Lecture 7:  
Part 2:  

Role-based Access Control

**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

Lecture outline

- RBAC principles
- Modelling languages for RBAC
  - SecureUML
  - UMLsec
Lecture outline

- **RBAC principles**
  
- Modelling languages for RBAC
  - SecureUML
  - UMLsec

Role-based Access Control
**RBAC_0**

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

Sandhu and Coyne, 1996; Ferraiolo et al., 2001
**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\(_2\)
Functionality of RBAC Implementation

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

Functionality of RBAC Implementation

- 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
Functionality of RBAC Implementation

- When User Assignment and Permission Assignment are defined
  - 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

Functionality of RBAC Implementation

- When User Assignment and Permission Assignment are defined
  - 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
Lecture outline

- RBAC principles
- **Modelling languages for RBAC**
  - SecureUML
  - UMLsec

Security Modelling Languages

<table>
<thead>
<tr>
<th>Early requirements</th>
<th>Late requirements</th>
<th>Architectural design</th>
<th>Detailed design</th>
<th>Implementation and testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure TROPOS</td>
<td>KAOS extension to security</td>
<td>Misuse cases</td>
<td>Mal-activity diagrams</td>
<td>SecureUML</td>
</tr>
</tbody>
</table>
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:
context MeetingAgreement::setTimePlace(): void
   pre: self.responsibleFFEmployee.assignedUser -> exists(i | i.assignedUser = 'Bob')

AC#2:
context MeetingAgreement::changeTimePlace(): void
   pre: self.responsibleFFEmployee.assignedUser -> exists(i | i.assignedUser = 'Bob')

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**?

---

**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**?
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

What are **users**?
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
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>
• \{\textit{protected} = \textit{protected\_action}\}
• \{\textit{role} = (\textit{actor}, \textit{role})\}
• \{\textit{right} = (\textit{role}, \textit{protected\_action})\}
Message to Take Home

• RBAC principles
• Modelling languages for RBAC
  – SecureUML
  – UMLsec