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 4 Deliverables

Week 4 – Compiler Architecture & System Design

1. Introduction

Week 4 focuses on the system design of the RBAC Policy DSL Compiler. The main objective is to define the architecture, show how modules interact, and justify the tools and technologies chosen for implementation. This stage ensures a clear understanding of the system flow and prepares for coding the compiler.

2. Compiler Architecture Diagram

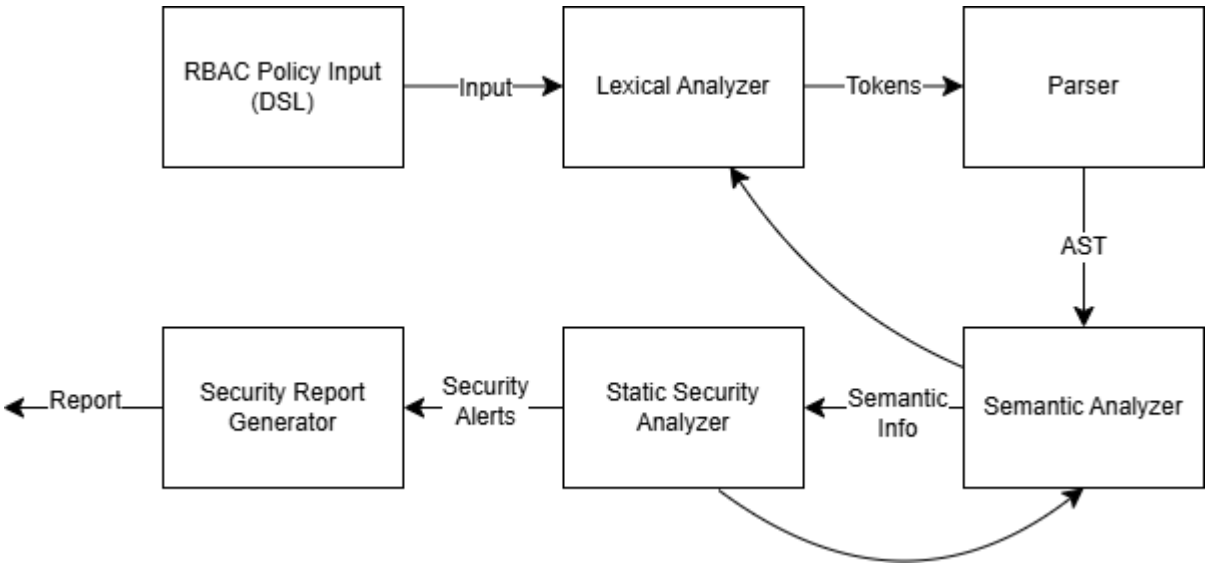
Purpose:
The Compiler Architecture Diagram visually represents the main components of the RBAC Policy DSL Compiler and the flow of data between them. It helps understand how RBAC policies are parsed, analyzed, and reported.

Components:

1. **RBAC Policy Input (DSL)** – The policy file containing roles, permissions, and users.
2. **Lexical Analyzer** – Converts the DSL input into tokens.
3. **Parser** – Generates an Abstract Syntax Tree (AST) from tokens.
4. **Semantic Analyzer** – Performs semantic validation, checks role hierarchies, and detects conflicts.
5. **Static Security Analyzer** – Detects privilege escalation, redundant permissions, and over-privileged roles.
6. **Security Report Generator** – Produces human-readable and explainable reports of detected issues.

Data Flow:

- Input → Lexical Analyzer → Parser → Semantic Analyzer → Static Security Analyzer → Security Report Generator



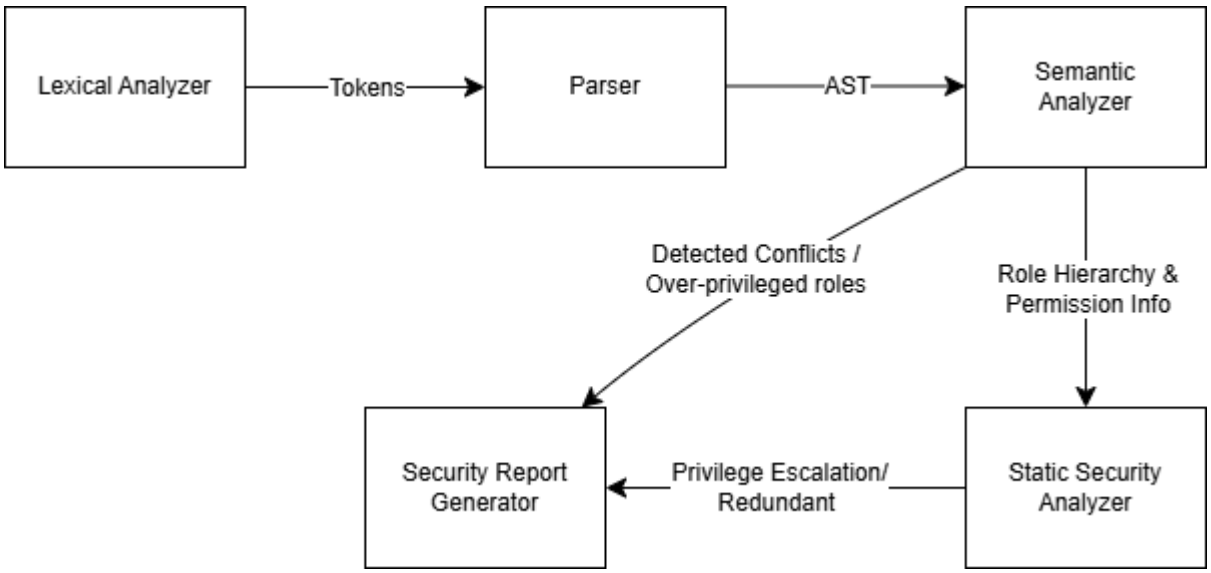
3. Module Interaction Diagram

Purpose:

The Module Interaction Diagram shows how the different components communicate with each other, highlighting data dependencies and the mapping of security threats to analysis modules.

Interactions:

Source Module	**Target Module**	**Data / Purpose**
Lexical Analyzer	Parser	Tokens
Parser	Semantic Analyzer	AST
Semantic Analyzer	Static Security Analyzer	Role hierarchy & permission info
Semantic Analyzer	Security Report Generator	Detected conflicts, over-privileged roles
Static Security Analyzer	Security Report Generator	Privilege escalation, redundant permissions



4. Tool and Technology Justification

Tool / Technology	Purpose	Justification
Python 3.x	Implementation language	Easy syntax, strong library support, dynamic typing, and object-oriented programming for compiler construction
PLY / ANTLR (Python version)	DSL parsing	Efficient lexical analysis, parsing, and AST generation in Python
draw.io	Diagram design	Easy creation of architecture and interaction diagrams
pip / virtualenv	Dependency management	Isolated Python environment and easy package installation
Git & GitHub	Version control	Tracks changes, supports collaboration, and submission
unittest / pytest	Testing	Unit testing of compiler modules ensures correctness

Justification:

Python provides a flexible and readable environment for building a DSL compiler. Libraries like PLY or ANTLR for Python handle parsing efficiently. Using version control, virtual environments, and testing frameworks ensures smooth development, reproducibility, and maintainability.