Access Control

Discretionary Access Control
Lecture 4
Introduction

“Access control” is where security engineering meets computer science.

Its function is to control which (active) subject have access to a which (passive) object with some specific access operation.
Access Control

Discretionary Access Control (DAC)

- Access Matrix Model
- Implementation of the Access Matrix
- Vulnerabilities of the Discretionary Policies
- Additional features of DAC
Discretionary Access Control

- Access to data objects (files, directories, etc.) is permitted based on the identity of users.

- Explicit access rules that establish who can, or cannot, execute which actions on which resources.

- Discretionary: users can be given the ability of passing on their privileges to other users, where granting and revocation of privileges is regulated by an administrative policy.
**Discretionary Access Control**

- DAC is flexible in terms of policy specification
- This is the form of access control widely implemented in standard multi-user platforms Unix, NT, Novell, etc.
Discretionary Access Control

Access control matrix
- Describes protection state precisely
- Matrix describing rights of subjects
- State transitions change elements of matrix

State of protection system
- Describes current settings, values of system relevant to protection
Access Control

Discretionary Access Control
  – Access Control Matrix Model
  – Implementation of the Access Matrix
  – Vulnerabilities of the Discretionary Policies
  – Additional features of DAC
Access Control Matrix Model

Access control matrix

– Firstly identify the objects, subjects and actions.
– Describes the protection state of a system.
– State of the system is defined by a triple (S, O, A)
  • S is the set of subject,
  • O is the set of objects,
  • A is the access matrix
– Elements indicate the access rights that subjects have on objects
  • Entry $A[s, o]$ of access control matrix is the privilege of $s$ on $o$
Description

objects (entities)

<table>
<thead>
<tr>
<th></th>
<th>$O_1$</th>
<th>$\ldots$</th>
<th>$O_m$</th>
<th>$S_1$</th>
<th>$\ldots$</th>
<th>$S_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\ldots$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S_n$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Subjects** $S = \{ s_1, \ldots, s_n \}$
- **Objects** $O = \{ o_1, \ldots, o_m \}$
- **Rights** $R = \{ r_1, \ldots, r_k \}$
- **Entries** $A[s_i, o_j] \subseteq R$

$A[s_i, o_j] = \{ r_{x_1}, \ldots, r_{y_i} \}$

means subject $s_i$ has rights $r_{x_1}, \ldots, r_{y_i}$ over object $o_j$
Boolean Expression Evaluation

ACM controls access to database fields
- Subjects have attributes
- Action/Operation/Verb define type of access
- Rules associated with objects, action pair

Subject attempts to access object
- Rule for object, action evaluated, grants or denies access
Example

Subject Annie
  – Attributes role (artist), groups (creative)

Verb paint
  – Default 0 (deny unless explicitly granted)

Object picture
  – Rule:
    Annie paint picture if:
      ‘artist’ in subject.role and
      ‘creative’ in subject.groups and
      time.hour ≥ 0 and time.hour < 5
ACM at 3AM and 10AM

At 3AM, time condition met; ACM is:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>annie</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>paint</td>
<td></td>
</tr>
</tbody>
</table>

... picture ...

At 10AM, time condition not met; ACM is:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>annie</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... picture ...
Access Controlled by History

Statistical databases need to
- answer queries on groups
- prevent revelation of individual records

Query-set-overlap control
- Prevent an attacker to obtain individual piece of information using a set of queries C
- A parameter $r (=2)$ is used to determine if a query should be answered

<table>
<thead>
<tr>
<th>Name</th>
<th>Position</th>
<th>Age</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Teacher</td>
<td>45</td>
<td>40K</td>
</tr>
<tr>
<td>Bob</td>
<td>Aide</td>
<td>20</td>
<td>20K</td>
</tr>
<tr>
<td>Cathy</td>
<td>Principal</td>
<td>37</td>
<td>60K</td>
</tr>
<tr>
<td>Dilbert</td>
<td>Teacher</td>
<td>50</td>
<td>50K</td>
</tr>
<tr>
<td>Eve</td>
<td>Teacher</td>
<td>33</td>
<td>50K</td>
</tr>
</tbody>
</table>
Access Controlled by History

Query 1:
- `sum_salary(position = teacher)`
- Answer: 140K

Query 2:
- `sum_salary(age > 40 & position = teacher)`
- Should not be answered as Matt’s salary can be deduced

Can be represented as an ACM
Solution: Query Set Overlap Control
(Dobkin, Jones & Lipton ’79)

Query valid if intersection of query coverage and each previous query < r

Can represent as access control matrix

– Subjects: entities issuing queries
– Objects: Powerset of records
– $O_s(i)$: objects referenced by s in queries $1..i$
– $M[s,o] = \begin{cases} \text{read} & \text{iff } \forall q \in O_s(i-1) \left| q \cap o \right| < r \end{cases}$
\[
\text{M}[s,o] = \text{read iff } \forall q \in O_{s(i-1)} \left| q \cap o \right| < r
\]

**Query 1:** \( O_1 = \{\text{Celia, Leonard, Matt}\} \) so the query can be answered. Hence
- \( \text{M[asker, Celia]} = \{\text{read}\} \)
- \( \text{M[asker, Leonard]} = \{\text{read}\} \)
- \( \text{M[asker, Matt]} = \{\text{read}\} \)

**Query 2:** \( O_2 = \{\text{Celia, Leonard}\} \) but \( |O_2 \cap O_1| = 2 \); so the query cannot be answered
- \( \text{M[asker, Celia]} = \emptyset \)
- \( \text{M[asker, Leonard]} = \emptyset \)
Access Control

Discretionary Access Control

- Access Matrix Model
- Implementation of the Access Control Matrix
- Vulnerabilities of the Discretionary Policies
- Additional features of DAC
ACM Implementation

- ACM is an **abstract** model  
  - Rights may vary depending on the object involved

- ACM is implemented primarily in three ways  
  - Authorization Table  
  - Capabilities (rows)  
  - Access control lists (columns)
Authorization Table

- Three columns: subjects, actions, objects
- Generally used in DBMS systems

<table>
<thead>
<tr>
<th>User</th>
<th>Access Mode</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ann</td>
<td>own</td>
<td>File 1</td>
</tr>
<tr>
<td>Ann</td>
<td>read</td>
<td>File 1</td>
</tr>
<tr>
<td>Ann</td>
<td>write</td>
<td>File 1</td>
</tr>
<tr>
<td>Ann</td>
<td>read</td>
<td>File 2</td>
</tr>
<tr>
<td>Ann</td>
<td>write</td>
<td>File 2</td>
</tr>
<tr>
<td>Ann</td>
<td>execute</td>
<td>Program 1</td>
</tr>
<tr>
<td>Bob</td>
<td>read</td>
<td>File 1</td>
</tr>
<tr>
<td>Bob</td>
<td>read</td>
<td>File 3</td>
</tr>
<tr>
<td>Bob</td>
<td>write</td>
<td>File 3</td>
</tr>
<tr>
<td>Carl</td>
<td>read</td>
<td>File 2</td>
</tr>
<tr>
<td>Carl</td>
<td>execute</td>
<td>Program 1</td>
</tr>
<tr>
<td>Carl</td>
<td>read</td>
<td>Program 1</td>
</tr>
</tbody>
</table>
Access Control List (ACL)

- Matrix is stored by column.
- Each object is associated with a list
- Indicate for each subject the actions that the subject can exercise on the object
**Capability List**

- Matrix is stored by row
- Each user is associated with a capability list
- Indicating for each object the access that the user is allowed to exercise on the object
**ACLs vs Capability List**

- Immediate to check the authorization holding on an object with ACLs. (subject?)
- Immediate to determine the privileges of a subject with Capability lists. (object?)

- Distributed system,
  - authenticate once, access various servers
  - choose which one?

- Limited number of groups of users, small bit vectors, authorization specified by owner.
  - Which one?
Basic Operations in Access Control

- **Grant permissions**
  - Inserting values in the matrix’s entries

- **Revoke permissions**
  - Remove values from the matrix’s entries

- **Check permissions**
  - Verifying whether the entry related to a subject $s$ and an object $o$ contains a given access mode
Access Control

Discretionary Access Control

– Access Matrix Model
– State of Protection System
– Implementation of the Access Matrix
– Vulnerabilities of the Discretionary Policies
– Additional features of DAC
Vulnerabilities of the Discretionary Policies

- No separation of users from subjects
- No control on the flow the information
- Malicious code, i.e., Trojan horse
Example

- **Vicky**, a top-level manager
- **A file Market** on the new products release
- **John**, subordinate of Vicky
- **A file called “Stolen”**
- **An application** with two hidden operations
  - Read operation on file Market
  - Write operation on file Stolen
Example (cond)

File Market
Aug. 00; product X; price 7,000
Dec. 00; product Y; price 3,500
Jan. 01; product Z; price 1,200

owner Vicky

File Stolen

owner John
\langle Vicky,\text{write},\text{Stolen} \rangle
Example (cond)

- Restriction should be enforced on the operations that processes themselves can execute.
- Mandatory policies provide a way to enforce information flow control through the use of labels.
Access Control

Discretionary Access Control
- Access Matrix Model
- State of Protection System
- Implementation of the Access Matrix
- Vulnerabilities of the Discretionary Policies
- Additional features of DAC
DAC – additional features and recent trends

Flexibility is enhanced by supporting different kinds of permissions

– Positive vs. negative
– Strong vs. weak
– Implicit vs. explicit
– Content-based
Positive and Negative Permissions

Positive permissions → Give access
Negative permissions → Deny access
Useful to specify exceptions to a given policy and to enforce stricter control on particular crucial data items
Positive and Negative Permissions

Main Issue: Conflicts
Authorization Conflicts

Main solutions:
- No conflicts
- Negative permissions take precedence
- Positive permissions take precedence
- Nothing take precedence
- Most specific permissions take precedence
Weak and Strong Permissions

Strong permissions cannot be overwritten

Weak permissions can be overwritten by strong and weak permissions
Implicit and Explicit Permissions

Some models support implicit permissions

Implicit permissions can be derived:

– by a set of *propagation rules* exploiting the subject, object, and privilege hierarchies

– by a set of user-defined *derivation rules*
Derivation Rules: Example

- Ann can read file F1 from a table if Bob has an explicit denial for this access.
- Tom has on file F2 all the permissions that Bob has.
- Derivation rules are a way to concisely express a set of security requirements.
Derivation Rules

Derivation rules are often expressed according to logic programming.

Several research efforts have been carried out to compare the expressive power of such languages.

We need languages based on SQL and/or XML.
Content-based Permissions

Content-based access control conditions the access to a given object based on its content.

This type of permissions are mainly relevant for database systems.

As an example, in a RDBMS supporting content-based access control it is possible to authorize a subject to access information only of those employees whose salary is not greater than 30K.
Content-based Permissions

Two most common approaches to enforce content-based access control in a DBMS are done:

– by associating a predicate (or a Boolean combination of predicates) with the permission

– by defining a view which selects the objects whose content satisfies a given condition, and then granting the permission on the view instead of on the basic objects
DAC models - DBMS vs OS

- Increased number of objects to be protected
- Different granularity levels (relations, tuples, single attributes)
- Protection of logical structures (relations, views) instead of real resources (files)
- Different architectural levels with different protection requirements
- Relevance not only of data physical representation, but also of their semantics
Access Control -- RBAC

Lecture 4
Many organizations base access control decisions on “the roles that individual users take on as part of the organization”.

They prefer to centrally control and maintain access rights that reflect the organization’s protection guidelines.

With RBAC, role-permission relationships can be predefined, which makes it simple to assign users to the predefined roles.

The combination of users and permissions tend to change over time, the permissions associated with a role are more stable.

RBAC concept supports three well-known security principles:

– Least privilege
– Separation of duties
– Data abstraction
Role Based Access Control (RBAC)

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”
RBAC

Access depends on role/function, not identity

- Example: Allison is bookkeeper for Math Dept. She has access to financial records. If she leaves and Betty is hired as the new bookkeeper, Betty now has access to those records. The role of “bookkeeper” dictates access, not the identity of the individual.
Advantages of RBAC

- Allows Efficient Security Management
  - Administrative roles, Role hierarchy
- Principle of least privilege allows minimizing damage
- Separation of Duties constraints to prevent fraud
- Allows grouping of objects
- Policy-neutral - Provides generality
- Encompasses DAC and MAC policies
RBAC

User permissions and roles

- Manager
- Senior Administrator
- Senior Engineer
- Administrator
- Engineer
- Employee

Users and Permissions

- $u_1$, $u_2$, ..., $u_n$
- $o_1$, $o_2$, ..., $o_m$

Assignments

- $n + m$ assignments
- $n \times m$ assignments

(a) vs (b)
RBAC (cont’d)

- Is RBAC a discretionary or mandatory access control?
  - RBAC is policy neutral; however individual RBAC configurations can support a mandatory policy, while others can support a discretionary policy.

- Role Hierarchies
- Role Administration

```
Project Supervisor
  /      \
/        \      /
Test engineer  Programmer
  |      |    |
  |      |    |
  |      |    |
  |      |    |
  |      |    |
Project Member
```
An important difference from classical models is that **Subject** in other models corresponds to a **Session** in RBAC.
Core RBAC (relations)

- Permissions = \(2^{\text{Operations} \times \text{Objects}}\)
- \(\text{UA} \subseteq \text{Users} \times \text{Roles}\)
- \(\text{PA} \subseteq \text{Permissions} \times \text{Roles}\)
- \(\text{assigned}_\text{users}: \text{Roles} \rightarrow 2^{\text{Users}}\)
- \(\text{assigned}_\text{permissions}: \text{Roles} \rightarrow 2^{\text{Permissions}}\)
- \(\text{Op}(p):\) set of operations associated with permission \(p\)
- \(\text{Ob}(p):\) set of objects associated with permission \(p\)
- \(\text{user}_\text{sessions}: \text{Users} \rightarrow 2^{\text{Sessions}}\)
- \(\text{session}_\text{user}: \text{Sessions} \rightarrow \text{Users}\)
- \(\text{session}_\text{roles}: \text{Sessions} \rightarrow 2^{\text{Roles}}\)
  - \(\text{session}_\text{roles}(s) = \{r \mid (\text{session}_\text{user}(s), r) \in \text{UA}\}\)
- \(\text{avail}_\text{session}_\text{perms}: \text{Sessions} \rightarrow 2^{\text{Permissions}}\)
RBAC with General Role Hierarchy

Users

Roles

Operations

Objects

user_sessions (one-to-many)

role_sessions (many-to-many)

RAH (role hierarchy)

UA

PA

Permissions
**RBAC with General Role Hierarchy**

- **authorized_users**: Roles $\rightarrow 2^{\text{Users}}$
  
  $\text{authorized_users}(r) = \{ u \mid r' \geq r \&(r', u) \in UA\}$

- **authorized_permissions**: Roles $\rightarrow 2^{\text{Permissions}}$
  
  $\text{authorized_permissions}(r) = \{ p \mid r' \geq r \&(p, r') \in PA\}$

- RH $\subseteq$ Roles x Roles is a partial order
  
  - called the inheritance relation
  
  - written as $\geq$.

  $(r_1 \geq r_2) \rightarrow \text{authorized_users}(r_1) \subseteq \text{authorized_users}(r_2) \& \text{authorized_permissions}(r_2) \subseteq \text{authorized_permissions}(r_1)$
Example

authorized_users(Employee)?
authorized_users(Administrator)?

authorized_permissions(Employee)?
authorized_permissions(Administrator)?
Constrained RBAC

Static Separation of Duty

Role Hierarchy (RH)

Dynamic Separation of Duty

User Sessions (one-to-many)

Permissions

Operations

Objects
Separation of Duties

- No user should be given enough privileges to misuse the system on their own.
- Statically: defining the conflicting roles
- Dynamically: Enforcing the control at access time
RBAC’s Benefits

<table>
<thead>
<tr>
<th>Task</th>
<th>RBAC</th>
<th>Non-RBAC</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign existing privileges to new users</td>
<td>6.14</td>
<td>11.39</td>
<td>5.25</td>
</tr>
<tr>
<td>Change existing users’ privileges</td>
<td>9.29</td>
<td>10.24</td>
<td>0.95</td>
</tr>
<tr>
<td>Establish new privileges for existing users</td>
<td>8.86</td>
<td>9.26</td>
<td>0.40</td>
</tr>
<tr>
<td>Termination of privileges</td>
<td>0.81</td>
<td>1.32</td>
<td>0.51</td>
</tr>
</tbody>
</table>
Cost Benefits

Saves about 7.01 minutes per employee, per year in administrative functions
- Average IT admin salary - $59.27 per hour
- The annual cost saving is:
  - $6,924/1000; $692,471/100,000

Reduced Employee downtime
- If new transitioning employees receive their system privileges faster, their productivity is increased
- 26.4 hours for non-RBAC; 14.7 hours for RBAC
- For average employee wage of $39.29/hour, the annual productivity cost savings yielded by an RBAC system:
  - $75000/1000; $7.4M/100,000