40. RBAC Policy DSL Compiler with Static Security & Privilege Escalation Detection

Course Name: Compiler Design

Course Code: CS1202

Name: Pinikeshi Meghana

Roll No: 24CSB0A54

Sec: CSE-A

Document Details: Week 1 Deliverables

Week 1 – RBAC Fundamentals & Problem Definition

1. Introduction

Role-Based Access Control (RBAC) is an access control model in which permissions are assigned to roles rather than directly to individual users. Users are then assigned appropriate roles based on their responsibilities within an organization. This approach simplifies access management, improves security, and supports scalability in large systems.

RBAC is widely used in real-world applications such as cloud Identity and Access Management (IAM) systems, enterprise software, and database access control. By grouping permissions into roles, organizations can enforce consistent security policies and reduce administrative overhead.

However, as systems grow, RBAC configurations become increasingly complex due to the introduction of role hierarchies and inheritance relationships. This complexity can lead to misconfigurations that are difficult to detect manually and may result in security vulnerabilities.

The objective of this project is to analyze RBAC policies at compile time and identify logical security issues such as role conflicts, redundant permissions, and unintended privilege escalation before deployment.

2. Core RBAC Concepts

This section describes the fundamental components of the Role-Based Access Control (RBAC) model that form the basis of this project.

(i) Users

A user represents an individual or system entity that requests access to resources within a system.

Example:
Alice (System Administrator), Bob (Software Developer)

(ii) Roles

A role is a collection of permissions that represents a specific job function or responsibility in an organization. Roles simplify access control by grouping permissions logically.

Examples:
Admin, Editor, Viewer

(iii) Permissions

A permission defines an allowed operation that can be performed on a specific resource.

Examples:
read file, write database, delete record

(iv) Role Assignment

Role assignment is the process of associating users with roles. When a user is assigned a role, the user automatically gains all permissions associated with that role.

Example:
Alice → Admin
Bob → Editor

(v) Role Inheritance

Role inheritance allows a role to inherit permissions from another role, enabling hierarchical access control. A higher-level role automatically includes the permissions of the roles it inherits.

Example:
Editor inherits Viewer
Admin inherits Editor

(vi) Least Privilege Principle

The principle of least privilege states that users should be granted only the minimum set of permissions necessary to perform their assigned tasks.

Example:
A developer should not be assigned an Admin role solely to modify application code.

3. Permission Propagation in RBAC

In Role-Based Access Control systems, permissions are not limited to those directly assigned to a role. When role inheritance is used, permissions propagate transitively through the role hierarchy. As a result, users assigned to higher-level roles automatically gain the permissions of all inherited roles.

Consider the following role hierarchy:

Viewer → read
Editor → write and inherits Viewer
Admin → delete and inherits Editor

In this hierarchy, the Editor role includes both read and write permissions, while the Admin role includes read, write, and delete permissions.

The effective permissions of a user are determined by the role assigned to the user and all roles inherited by that role.

Example:

User	Assigned Role	Effective Permissions
Anu	Admin	Read,write,delete
Barbie	Editor	Read,write
Claire	Viewer	Read

While permission propagation simplifies access management, it also increases the risk of unintended permission assignment. Misconfigured inheritance relationships can result in users obtaining privileges that exceed their intended access level, making permission propagation a critical aspect to analyze in RBAC systems.

4. RBAC Misconfiguration Scenarios

Although RBAC simplifies access control, improper configuration of roles, permissions, and inheritance relationships can lead to serious security vulnerabilities. This section discusses common RBAC misconfiguration scenarios observed in real-world systems such as cloud IAM platforms and database access control mechanisms.

(i) Role Conflicts

Role conflicts occur when a user is assigned multiple roles with conflicting responsibilities or privileges. Such conflicts often violate the principle of separation of duties.

Example:
A user assigned both the Auditor role, which requires read-only access, and the Editor role, which allows data modification.

Security Impact:
This misconfiguration may allow unauthorized modification of sensitive data and compromise system integrity.

(ii) Redundant Permissions

Redundant permissions arise when the same permission is granted to a user through multiple roles. This commonly occurs in large systems with overlapping role definitions.

Example:
Two different roles independently grant the same read permission to a user.

Security Impact:
Redundant permissions increase policy complexity, make access reviews difficult, and complicate security auditing.

(iii) Privilege Escalation

Privilege escalation occurs when a user gains higher privileges than intended due to incorrect role inheritance or misconfigured role hierarchies.

Example:
A low-privilege role unintentionally inherits permissions from a higher-privilege role, granting users access beyond their intended scope.

Security Impact:
This violates the principle of least privilege and may lead to unauthorized access or system compromise.

These misconfiguration scenarios highlight the limitations of manual RBAC management and emphasize the need for automated, compile-time analysis of RBAC policies.

5. Problem Definition

Role-Based Access Control (RBAC) is widely adopted due to its simplicity and scalability. However, as the number of users, roles, permissions, and inheritance relationships increases, RBAC policies become complex and difficult to analyze manually. Logical errors such as role conflicts, redundant permissions, and unintended privilege escalation often arise from this complexity.

Existing RBAC systems primarily rely on runtime access checks, which only enforce permissions after the policies have already been deployed. These runtime mechanisms are insufficient for detecting misconfigurations in advance, allowing security vulnerabilities to remain hidden until they are exploited.

Manual inspection of RBAC configurations is error-prone and does not scale well in large systems. There is a need for a systematic and automated approach to analyze RBAC policies before deployment.

This project addresses this problem by performing static, compile-time analysis of RBAC policies to detect security vulnerabilities early. By treating RBAC policies as analyzable specifications, the project aims to identify role conflicts, redundant permissions, and privilege escalation scenarios in a precise and scalable manner.

6. Security Motivation and Project Objectives

Security Motivation :

In most access control systems, RBAC policies are enforced at runtime, meaning access checks occur only when a user attempts to perform an action. While this ensures policy enforcement, it does not prevent security issues caused by misconfigured RBAC policies from being deployed in the first place.

Misconfigurations such as role conflicts, redundant permissions, and unintended privilege escalation are logical errors that cannot be easily detected through runtime checks alone. These issues often remain unnoticed until they lead to security breaches or unauthorized access.

Performing compile-time analysis of RBAC policies enables early detection of such security flaws, reducing the attack surface and improving overall system security. This motivates the need for a compiler-based approach to RBAC policy validation.

Project Objectives :

The primary objectives of this project are:

- To statically analyze RBAC policies before deployment
- To detect role conflicts that violate separation of duties
- To identify redundant permissions within role hierarchies
- To detect direct and indirect privilege escalation scenarios
- To apply compiler analysis techniques to improve RBAC security

7. Project Scope and Assumptions

Project Scope:

The scope of this project is limited to the static analysis of Role-Based Access Control (RBAC) policies. The project focuses on analyzing the logical structure of RBAC configurations, including roles, permissions, role assignments, and role inheritance.

The analysis aims to detect security-related issues such as role conflicts, redundant permissions, and unintended privilege escalation. The project does not involve runtime access control enforcement or monitoring.

Assumptions:

The following assumptions are made in this project:

- RBAC policies are statically defined before deployment
- Roles and permissions are predefined and do not change at runtime
- Role inheritance relationships are explicitly specified
- Dynamic role creation and runtime policy modifications are not considered