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
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 Risk Management
Domain Model

Risk treatment
- Cost
- Risk reduction

Risk
- Risk level

Security criterion
- 1..*

Security objective
- Security need

Security requirement
- Cost
- Risk reduction

Event
- Potentiality

Impact
- Impact level

Asset

IS asset

Business asset

Threat
- Likelihood

Vulnerability
- Vulnerability level

Threat agent

Attack method

© Springer International Publishing AG 2017
RBAC: Role-based Access Control

**Access** – a specific type of interaction between a subject and an object that results 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}

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

© Springer International Publishing AG 2017
User - any person who interacts directly with a computer system

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

© Springer International Publishing AG 2017
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

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

© Springer International Publishing AG 2017
**Session** – a mapping between a user and an activated subset of roles the user is assigned to.
**RBAC**

**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
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

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

© Springer International Publishing AG 2017
**User** - any person who interacts directly with a computer system

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

**Object** – a passive entity that contains or receives information

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

**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 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\(_2\)

---

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

© Springer International Publishing AG 2017
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
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

- Early requirements
- Late requirements
- Architectural design
- Detailed design
- Implementation and testing

- Secure TROPOS
- KAOS extension to security
- Misuse cases
- Mal-activity diagrams
- SecureUML
- UMLsec

© Springer International Publishing AG 2017
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
Authorisation Constraints

AC#1:

```plaintext
class Game::createGame () : void
    pre: self.responsibleFFE.assignedUser -> exists(i | i.assignedUser = 'Bob')
```

AC#2:

```plaintext
class Game::updateConfirmation () : void
    pre: self.responsibleFFE.assignedUser -> exists(i | i.assignedUser = 'Bob')
```
Toy Example

Exercise
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,</td>
<td>after start eventually reach stop</td>
<td>Enforce fair exchange</td>
</tr>
<tr>
<td></td>
<td></td>
<td>adversary</td>
<td></td>
<td></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,</td>
<td>Provides secrecy, integrity, authenticity,</td>
<td>Basic data security constraints</td>
</tr>
<tr>
<td></td>
<td></td>
<td>integrity,</td>
<td>freshness</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>authenticity</td>
<td></td>
<td></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>

© Springer International Publishing AG 2017
### UMLsec

#### «rbac»

**Activity Example of UMLsec model**

<table>
<thead>
<tr>
<th>FootballFederationEmployee</th>
<th>Game</th>
<th>Umpire</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submit game info</td>
<td>Create game</td>
<td>Submit game report</td>
</tr>
<tr>
<td>Submit confirmation</td>
<td>Update game report</td>
<td>Update confirmation</td>
</tr>
</tbody>
</table>

© Springer International Publishing AG 2017
<table>
<thead>
<tr>
<th>FootballFederationEmployee</th>
<th>Game</th>
<th>Umpire</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- `{protected} = protected_action`
- `{role} = (actor, role)`
- `{right} = (role, protected_action)`
```java
{protected = Create game}
{role = (Bob, FootballFederationEmployee)}
{right = (FootballFederationEmployee, Create game)}
```
Toy Example

Exercise - 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
# Language comparison

## Extension Mechanism

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

### 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>\begin{itemize}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\begin{itemize}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\begin{itemize}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\begin{itemize}</td>
</tr>
<tr>
<td>Permission assignment</td>
<td>Association class stereotype «secuml.permission»</td>
<td>\begin{itemize}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\begin{itemize}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\begin{itemize}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\begin{itemize}</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>\begin{itemize}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\begin{itemize}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\begin{itemize}</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 `{right}`

- The value of `right` can be determined from the authorisation constraint defined for the attribute of the SecureUML association class.

\[
\{\text{right} = (\text{FootballFederationEmployee, createGame})\}\]

\[
\{\text{right} = (\text{Umpire, updateGameReport})\}\]
SU.5: Received activity diagram is annotated with the $<<\text{rbac}>>$ stereotype
protected = (createGame)  
right = (FootballFederationEmployee, createGame)
protected = (updateGameReport)  
right = (Umpire, updateGameReport)
protected = (updateConfirmation)  

role = (Bob, FootballFederationEmployee)
role = (Karl, Umpire)
role = (John, Umpire)
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

<table>
<thead>
<tr>
<th>FootballFederationEmployee</th>
<th>Game</th>
<th>Umpire</th>
</tr>
</thead>
<tbody>
<tr>
<td>submitGameInfo</td>
<td>createGame</td>
<td>submitGameReport</td>
</tr>
<tr>
<td>submitConfirmation</td>
<td>updateGameReport</td>
<td></td>
</tr>
<tr>
<td></td>
<td>updateConfirmation</td>
<td></td>
</tr>
</tbody>
</table>

- **protected** = `(createGame))`
- **right** = `(FootballFederationEmployee, createGame))`
- **role** = `(Bob, FootballFederationEmployee))`

- **protected** = `(updateGameReport))`
- **right** = `(Umpire, updateGameReport))`
- **role** = `(John, Umpire))`

- **protected** = `(updateConfirmation))`
- **right** = `(FootballFederationEmployee, updateConfirmation))`
- **role** = `(Bob, FootballFederationEmployee))`

- **protected** = `(updateGameReport))`
- **right** = `(Umpire, updateGameReport))`
- **role** = `(Karl, Umpire))`
Demo
Secure UML model

UMLsec model
UMLsec model

**Example of UMLsec model**

- **Football Federation Employee**
  - Submit game info
  - Submit confirmation

- **Game**
  - Create game
  - Update game report
  - Update confirmation

- **Umpire**
  - Submit game report
### UMLsec model

#### Example of UMLsec model

<table>
<thead>
<tr>
<th>Role</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>FootballFederationEmployee</td>
<td>Create game</td>
</tr>
<tr>
<td></td>
<td>(Bob, FootballFederationEmployee)</td>
</tr>
<tr>
<td></td>
<td>(FootballFederationEmployee, Create game)</td>
</tr>
<tr>
<td></td>
<td>Update game report</td>
</tr>
<tr>
<td></td>
<td>(John, Umpire)</td>
</tr>
<tr>
<td></td>
<td>(Umpire, Update game report)</td>
</tr>
<tr>
<td></td>
<td>Update game report</td>
</tr>
<tr>
<td></td>
<td>(Karl, Umpire)</td>
</tr>
<tr>
<td></td>
<td>(Umpire, Update game report)</td>
</tr>
<tr>
<td></td>
<td>Update confirmation</td>
</tr>
<tr>
<td></td>
<td>(Bob, FootballFederationEmployee)</td>
</tr>
<tr>
<td></td>
<td>(FootballFederationEmployee, Update confirmation)</td>
</tr>
</tbody>
</table>
US1. Association tags \{\textbf{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 «\textbf{secuml.resource}». 
US2. The UMLsec activity partitions which do not hold secured protected actions can be transformed to «`secuml.role`» stereotyped classes.
US3. Association tag \{role\} allows identifying the «assignment» dependency relationship between classed with a stereotype «secuml.user» and their «secuml.role» stereotypes.
US4. From UMLsec association tag \{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 «\texttt{umlsec.role}» and a «\texttt{umlsec.resource}» is introduced.
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
SecureUML model

Finish the transformation manually

• Attributes of the «umlsec.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
Demo
Secure UML model

UMLsec 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
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 transformation rules
**Insert security constraint**

```sql
-- 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;
```

© Springer International Publishing AG 2017
Insert security constraint

- **Resource** that needs to be secure
  - `gameInfo`
  - `gameReport`
  - `confirmation`

```sql
-- 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;
```
**Insert security constraint**

```sql
-- 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;
```

- **Security action**
  - **Insert**
Insert security constraint

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
Insert security constraint

```
-- 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 (gamInfo, gameReport, confirmation)
    VALUES (:NEW.gamInfo, :NEW.gameReport, :NEW.confirmation);
  ELSE
    RAISE ex_denied;
  END IF;
EXCEPTION
  WHEN ex_denied THEN
    raise_application_error (-20000, 'Access denied!');
END;
```

Insert values
- gamInfo
- gameReport
- confirmation
Model Driven Security

Applying Authorisation Constraints

UML (modelling languages)

MagicDraw (modelling tool)

SecureUML (modelling languages)

System/Software model

Transformation rules

Software Code

Security Model

Security Transformation rules

Security constraints (in PL/SQL)

© Springer International Publishing AG 2017
Model Driven Security

Applying Authorisation Constraints

Database management systems (e.g., SQL*Plus)

UML (modelling languages)

MagicDraw (modelling tool)

SecureUML (modelling languages)

System/Software model

Security constraints (in PL/SQL)

Software Code
Demo
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
Demo
Scenario

- Increasing number of documents
- Need to secure the document content
- Display part of document to different stakeholders

Patient's Ambulatory Treatment Record (First visit)

<table>
<thead>
<tr>
<th>Patient</th>
<th>Name:</th>
<th>Aleksandr Skafandr</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Legal code:</td>
<td>38213123111321</td>
</tr>
<tr>
<td></td>
<td>Age:</td>
<td>30 years</td>
</tr>
</tbody>
</table>

- Complaints: head ache in forehead during last 2 days. Striking pain to left arm and elbow. Slept 5-6 hours for the last three weeks period
- Primary diagnosis: R.53.83 - Exhaustion, exhaustive (physical NEC)
- Opinion: Blood pressure high and slept too little. Should go on a regular 8 hour sleeping mode. 1 week in home regime.
- Observations: 1 x massage 45min; 1 x term 30min

Visit nr V2013-12-00014000

Doctor's signature and date
Scenario

- Increasing number of documents
- Need to secure the document content
- Display part of document to different stakeholders
{role = (Eve, Receptionist)}
{protected = (addPatientData)}
{right = (Receptionist, addPatientData)}

{role = (Ann, Nurse)}
{protected = (addComplainsObservations)}
{right = (Nurse, addComplainsObservations)}

{role = (John, Doctor)}
{protected = (displayRecord)}
{right = (Doctor, displayRecord)}
{protected = (addDiagnosis)}
{right = (Doctor, addDiagnosis)}
### Dynamic forms

- **Role:** Nurse

**Documents:** Refresh

<table>
<thead>
<tr>
<th>document name</th>
<th>version</th>
<th>XML</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient diary record</td>
<td>20</td>
<td>206</td>
<td>Edit</td>
</tr>
<tr>
<td>Patient diary record</td>
<td>2</td>
<td>209</td>
<td>Edit</td>
</tr>
<tr>
<td>Patient diary record</td>
<td>4</td>
<td>210</td>
<td>Edit</td>
</tr>
<tr>
<td>Patient diary record</td>
<td>2</td>
<td>211</td>
<td>Edit</td>
</tr>
<tr>
<td>Patient diary record</td>
<td>1</td>
<td>212</td>
<td>Edit</td>
</tr>
<tr>
<td>Patient diary record</td>
<td>1</td>
<td>213</td>
<td>Edit</td>
</tr>
<tr>
<td>Patient diary record</td>
<td>6</td>
<td>214</td>
<td>Edit</td>
</tr>
<tr>
<td>Patient diary record</td>
<td>6</td>
<td>215</td>
<td>Edit</td>
</tr>
<tr>
<td>Patient diary record</td>
<td>6</td>
<td>216</td>
<td>Edit</td>
</tr>
</tbody>
</table>

#### Add new Form

<table>
<thead>
<tr>
<th>document name</th>
<th>xsdid</th>
<th>Description</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient diary record</td>
<td>101</td>
<td>Diary record</td>
<td>Add new</td>
</tr>
<tr>
<td>ExampleApplication</td>
<td>200</td>
<td>Example1 - Application</td>
<td>Add new</td>
</tr>
<tr>
<td>Group1</td>
<td>201</td>
<td>Group1 form - Application</td>
<td>Add new</td>
</tr>
<tr>
<td>Group2</td>
<td>202</td>
<td>Group2 form - Application</td>
<td>Add new</td>
</tr>
<tr>
<td>Group3</td>
<td>203</td>
<td>Group3 form - Purchase order</td>
<td>Add new</td>
</tr>
<tr>
<td>Group4</td>
<td>204</td>
<td>Group4 form - Purchase order</td>
<td>Add new</td>
</tr>
<tr>
<td>Group5</td>
<td>205</td>
<td>Group5 form - Application</td>
<td>Add new</td>
</tr>
<tr>
<td>Group6</td>
<td>206</td>
<td>Group6 form - Purchase order</td>
<td>Add new</td>
</tr>
<tr>
<td>Group7</td>
<td>207</td>
<td>Group7 form - Purchase order</td>
<td>Add new</td>
</tr>
<tr>
<td>Group8</td>
<td>208</td>
<td>Group8 form - Application</td>
<td>Add new</td>
</tr>
<tr>
<td>Group9</td>
<td>209</td>
<td>Group9 form - Application</td>
<td>Add new</td>
</tr>
<tr>
<td>ExamplePurchaseOrder</td>
<td>300</td>
<td>Example2 - Purchase Order</td>
<td>Add new</td>
</tr>
</tbody>
</table>
Model definition

• What are objects?
  – … and their attributes
  – What are operations that changes values of the attributes?

• What are roles?

• What are security actions?

• What are users?
What are **objects**?
- ... and their **attributes**
  - Cave and Food stored in it
- What are **operations** that changes values of the attributes?

What are **roles**?

What are **security actions**?

Who are **users**?
Model definition: (toy)

Example

• What are **objects**?
  – … and their **attributes**
    • Cave and Food stored in it
  – What are **operations** that changes values of the attributes?
    • Store food, Remove food, Give foods image, Change food

• What are **roles**?

• What are **security actions**?

• Who are **users**?
What are **objects**?
- ... and their **attributes**
  - Cave and Food stored in it

- What are **operations** that changes values of the attributes?
  - Store food, Remove food, Give foods image, Change food

What are **roles**?
- Cavemen, Good friend

What are **security actions**?

Who are **users**?
• What are objects?  
  – … and their attributes  
    • Cave and Food stored in it

  – What are operations that changes values of the attributes?  
    • Store food, Remove food, Give foods image, Change food

• What are roles?  
  – Cavemen, Good friend

• What are security actions?  
  – Add, Remove, View, Change

• Who are users?
Model definition: (toy)

Example

• What are **objects**?
  – … and their **attributes**
    • Cave and Food stored in it
    – What are **operations** that changes values of the attributes?
      • Store food, Remove food, Give foods image, Change food

• What are **roles**?
  – Cavemen, Good friend

• What are **security actions**?
  – Add, Remove, View, Change

• Who are **users**?
Model definition: (toy)

Example