Access Control

Data Security and Privacy @ uninsubria

Roberto Vicario

2024/2025

Contents

1 Introduction **2**

2 Mandatory Access Control (MAC) **2**
2.1 Bell and La Padula Model 2
2.1.1 Key Components 2
2.1.2 Access Classes 2
2.1.3 Simple Security Property 3
2.1.4 Star Property 3
2.1.5 Tranquility Property 4

3 Role-Based Access Control (RBAC) **4**
3.1 Key Components 4
3.2 Role Hierarchy 4
3.2.1 Semantic Role Inheritance 5
3.3 Separation of Duties (SoD) 5
3.3.1 Types of SoD 5

4 SQL Access Control **5**
4.1 SQL Injection 6
4.1.1 SQL Injection Types 6
4.1.2 Mitigation Techniques 7

1 Introduction

…

2 Mandatory Access Control (MAC)

Mandatory Access Control (MAC) is a security model that enforces access control policies based on the classification of information and the user's security clearance. In MAC, access decisions are made by the system based on predefined security labels, and users cannot change these labels.

2.1 Bell and La Padula Model

The *Bell and La Padula model* is a formal model for mandatory access control that focuses on maintaining the confidentiality of information. It uses security labels to classify data and users to ensure that sensitive information is accessed only by authorized users.

2.1.1 Key Components

Component	Description
Subjects	Active entities (e.g., users, processes) that access objects.
Objects	Passive entities (e.g., files, databases) that are accessed by subjects.
Security Levels	Hierarchical levels of security (e.g., Top Secret, Secret, Confidential) that determine access rights.

2.1.2 Access Classes

Access class $c_i = (L_i, SC_i)$ dominates access class $c_k = (L_k, SC_k)$, denoted as $c_i > c_k$, if both the following conditions hold:

1. $L_i \geq L_k$: The security level of c_i is greater than or equal to the security level of c_k.
2. $SC_i \supseteq SC_k$: The category set of c_i includes the category set of c_k.

Access classes are partially ordered. If $L_i > L_k$ and $SC_i \supset SC_k$, we say that c_i strictly dominates c_k.

Access classes c_i and c_k are said to be incomparable, denoted as $c_i \bowtie c_k$, if neither $c_i > c_k$ nor $c_k > c_i$ holds.

2.1.3 Simple Security Property

The *simple security property* ensures that access to objects is controlled based on the security levels of subjects and objects. A given state (A, L) satisfies the simple security property if, for each element $(s, o, m) \in A$, one of the following conditions holds:

1. $m =$ append
2. $m =$ read or $m =$ write and $L(s) > L(o)$

Here: - s: Subject attempting the operation. - o: Object being accessed. - m: Mode of access (e.g., read, write, append). - $L(s)$: Security level of the subject. - $L(o)$: Security level of the object.

This property ensures that subjects can only access objects in ways that do not violate the confidentiality constraints imposed by their security levels.

2.1.4 Star Property

The _star (*) property_ ensures that information flows up, not down, in a mandatory access control system. This property enforces that “writes up” are allowed, while “writes down” are disallowed, thereby maintaining confidentiality.

A given state (A, L) satisfies the star property if, for each element $(s, o, m) \in A$, one of the following conditions holds:

1. $m =$ read
2. $m =$ append or $m =$ write and $L(o) > L(s)$

Here: - s: Subject attempting the operation. - o: Object being accessed. - m: Mode of access (e.g., read, write, append). - $L(s)$: Security level of the subject. - $L(o)$: Security level of the object.

This property ensures that subjects can only write to objects at higher security levels, preventing the leakage of sensitive information to lower security levels.

> A system state is considered **secure** if and only if it satisfies both the *simple security property* and the *star property*. Access is **granted** only when the resulting state remains secure, ensuring that the system's confidentiality constraints are upheld.

2.1.5 Tranquility Property

The *tranquility property* states that the security levels of subjects and objects do not change during the execution of a program. This property ensures that once a subject or object is assigned a security level, it remains constant throughout its lifetime.

3 Role-Based Access Control (RBAC)

Role-Based Access Control (RBAC) is a security model that assigns permissions to users based on their roles within an organization. In RBAC, users are assigned to roles, and each role has specific permissions associated with it. This model simplifies access management by grouping users into roles, making it easier to manage permissions at scale.

3.1 Key Components

Component	Description
Users	Individuals or entities that access the system.
Roles	Named groups of permissions that can be assigned to users. Roles represent job functions or responsibilities within the organization.
Permissions	Specific access rights to perform actions on resources. Permissions are associated with roles rather than individual users.
Sessions	Temporary associations between users and roles. A user can activate a role during a session to perform specific tasks.

3.2 Role Hierarchy

> **Definition:** *Role hierarchy* is a feature of RBAC that allows roles to inherit permissions from other roles. This simplifies permission management by allowing administrators to define a hierarchy of roles, where higher-level roles inherit permissions from lower-level roles.

More formally, a *role hierarchy* can be mathematically formalized as a partial order $RH \subseteq ROLES \times ROLES$ on the set of roles $ROLES$. The partial order is denoted by the dominance relation $\geq$:

- When $r_1 \geq r_2$, we say that r_1 is *senior* to r_2, and r_2 is *junior* to r_1.

3.2.1 Semantic Role Inheritance

Semantic	Description
User Inheritance	All users authorized for a role r are also authorized for any role r' where $r \geq r'$.
Permission Inheritance	A role r is authorized for all permissions for which any role r', such that $r \geq r'$, is authorized.

3.3 Separation of Duties (SoD)

> **Definition:** *Separation of Duties (SoD)* is a security principle that divides responsibilities among multiple users to prevent fraud and errors. It ensures that no single user has complete control over a critical process, reducing the risk of unauthorized actions.

3.3.1 Types of SoD

Type	Description
Static	Conflicts are resolved during system design.
Dynamic	Conflicts are resolved at runtime based on user actions.

4 SQL Access Control

SQL access control is a mechanism used to manage and restrict access to database resources based on user roles and permissions. It ensures that only authorized users can perform specific actions on

the database, such as querying, updating, or deleting data.

4.1 SQL Injection

SQL injection is a code injection technique that exploits vulnerabilities in an application's software by injecting malicious SQL code into a query. This can lead to unauthorized access to sensitive data, data manipulation, or even complete control over the database.

> **Example:** Consider a web application that allows users to log in using their username and password. The application constructs a SQL query to authenticate users:

```sql
SELECT * FROM users WHERE username = ? AND password = ?;
```

If the application does not properly validate or sanitize user input, an attacker could inject malicious SQL code into the username or password fields. For example, if the attacker enters the following input for the username:

```
' OR 1 = 1; --
```

The resulting SQL query would be:

```sql
SELECT * FROM users WHERE username = '' OR 1 = 1; --' AND password = ?;
```

This query would always return true, allowing the attacker to bypass authentication and gain unauthorized access to the application.

4.1.1 SQL Injection Types

Type	Description
Classic	The attacker injects malicious SQL code into a query, allowing them to manipulate the database. This can include bypassing authentication, retrieving sensitive data, or modifying data.
Union-based	Combines results from multiple queries using the `UNION` operator to extract data from other tables.

Type	Description
Time-based	Exploits response delays to infer database information by observing query execution time.
Blind	Exploits application behavior to infer database information without direct feedback.

4.1.2 Mitigation Techniques

Technique	Description
Parameterized Queries	Use prepared statements to ensure user input is treated as data, not code.
Input Validation	Ensure user input is sanitized and conforms to expected formats to prevent malicious code execution.
Least Privilege	Grant only the necessary privileges to users and applications. Avoid using admin accounts for routine tasks.
Web Application Firewalls (WAFs)	Use WAFs to detect and block SQL injection attempts.
Object-Relational Mapping (ORM) Frameworks	Use ORM frameworks to simplify database interactions and mitigate SQL injection risks.