor

PE''_1 = {Sun, Tue}
PE''_2 = {Thu, Fri}
PE''_3 = {Mon, Wed}
PE''_4 = {Sat}

Alternative (c): \( p.S + s.T + s.R + s.H \)
where \( n \leq p \leq n2^n-1; \ 1 \leq s \leq (2^n -1) \)

Alternative (b): \( n.S + n.T + n.R + n.H \)
Design Guidelines

- \( \text{GTRBAC}_{1,\text{U}} \) is better than \textit{alternative b} as the policy representation is less complex in terms of number of roles and hierarchies.
- \( \text{GTRBAC}_{1,\text{U}} \) is flexible – one can schedule role enabling and user assignments separately.
- when \( p_n \) and \( s_n \) are close to \( n \) and 1, \textit{alternative c} may be better than \( \text{GTRBAC}_{1,\text{U}} \) representation.

Alternative (b):
\[
\]

Alternative (c):
\[
p.S + s.T_R + s.R + s.H
\]
where
\[
n \leq p_n \leq n2^{n-1};
\]
\[
1 \leq s_n \leq (2^n - 1)
\]
Replacing *per-user-role* by *per-role*

Alternative (a): \( n.A_{ur}, \text{ where } A_{ur} \text{ user-role activation} \)

GTRBAC$_{1,\lambda}$ representation: $(n_x - n_y)A_{UR} + n_yA_R + c(b.n_y + 1)(R + H)$;
where, (1) $n_x = |D_m|$ and $n_y = |D'|$, such that (1) $D' \subseteq D_{m'}$ and (2) if $d \in D'$ then $C_m(d) > 1$; (2) $b = 1$ if $(n > n_x)$; $b = 0$ otherwise; (3) $c = 1$ if $(n > n_x > 0)$; $c = 0$ otherwise.
Design Guidelines

- Per-role constraint with default value
  - If there are many common durations,
- Per-user-role constraint
  - Per-user requirements vary significantly
  - More flexibility (e.g., requirements vary constantly)
- Hybrid approach (b in previous slide) can give balanced representations
Related Work

- OASIS RBAC Model [Bacon, 02]
  - Precondition on role activation to support active security
  - No triggers
- RSL2000 Constraint Specification Language [Ahn et. al., 2000],
  - Need for activation hierarchy [Sandhu, 1999]
    - Identified its usefulness in expressing MAC
- Separation Duty Constraints
  - Listing of useful set of SoD Constraints
    - [Simon et. al., 1997],[Gligor et. al., 1998]
- None address timing issues
- GTRBAC SoDs subsume all the SoD constraints identified
- GTRBAC Triggers and SoDs provide a technical foundation for enforcing history based SoD constraints
Conclusion

- Role based access control can be used to support diverse set of access control requirements
- Time-based access is a crucial requirement in emerging applications
- GTRBAC’s constraint set is not minimal—however, they are beneficial for practical use
Current and Future work

- Secure Interoperation
  - Integer Programming approach (for tightly coupled environments)
  - Trust-based access management (loosely coupled environment)
    - Issues related to Grid, P2P
- GTRBAC - extended to LoT-RBAC
  - Implementation in mobile environment (near completion)
- Policy evolution and hybrid hierarchy management
  - Administrative tools and techniques
- Extension of the policy design work to generate tools for efficient RBAC policy administration
Relevant References


