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 2 Deliverables

Week 2 – Literature Survey & Gap Identification

1. Introduction

After defining the RBAC problem and security motivation in Week 1, this week focuses on studying existing standards, tools, and research approaches related to RBAC and access control policy analysis. The objective is to understand how RBAC policies are currently specified and verified, identify their limitations, and clearly justify the need for a compiler-based static analysis solution.

2. Study of Relevant Standards and Tools

The objective of this section is to study existing RBAC standards, commonly used RBAC-based IAM systems, and static security analysis techniques in order to understand how RBAC policies are currently specified, enforced, and analyzed. This study helps identify limitations in existing approaches and motivates the need for a compiler-based static analysis solution**.**

(i) NIST RBAC Standard

The NIST RBAC standard defines a formal and widely accepted model for Role-Based Access Control. It specifies core RBAC entities such as users, roles, permissions, role assignments, role hierarchies, and constraints including separation of duty. This standard forms the conceptual foundation for most enterprise and cloud-based access control systems.

Limitations:

- Focuses primarily on policy definition and runtime enforcement.
- Provides no mechanism for compile-time verification or static detection of security issues.
- Policy misconfigurations may remain undiscovered until after deployment.

(ii) Existing RBAC and IAM Systems

RBAC is widely implemented in modern IAM systems used across enterprise software, databases, and cloud platforms. These systems group permissions into roles and assign users to roles dynamically.

Limitations:

- Emphasize runtime enforcement over design-time verification.
- Offer limited support for structural analysis of RBAC policies.
- Manual inspection is often required to detect conflicts or redundant permissions, which does not scale well for large RBAC configurations.

(iii) Static Security and Policy Analysis Techniques

Static analysis techniques examine specifications without executing them and are extensively used in compiler design and software security. Applied to RBAC, these techniques can analyze role hierarchies, permission propagation, and user-role relationships at compile time.

Limitations:

- Often focus on specific security properties and rely on complex formal models.
- Lack integration into practical RBAC policy authoring or management workflows.

Observations from Standards and Tools Study:

- Existing RBAC standards emphasize specification, not verification.
- IAM systems prioritize runtime enforcement over early error detection.
- Static analysis techniques are powerful but underutilized in RBAC.

This highlights the potential value of a compiler-based static analysis approach for RBAC policy verification.

3. Survey of Existing Research Work

Research Focus Areas

The survey includes recent papers from IEEE, ACM, and Springer, published in the last 4–5 years, focusing on RBAC or access control policy analysis.

Major focus areas identified:

1. **RBAC Policy Verification:** Formal verification techniques check policy consistency, constraint satisfaction, and separation-of-duty violations.
   Limitation: Complex methods that are difficult to scale to large systems.
2. **Privilege Escalation Detection:** Graph-based models and reachability analysis detect direct and indirect permission escalation.
   Limitation: Often performed post-deployment and results lack actionable guidance.
3. **Role Conflict Analysis:** Constraint-based approaches enforce organizational rules and detect incompatible role assignments.
   Limitation: Limited interaction analysis between role conflicts and hierarchies.
4. **Static Security Analysis:** Techniques inspired by compiler analysis detect structural inconsistencies without execution.
   Limitation: Rarely implemented in full RBAC policy analysis pipelines.

4. Literature Review Table

Paper / System	Authors	Year	Technique / Approach	Key Contributions	Limitations
NIST RBAC Model	Vinay Reddy	2020	Formal RBAC specification	Defines standard RBAC components such as users, roles, permissions, role hierarchies, and constraints; forms the foundation for RBAC implementations	Focuses on policy specification and runtime enforcement; no compile-time or static security analysis
Model Checking Access Control Policies (Google Cloud IAM)	Gouglidis et al.	2023	Formal model checking	Verifies RBAC policies against security properties using model checking; detects policy inconsistencies early	Requires formal modeling expertise; not integrated into a DSL-based compiler framework
Privilege Escalation Detection using Model-Driven Engineering	Abu Zaid et al.	2022	Static analysis + Model-Driven Engineering	Detects direct and indirect privilege escalation caused by permission inheritance using static models	Focused on IoT systems; limited generalization to enterprise RBAC policies
Automated RBAC Assessment for Microservice Mesh	Das et al.	2021	Static RBAC assessment	Detects inconsistencies and misconfigurations in RBAC policies using centralized static analysis	Domain-specific to microservices; lacks DSL-level policy specification
Static Analysis for Access Control Policies	Li et al.	2024	Rule-based static security analysis	Enables early detection of access-control	Not implemented as a full

				inconsistencies using static rule evaluation	compiler pipeline; limited explainability of detected issues
Securing Academic Social Platforms: Implementing RBAC in University-Based Digital Systems	Vibhas Ratna, Pritesh Khairnar, Shashank Bhardwaj, Ravi Khatri	2023	RBAC implementation in real system	Shows practical RBAC application in academic social platforms; demonstrates role hierarchy and access control enforcement	Focused on university systems; not generalized to enterprise RBAC
Static Analysis-Based Approaches for Secure Software Development	Miltiadis Siavvas, Erol Gelenbe, Dionysios Kehagias, Dimitrios Tzovaras	2022	Static security analysis	Uses static analysis to detect vulnerabilities early in software; highlights compile-time security enforcement	Not directly integrated with RBAC DSL; theoretical focus
Enhancing Security and Flexibility with Combined RBAC and ABAC Models	P.M. Jyosthna, Sachin Kumar Ray, Anirudh Varma Mandapati, B. Yashwanth Sai Kumar, Matam Surya Teja	2022	Hybrid RBAC+ABAC	Combines RBAC and ABAC for flexible access control; handles complex permission scenarios	Experimental; limited large-scale deployment
A Systematic Review of Access Control Models	Nastaran Farhadighalati, Luis A. Estrada-Jimenez, Sanaz Nikghadam-	2021	Systematic literature survey	Provides comprehensive survey of access control models; highlights gaps and challenges	High-level survey; does not propose concrete tools

	Hojjati, Jose Barata				
A Survey Paper on Role Based Access Control	Dipmala Salunke, Anilkumar Upadhyay, Amol Sarwade, Vaibhav Marde, Sachin Kandekar	2020	Literature survey	Provides a comprehensive survey of RBAC models, implementations, and security challenges; identifies common gaps in existing approaches	High-level survey; no experimental evaluation or new tool proposed

Key Learnings from Literature Survey

1. **NIST RBAC Model (2020)**
    - Standardizes RBAC components: users, roles, permissions, role hierarchies.
    - Provides a formal foundation for RBAC implementations.
    - Highlights importance of clearly defined policy specifications.
2. **Model Checking Access Control Policies (Google Cloud IAM) (2023)**
    - Detects policy conflicts and inconsistencies before deployment.
    - Demonstrates automated correctness checks for RBAC policies.
    - Shows the value of early verification in preventing misconfigurations.
3. **Privilege Escalation Detection using Model-Driven Engineering (2022)**
    - Detects direct and indirect privilege escalation.
    - Emphasizes role inheritance as a potential source of security vulnerabilities.
    - Guides design of static security analysis modules in a compiler.
4. **Automated RBAC Assessment for Microservice Mesh (2021)**
    - Centralized static assessment identifies misconfigurations early.
    - Detects redundant permissions and over-privileged roles.
    - Useful for modular static analyzers in policy verification.

5. **Static Analysis for Access Control Policies (2024)**
    - Rule-based static evaluation enables early detection of inconsistencies.
    - Highlights need for explainable outputs for detected issues.
    - Motivates integration into a full compiler pipeline.
6. **Securing Academic Social Platforms (2023)**
    - Demonstrates real-world RBAC implementation with role hierarchies.
    - Shows practical user-role mappings and permission assignments.
    - Reinforces importance of enforcing access control in applications.
7. **Static Analysis-Based Approaches for Secure Software Development (2022)**
    - Early detection of software vulnerabilities via static analysis.
    - Emphasizes compile-time, deterministic, and explainable results.
    - Inspiration for static RBAC policy checking in a compiler.
8. **Enhancing Security and Flexibility with Combined RBAC and ABAC Models (2022)**
    - Combines RBAC with ABAC to handle complex permission scenarios.
    - Increases flexibility of access control systems.
    - Motivates hybrid approaches for modern policy specifications.
9. **A Systematic Review of Access Control Models (2021)**
    - Highlights research gaps and challenges in access control models.
    - Identifies trends in RBAC and ABAC approaches.
    - Guides design decisions for compiler-based static analysis.
10. **A Survey Paper on Role Based Access Control (2020)**

- Summarizes gaps and limitations in current RBAC implementations.
- Emphasizes need for automated verification and static analysis.
- Provides a baseline justification for the RBAC compiler project.

5. Comparative Analysis of Existing Approaches

Aspect	RBAC Standards (e.g., NIST)	RBAC / IAM Tools	Research-Based Approaches	Proposed Compiler-Based Approach
Policy Representation	Conceptual RBAC model	Configuration rules	Formal or mathematical models	Domain-Specific Language (DSL)
Stage of Analysis	Runtime enforcement	Runtime enforcement	Mostly post-deployment	Compile-time
Static Security Analysis	Not supported	Very limited	Partially supported	Fully supported
Role Conflict Detection	Manual	Limited	Supported	Fully automated
Redundant Permission Detection	Not supported	Rarely supported	Partially supported	Fully supported
Privilege Escalation Detection	Not addressed	Limited	Supported (often post-deployment)	Compile-time detection (direct and indirect)
DSL-Level Policy Verification	Not available	Not available	Rarely available	Fully supported
Explainability of Results	Not applicable	Minimal	Limited	Human-readable explanations
Scalability for Large Policies	Conceptual only	High (for enforcement)	Limited	High
Ease of Policy Authoring	Moderate	Moderate	Low (complex formalisms)	High (structured DSL)

Key Observations from Comparative Analysis

- Existing RBAC standards focus on policy definition, not verification
- IAM tools prioritize runtime enforcement, not early error detection
- Research approaches provide theoretical analysis but lack practical integration
- None of the existing approaches combine:
    - DSL-based policy specification
    - Compiler-style static analysis
    - Explainable security reporting
- The proposed compiler-based approach addresses all these limitations in a unified framework

6. Identification of Research Gaps

Based on the detailed study of RBAC standards, existing IAM systems, and recent research work, several critical research gaps have been identified. These gaps highlight the limitations of current approaches and clearly motivate the need for the proposed compiler-based static analysis framework.

6.1 Lack of DSL-Based RBAC Policy Specification

- Existing RBAC standards and IAM tools:
    - Rely on configuration files or administrative rules
    - Do not provide a formal “Domain-Specific Language (DSL)” for RBAC policy definition
- Absence of a DSL leads to:
    - Ambiguous and error-prone policy specifications
    - Difficulty in performing syntax and structural validation
- Limited or no support for:
    - Early syntax checking
    - Structural consistency validation of RBAC policies

6.2 Limited Compile-Time Security Analysis

- Most existing approaches focus on:
    - Runtime access control enforcement
    - Post-deployment security audits
- Compile-time analysis of RBAC policies is:

    - Rarely supported
    - Not systematically implemented
- As a result:
    - Security misconfigurations remain undetected until deployment
    - Logical errors propagate into production systems

6.3 Poor Explainability of Detected Security Issues

- Existing tools and research approaches often:
    - Report security violations as low-level warnings
    - Provide limited contextual information
- Lack of:
    - Clear explanation of why a violation occurs
    - Guidance for correcting RBAC misconfigurations
- This makes:
    - Debugging complex RBAC policies difficult
    - Manual correction time-consuming and error-prone

6.4 Inadequate Privilege Escalation Detection

- Many existing approaches detect:
    - Only direct privilege escalation scenarios
- Limited support for:
    - Indirect or transitive privilege escalation
    - Multi-level role inheritance analysis
- Graph-based analysis techniques are:
    - Used inconsistently
    - Rarely automated in practical tools

6.5 Fragmented and Non-Integrated Solutions

- Existing research solutions:
    - Address individual RBAC security problems in isolation
    - Lack end-to-end integration

- No unified framework exists that combines:
    - Policy specification
    - Parsing and validation
    - Static security analysis
    - Explainable reporting

7. Problem Gap Statement

The literature survey indicates that although Role-Based Access Control (RBAC) is widely adopted and extensively studied, existing standards, tools, and research approaches do not adequately support early detection of RBAC policy misconfigurations. Most RBAC standards focus on policy definition and runtime enforcement, without providing mechanisms for verifying policy correctness before deployment.

Current RBAC and IAM tools primarily enforce access decisions at runtime and offer limited support for detecting logical errors such as role conflicts, redundant permissions, and unintended privilege escalation in advance. As a result, security vulnerabilities often remain unnoticed until after deployment or during manual audits.

While research-based approaches propose formal and graph-based techniques for RBAC security analysis, they are often complex, lack practical integration, and provide limited explainability for detected issues. Very few approaches treat RBAC policies as analyzable specifications or support compile-time security verification.

Therefore, there exists a clear problem gap for a unified approach that enables DSL-based RBAC policy specification, compiler-style static analysis, and explainable security reporting to detect RBAC misconfigurations before deployment.

8. Justification for Compiler-Based Static Analysis

8.1 Why a Compiler-Based Approach is Suitable

- RBAC policies have structured components:
    - Roles, permissions, users, and inheritance
- These components follow well-defined rules and constraints
- Hence, RBAC policies can be treated as program-like specifications
- Compiler techniques naturally fit this structured analysis

8.2 Benefits of Compile-Time Verification

- Detects security misconfigurations before deployment

- Prevents insecure RBAC policies from entering production
- Provides early feedback during policy design
- Reduces dependence on runtime monitoring and audits

8.3 Role of Static Security Analysis

- Analyzes RBAC policies without execution
- Enables detection of:
    - Role conflicts
    - Redundant permissions
    - Direct and indirect privilege escalation
- Graph-based analysis supports:
    - Role hierarchy modeling
    - Permission propagation analysis using DFS/BFS

8.4 Advantages Over Existing Approaches

- Unlike runtime-only systems:
    - Identifies logical errors at design time
- Unlike isolated research tools:
    - Integrates DSL design, parsing, analysis, and reporting
- Produces human-readable and explainable security diagnostics
- Supports secure-by-design RBAC policy development