Chapter 10: 
Role-Based Access Control

Raimundas Matulevičius 
University of Tartu, Estonia, rma@ut.ee

Goals

• Introduce principles of role-based access control (RBAC)
• Present requirements for RBAC solution development and administration
• Discuss how SecureUML and UMLsec could be used to define RBAC policies
• Overview principles of model driven security
Outline

• Principles of role-based access control
• RBAC implementation requirements
• RBAC modelling languages
  – SecureUML
  – UMLsec
  – Language comparison
  – Transformation

• Model-driven security
  – Model-driven development
  – Security model transformation

• Further reading
RBAC:
Role-based Access Control

**Access** – a specific type of interaction between a subject and an object that result in the flow of information from one to the other

**Access control** – the process of limiting access to the resources of a system only to authorised programs, processes or other systems
RBAC\textsubscript{0}

User - any person who interacts directly with a computer system

Sandhu and Coyne, 1996; Ferraiolo et al., 2001
© Springer International Publishing AG 2017
RBAC\textsubscript{0}

**User** - any person who interacts directly with a computer system

**Role** – a job function within the organisation that describes the authority and responsibility conferred on a user assigned to the role

**Session** – a mapping between a user and an activated subset of roles the user is assigned to

---

Sandhu and Coyne, 1996; Ferraiolo et al., 2001

© Springer International Publishing AG 2017
**Subject** - an active entity that causes information to flow among objects or changes the system state

Sandhu and Coyne, 1996; Ferraiolo et al., 2001

© Springer International Publishing AG 2017
RBAC

User - any person who interacts directly with a computer system

Role - a job function within the organisation that describes the authority and responsibility conferred on a user assigned to the role

Session - a mapping between a user and an activated subset of roles the user is assigned to

Subject - an active entity that causes information to flow among objects or changes the system state

Object - a passive entity that contains or receives information

Role hierarchy - a partial order relationship established among roles

Constraint - a relationship among roles

RBAC family

- **RBAC\(_0\)**
  - Everything except role hierarchies and constraints

- **RBAC\(_1\)**
  - RBAC\(_0\) plus role hierarchies

- **RBAC\(_2\)**
  - RBAC\(_0\) plus role constraints

- **RBAC\(_3\)**
  - RBAC\(_1\) plus RBAC\(_2\)
Outline

- Principles of role-based access control
- **RBAC implementation requirements**
  - RBAC modelling languages
    - SecureUML
    - UMLsec
    - Language comparison
    - Transformation
- Model-driven security
  - Model-driven development
  - Security model transformation
- Further reading

Implementation requirements

**System administrator** – the individual who establishes the system security policies, performs the administrative roles and reviews the system audit trail

- **Operations** and **Objects** are considered predefined by the underlying system
- **Administrator**
  - manage **Users, Roles**
  - create assignment relationships
  - establish relationships between **Roles** and secured **Operations** and **Objects**.

Sandhu and Coyne, 1996; Ferraiolo et al., 2001
Implementation requirements

• **To activate RBAC**
  – **create session**
    • for creating a user session and assigning the user with a default set of roles
  – **add role**
    • for creating new roles for the current session
  – **drop role**
    • for deleting a role from the role set for the current session
  – **check access**
    • for determining if the session user has permission to perform the requested operation on an object

Sandhu and Coyne, 1996; Ferraiolo et al., 2001

© Springer International Publishing AG 2017

Implementation requirements

• **User Assignment and Permission Assignment**
  – **view assigned users**
    • for displaying a set of users assigned to a given role
  – **view assigned roles**
    • for displaying a set of roles assigned to a given user
  – **view role permissions**
    • for displaying a set of permissions granted to a given role
  – **view user permissions**
    • for displaying a set of permissions a given user gets through his or her assigned roles

Sandhu and Coyne, 1996; Ferraiolo et al., 2001

© Springer International Publishing AG 2017
Implementation requirements

- **User Assignment** and **Permission Assignment**
  - **view session roles**
    - for displaying a set of roles associated with a session
  - **view session permissions**
    - for displaying a set of permissions available in the session
  - **view role operations on object**
    - for displaying a set of operations a given role may perform on a given object; and
  - **view user operations on object**
    - for displaying a set of operations a given user may perform on a given object

Sandhu and Coyne, 1996; Ferraiolo et al., 2001

Outline

- Principles of role-based access control
- RBAC implementation requirements
  - **RBAC modelling languages**
    - SecureUML
    - UMLsec
    - Language comparison
    - Transformation
- Model-driven security
  - Model-driven development
  - Security model transformation
- Further reading
Security Modelling Languages

Outline

• Principles of role-based access control
• RBAC implementation requirements
• RBAC modelling languages
  – SecureUML
    – UMLsec
    – Language comparison
    – Transformation
• Model-driven security
  – Model-driven development
  – Security model transformation
• Further reading
SecureUML

- **Extension of the UML class diagrams**
  - Stereotypes
  - Tagged values
  - Authentication constraints
- **Based on the RBAC model**
SecureUML

Access Rules

- Security actions

© Springer International Publishing AG 2017
Authorisation Constraints

AC#1:

context Game::createGame(): void
pre: self.responsibleFFE.assignedUser →
exists(i | i.assignedUser = 'Bob')

AC#2:

context Game::updateConfirmation(): void
pre: self.responsibleFFE.assignedUser →
exists(i | i.assignedUser = 'Bob')

Outline

• Principles of role-based access control
• RBAC implementation requirements
• RBAC modelling languages
  – SecureUML
  – UMLsec
  – Language comparison
  – Transformation
• Model-driven security
  – Model-driven development
  – Security model transformation
• Further reading
UMLsec

- Extension of the UML diagrams:
  - Stereotypes;
  - Tagged values;
  - Authentication constraints

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Base class</th>
<th>Tags</th>
<th>Constraints</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fair exchange</td>
<td>subsystem</td>
<td>start, stop, adversary</td>
<td>after start eventually reach stop</td>
<td>Enforce fair exchange</td>
</tr>
<tr>
<td>smart card</td>
<td>node</td>
<td>-</td>
<td>-</td>
<td>smart card node</td>
</tr>
<tr>
<td>data security</td>
<td>subsystem</td>
<td>adversary, integrity, authenticity</td>
<td>Provides secrecy, integrity, authenticity, freshness</td>
<td>Basic data security constraints</td>
</tr>
<tr>
<td>rbac</td>
<td>subsystem</td>
<td>protected, role, right</td>
<td>only permitted activities executed</td>
<td>enforces RBAC</td>
</tr>
</tbody>
</table>
• \{\texttt{protected} = \texttt{protected}\_action}\}
• \{\texttt{role} = (\texttt{actor}, \texttt{role})\}
• \{\texttt{right} = (\texttt{role}, \texttt{protected}\_action)\}

\{\texttt{protected} = \texttt{Create game}\}  \\  \\
\{\texttt{role} = (\texttt{Bob, FootballFederationEmployee})\}  \\
\{\texttt{right} = (\texttt{FootballFederationEmployee, Create game})\}
Outline

• Principles of role-based access control
• RBAC implementation requirements
• RBAC modelling languages
  – SecureUML
  – UMLsec
  – Language comparison
    – Transformation
• Model-driven security
  – Model-driven development
  – Security model transformation
• Further reading

Language comparison

Extension Mechanism

<table>
<thead>
<tr>
<th>Criteria</th>
<th>SecureUML</th>
<th>UMLsec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta-model</td>
<td>Explicit based on the RBAC model</td>
<td>Not explicit as the UML profile extension</td>
</tr>
<tr>
<td>UML profile</td>
<td>Mainly class diagram</td>
<td>The whole UML profile</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i.e., use cases, class, activity, state, component, and other diagrams</td>
</tr>
<tr>
<td>Extension</td>
<td>Stereotypes, tagged values and authentication</td>
<td>Stereotypes, tagged values and constraints</td>
</tr>
<tr>
<td></td>
<td>constraints</td>
<td></td>
</tr>
<tr>
<td>Constraints</td>
<td>OCL</td>
<td>Constraint language is not identified</td>
</tr>
</tbody>
</table>
## Language comparison

### Modelling Targets and Application Guidelines

<table>
<thead>
<tr>
<th>Criteria</th>
<th>SecureUML</th>
<th>UMLsec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security criteria</td>
<td>Not identified</td>
<td>Confidentiality, integrity (and derived ones, like authenticity and others)</td>
</tr>
<tr>
<td>Security requirements / controls</td>
<td>RBAC</td>
<td>RBAC but also non-repudiations, secure communication links, secrecy and integrity, authenticity, freshness, secure information flows, guard access</td>
</tr>
<tr>
<td>Method</td>
<td>Development of the RBAC models</td>
<td>Not explicit but implicitly supports standard security management methods</td>
</tr>
</tbody>
</table>

### Construct Semantics

<table>
<thead>
<tr>
<th>RBAC concepts</th>
<th>SecureUML</th>
<th>UMLsec</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>Class stereotype «secuml.user»</td>
<td>Actor value of association tag {role}</td>
</tr>
<tr>
<td>User assignment</td>
<td>Dependency stereotype «assignment»</td>
<td>Associated tag {role}</td>
</tr>
<tr>
<td>Roles</td>
<td>Class stereotype «secuml.role»</td>
<td>Activity partition, Role value of association tag {role}</td>
</tr>
<tr>
<td>Permission assignment</td>
<td>Association class stereotype «secuml.permission»</td>
<td>Action, Associated tag {right}</td>
</tr>
<tr>
<td>Object</td>
<td>Class stereotype «secuml.resource»</td>
<td>Activity partition</td>
</tr>
<tr>
<td>Operation</td>
<td>Operation of «secuml.resource» class</td>
<td>Action, Associated tag {protected}</td>
</tr>
<tr>
<td>Permission</td>
<td>Authorisation constraints</td>
<td>Not defined</td>
</tr>
</tbody>
</table>
Outline

• Principles of role-based access control
• RBAC implementation requirements
• RBAC modelling languages
  – SecureUML
  – UMLsec
  – Language comparison
  – Transformation
• Model-driven security
  – Model-driven development
  – Security model transformation
• Further reading
SecureUML model

- **SU.1**: A class with a stereotype `<<secuml.resource>>` is transformed to an activity partition in the UMLsec model.
- Operations of this class become *actions* belonging to this partition.
  - each operation becomes a value the UMLsec associated tag `{protected}`.

```plaintext
{protected = (createGame)}
{protected = (updateGameReport)}
{protected = (updateConfirmation)}
```
• **SU.2**: A relationship with a stereotype `<<assignment>>` relationship used to connect users and their roles is transformed to an associated tag `{role}`

```
{role} = (Bob, FootballFederationEmployee)
{role} = (John, Umpire)
{role} = (Karl, Umpire)
```

• **SU.3**: A class with the stereotype `<<secuml.roles>>` is transformed to the UMLsec activity partition
  
  - The attributes of an association class that connects the `<<secuml.roles>>` class with `<<secuml.resource>>` class, become actions in the corresponding activity partition
**SU.4**: The association class with the stereotype <<secuml.permission>> defines the role value for the associated tag \(\texttt{right}\).

- The value of \(\texttt{right}\) can be determined from the authorisation constraint defined for the attribute of the SecureUML association class.

\[
\{\texttt{right} = (\texttt{FootballFederationEmployee}, \texttt{createGame})\} \\
\{\texttt{right} = (\texttt{Umpire}, \texttt{updateGameReport})\}
\]

**SU.5**: Received activity diagram is annotated with the <<rbac>> stereotype.
Finish the transformation manually

- Define initial and final activity nodes
- Identify logical sequence of activities
  - Specify missing control flows
  - Identify missing conditions
- Define missing and assembly existing association tags
UMLsec model

{\text{protected} = (\text{createGame})} \\
{\text{right} = (\text{FootballFederationEmployee}, \text{createGame})} \\
{\text{role} = (\text{Bob, FootballFederationEmployee})}

{\text{protected} = (\text{updateGameReport})} \\
{\text{right} = (\text{FootballFederationEmployee}, \text{updateGameReport})} \\
{\text{role} = (\text{Bob, FootballFederationEmployee})}

{\text{protected} = (\text{updateConfirmation})} \\
{\text{right} = (\text{FootballFederationEmployee}, \text{updateConfirmation})} \\
{\text{role} = (\text{Bob, FootballFederationEmployee})}

{\text{protected} = (\text{updateGameReport})} \\
{\text{right} = (\text{Umpire, updateGameReport})} \\
{\text{role} = (\text{John, Umpire})}

{\text{protected} = (\text{updateGameReport})} \\
{\text{right} = (\text{Umpire, updateGameReport})} \\
{\text{role} = (\text{Karl, Umpire})}

Secure UML model

UMLsec model
{\textbf{protected} = \textit{Create game}}
{\textbf{role} = (Bob, FootballFederationEmployee)}
{\textbf{right} = (FootballFederationEmployee, Create game)}

{\textbf{protected} = \textit{Update game report}}
{\textbf{role} = (John, Umpire)}
{\textbf{right} = (Umpire, Update game report)}

{\textbf{protected} = \textit{Update game report}}
{\textbf{role} = (Karl, Umpire)}
{\textbf{right} = (Umpire, Update game report)}

{\textbf{protected} = \textit{Update confirmation}}
{\textbf{role} = (Bob, FootballFederationEmployee)}
{\textbf{right} = (FootballFederationEmployee, Update confirmation)}
Association tags `{protected}` allow identifying the operations that belong to a secured resource. The activity partitions which hold these operations are transformed to the SecureUML class with a stereotype `secuml.resource`.

US1. Association tags `{protected}` allow identifying the operations that belong to a secured resource.

- The activity partitions which hold these operations are transformed to the SecureUML class with a stereotype `secuml.resource`.

US2. The UMLsec activity partitions which do not hold secured protected actions can be transformed to `secuml.role` stereotyped classes.

© Springer International Publishing AG 2017
US3. Association tag \{\texttt{role}\} allows identifying the «assignment» dependency relationship between classed with a stereotype «secuml.user» and their «secuml.role» stereotypes

\[
\text{\{role} = (\text{Bob, FootballFederationEmployee})
\]
\[
\text{\{role} = (\text{John, Umpire})
\]
\[
\text{\{role} = (\text{Karl, Umpire})
\]

US4. From UMLsec association tag \{\texttt{right}\} we are able to identify on which operations the role can perform security actions. Thus, from each occurrence of this association tag in the SecureUML model, a corresponding association class between a «umlsec.role» and a «umlsec.resource» is introduced

\[
\text{\{right} = (\text{FootballFederationEmployee, Create game})
\]
\[
\text{\{right} = (\text{Umpire, Update game report})
\]
\[
\text{\{right} = (\text{FootballFederationEmployee, Update confirmation})
\]
US5. In the UMLsec activity diagram it is possible to identify the security actions that are carried towards the secured operations: these are unprotected actions performed before the protected ones.

SecureUML model
Finish the transformation manually

- Attributes of the «umlsct.resource» class that define the state of the secured resource(s)
- Names for the association classes
- Multiplicities for all the association relationships
- Necessary authorisation constraints

SecureUML model
Two approaches complement each other by providing different viewpoints to the secure software design.

Outline

- Principles of role-based access control
- RBAC implementation requirements
- RBAC modelling languages
  - SecureUML
  - UMLsec
  - Language comparison
  - Transformation
- Model-driven security
  - Model-driven development
- Security model transformation
- Further reading

© Springer International Publishing AG 2017
Model Driven Development

- Definition of the system/software model
- Systematic development of the set of the transformation rules
- Application of these rules to generate executable software code from the model

Model Driven Security

- Security model is translated to security code
- Software code and security code are generated into system architectures
Outline

- Principles of role-based access control
- RBAC implementation requirements
- RBAC modelling languages
  - SecureUML
  - UMLsec
  - Language comparison
  - Transformation
- Model-driven security
  - Model-driven development
  - Security model transformation
- Further reading

Security Model
Security transformation rules

```
--- Imported common-sql.vtl
CREATE OR REPLACE TRIGGER Game_sec_insert_trg
INSTED OF INSERT ON Game
REFERENCING NEW AS NEW
FOR EACH ROW
DECLARE
  ex_denied EXCEPTION;
BEGIN
  IF sec.is_role('FootballFederationEmployee') = 'Y' AND
    sec.FootballFederationEmployeeAuthConstraint(self.id)="Y"
  THEN
    INSERT INTO Game (gameInfo, gameReport, confirmation)
    VALUES (:NEW.gameInfo, :NEW.gameReport, :NEW.confirmation);
  ELSE
    RAISE ex_denied;
  END IF;
EXCEPTION
  WHEN ex_denied THEN
    raise_application_error (-20000, 'Access denied!');
END;
```
**Resource** that needs to be secure

- gameInfo
- gameReport
- confirmation

**Security action**

- Insert
Checking the **Role** and …

**Insert security constraint**

```
-- Imported common.sql.vtl
CREATE OR REPLACE TRIGGER Game_sec_insert_trg
INSTEAD OF INSERT ON Game
REFERENCING NEW AS NEW
FOR EACH ROW
DECLARE
  ex_denied EXCEPTION;
BEGIN
  IF sec.is_role("FootballFederationEmployee") = "Y" AND
     sec.FootballFederationEmployeeAuthConstraint(self.id)="Y"
  THEN
    INSERT INTO Game (gameInfo, gameReport, confirmation)
    VALUES (:NEW.gameInfo, :NEW.gameReport, :NEW.confirmation);
  ELSE
    RAISE ex_denied;
  END IF;
EXCEPTION
  WHEN ex_denied THEN
    raise_application_error (-20000, 'Access denied!');
END;
```

... the **authorisation constraint**

```
-- Imported common.sql.vtl
CREATE OR REPLACE TRIGGER Game_sec_insert_trg
INSTEAD OF INSERT ON Game
REFERENCING NEW AS NEW
FOR EACH ROW
DECLARE
  ex_denied EXCEPTION;
BEGIN
  IF sec.is_role("FootballFederationEmployee") = "Y" AND
     sec.FootballFederationEmployeeAuthConstraint(self.id)="Y"
  THEN
    INSERT INTO Game (gameInfo, gameReport, confirmation)
    VALUES (:NEW.gameInfo, :NEW.gameReport, :NEW.confirmation);
  ELSE
    RAISE ex_denied;
  END IF;
EXCEPTION
  WHEN ex_denied THEN
    raise_application_error (-20000, 'Access denied!');
END;
```
Model Driven Security
Applying Authorisation Constraints

- GameSecInsertTrg
  - GAME_SEC_INSERT_TRG
  - CREATE OR REPLACE TRIGGER
  - INSTEAD OF INSERT ON Game
  - REFERENCING NEW AS NEW
  - FOR EACH ROW
  - DECLARE
  - ex_denied EXCEPTION;
  - BEGIN
  - IF sec.is_role("FootballFederationEmployee") = 'Y' AND
  - sec.FootballFederationEmployeeAuthConstraint(self.id)='Y'
  - THEN
  - INSERT INTO Game (gameInfo, gameReport, confirmation)
  - VALUES (NEW.gameInfo, :NEW.gameReport, :NEW.confirmation);
  - ELSE
  - RAISE ex_denied;
  - END IF;
  - EXCEPTION
  - WHEN ex_denied THEN
  - raise_application_error (-20000, 'Access denied!');
  - END;
Model Driven Security
Applying Authorisation Constraints

Outline

• Principles of role-based access control
• RBAC implementation requirements
• RBAC modelling languages
  – SecureUML
  – UMLsec
  – Language comparison
  – Transformation
• Model-driven security
  – Model-driven development
  – Security model transformation

Further reading
Further reading

Access Control Approaches

- **ABAC**: Attribute-based access control
  [Hu et al., 2014, 2015]

- **UCON**: Usage control model
  [Park and Sandhu, 2004]

- **RAdAC**: Risk-adaptive access control
  [McGraw, 2009; Shaikh et al., 2012]

- **TBAC**: Token-based access control
  [Radhakrishnan, 2012]

Further reading

Model-driven security

- Framework for RBAC modelling using XACML architecture
  [Xin, 2006]

- UML for access control features to support policy validation using OCL
  [Ahn and Hu, 2007]

- UML Profile for RBAC to integrate access control specifications with the development process
  [Cirit and Buzluca, 2009]

- SecureUML is applied to define RBAC policy on XML documents to dynamically define document structure and security policy
  [Tark and Matulevicius, 2014]

- A method to recover the RBAC security model from structural and behavioural models of Web applications
  [Alalfi et al., 2012]

- Access control policies are captured from the Spring Framework applications to facilitate needed access changes
  [Sergeev, 2016]
Outline

• Principles of role-based access control
• RBAC implementation requirements
• RBAC modelling languages
  – SecureUML
  – UMLsec
  – Language comparison
  – Transformation
• Model-driven security
  – Model-driven development
  – Security model transformation
• Further reading