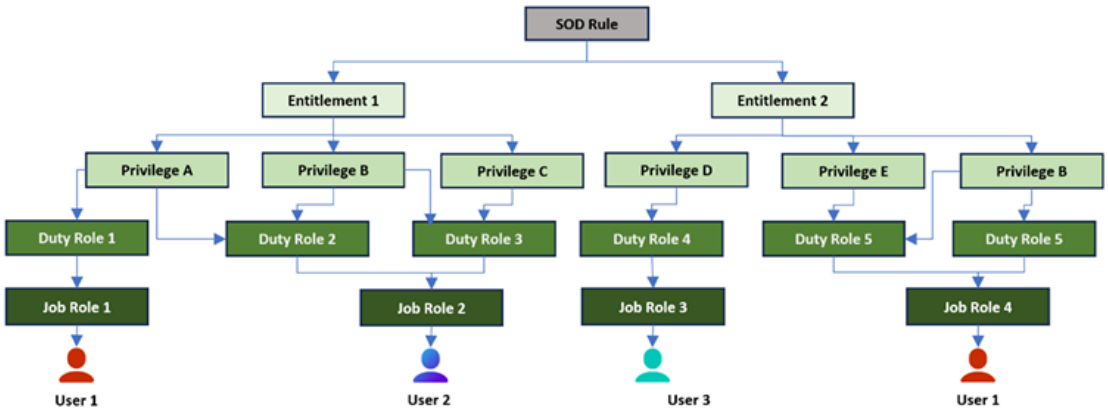
SECURE CODE HACKATHON

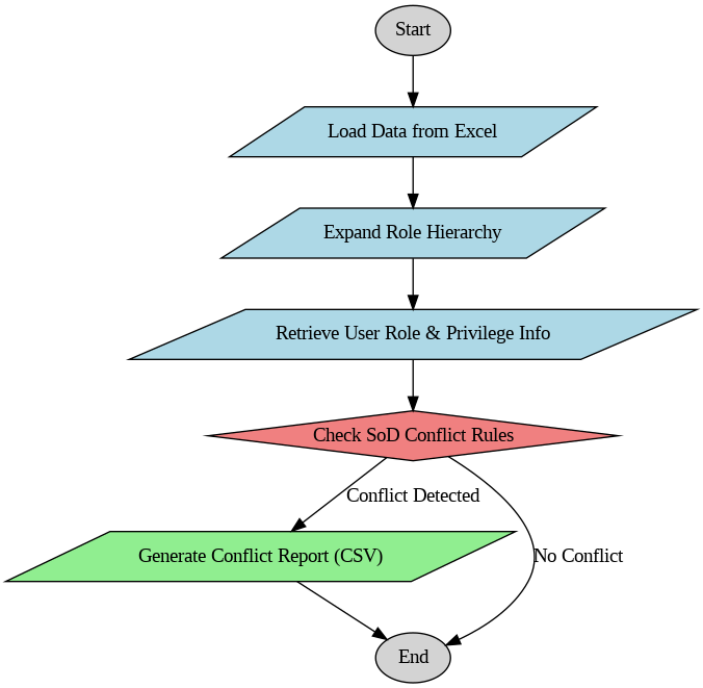
TEAM DR:

DEBDEEP BANERJEE(HERITAGE INSTITUTE OF TECHNOLOGY)

ROHAN JAIN(JADAVPUR UNIVERSITY)



APPROACH:



Problem Statement

The objective of this project is to detect **Segregation of Duties (SoD) conflicts** in an **ERP system** using provided dataset files. The algorithm ensures **deterministic** access risk analysis, preventing unauthorized role conflicts.

High-Level Design:

1. **Load Data from Excel Files**
   - Extract user-role mappings, role hierarchy, privileges, and SoD rules.
2. **Expand Role Privileges**
   - Resolve hierarchical relationships to find **inherited** privileges.
3. **Detect SoD Conflicts**
   - Cross-reference SoD rules against assigned user roles and privileges.
4. **Output Conflicts in CSV Format**
   - Structured report listing all violations.

2. EXPAND ROLE PRIVILEGES

- IDENTIFY PARENT-CHILD ROLE RELATIONSHIPS.
- AGGREGATE INHERITED PRIVILEGES FROM TOP-LEVEL ROLES TO LEAF-LEVEL ROLES.
- ENSURE THAT ALL ROLES CARRY THE NECESSARY PRIVILEGES WITHOUT DUPLICATION.

3. RETRIEVE USER ROLE & PRIVILEGE INFORMATION

- ASSIGN EFFECTIVE PRIVILEGES TO EACH USER BY EXPANDING ROLES INTO PERMISSIONS.
- CONVERT ROLE-BASED ACCESS CONTROL (RBAC) INTO A PRIVILEGE-BASED MODEL.

4. CHECK SOD CONFLICT RULES

- COMPARE ASSIGNED PRIVILEGES AGAINST PREDEFINED SOD RULES.
- IF A USER HAS BOTH CONFLICTING ENTITLEMENTS, IT IS FLAGGED AS A RISK.

5. GENERATE CONFLICT REPORT IN CSV FORMAT

- OUTPUT STRUCTURED RESULTS LISTING:
  - USER ID
  - CONFLICTING PRIVILEGES
  - ROLES RESPONSIBLE FOR CONFLICTS
  - SOD POLICY VIOLATED
- ENSURES A CLEAR AND AUDITABLE RECORD OF VIOLATIONS.

FUNCTION DESCRIPTIONS:

Function Descriptions

Data Loading :The system reads input data from multiple CSV files containing user-role mappings, privilege assignments, and SoD rules.

Main Functions: parseCSV(filename) (Located in parser.cpp) Reads a CSV file and parses rows into a vector of strings. Skips the header row for cleaner processing. Handles missing data gracefully. Example usage: auto data = parseCSV("XX_2_USER_DETAILS_RPT.csv"); File Processed: XX_2_USER_DETAILS_RPT.csv, XX_3_USER_ROLE_MAPPING_RPT.csv, XX_4_ROLE_MASTER_DETAILS_RPT.csv, XX_5_ROLE_TO_ROLE_HIER_RPT.csv, XX_6_PVLG_TO_ROLE_RELATION_RPT.csv, XX_7_PVLGS_MASTER_RPT.csv, SOD_Ruleset.xlsx

Core Algorithm The core logic of the system is responsible for performing risk analysis by detecting SoD violations. It ensures that no user is assigned conflicting roles or privileges.

Algorithm Flow: User inputs a PRIVILEGE_NAME. Privilege Lookup (Located in privilege_lookup.cpp) getPrivilegeID(privilege_name) fetches PRIVILEGE_ID from XX_7_PVLGS_MASTER_RPT.csv. Role Lookup (Located in role_lookup.cpp) getRolesForPrivilege(privilege_id) retrieves ROLE_IDs linked to that privilege from XX_6_PVLG_TO_ROLE_RELATION_RPT.csv. Role Hierarchy Expansion getTopParentRole(role_id) resolves parent-child role relationships from XX_5_ROLE_TO_ROLE_HIER_RPT.csv. User Lookup (Located in user_lookup.cpp) getUserIDsForRole(role_id) finds users mapped to roles from XX_3_USER_ROLE_MAPPING_RPT.csv. SoD Conflict Detection The system cross-checks user-role assignments with SOD_Ruleset.xlsx to determine if conflicting roles exist. If a user has conflicting entitlements, it is flagged as a risk. Report Generation (Located in main.cpp) Outputs violations to output.csv in a structured format.

Helper Functions These functions support the main algorithm by performing role expansions and reporting.

getTopParentRole(role_id) (Located in role_lookup.cpp) Resolves hierarchical role relationships to find the highest-level parent role. Prevents duplicate privilege assignments across nested roles. getUserDetails(user_ids) (Located in user_lookup.cpp) Fetches full user details from XX_2_USER_DETAILS_RPT.csv. Ensures the correct user-role mapping is presented. printUsers(users) (Located in utils.cpp) Displays the final list of identified users and violations. Ensures that reports are readable and easy to analyze.

utils.cpp Contains helper functions such as printUsers(). Formats and displays reports for easy review. Ensures that outputs are structured and readable.

TESTING METHODOLOGY

Testing Methodology

Unit Testing

Each function was independently tested using mock data to ensure correctness.

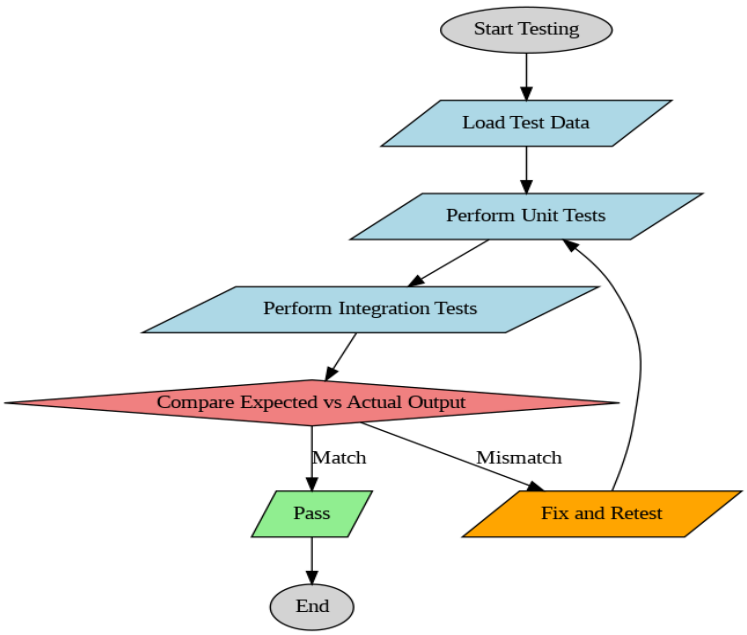
- `parseCSV(filename)`: Verified correct parsing of CSV files by checking row counts and expected column values.
- `getPrivilegeID(privilege_name)`: Ensured correct mapping between privilege names and privilege IDs.
- `getRolesForPrivilege(privilege_id)`: Validated retrieval of ROLE_IDs associated with a privilege.
- `getTopParentRole(role_id)`: Ensured correct resolution of hierarchical role relationships.
- `getUserIDsForRole(role_id)`: Confirmed accurate mapping of users to roles.

Integration Testing

- Loaded the full dataset from Excel files and ensured that all mappings (user-role, role-privilege, SoD rules) were processed correctly.
- Simulated real-world test cases where users had specific roles and checked if conflicts were detected as per the SoD rules.
- Compared the generated conflict report against expected test case outputs.

Performance Testing

- Evaluated execution time on large datasets to ensure scalability.
- Optimized memory usage and processing speed using multi-threading.
- Measured system response time with increasing data sizes to verify efficiency.



SETUP AND EXECUTION INSTRUCTIONS

Setup and Execution Instructions

1. Prerequisites

Before running the application, ensure the following dependencies are installed:

- **Docker: Install Docker from Docker Official Site.**
- **C++ Compiler (g++): Required for local compilation (optional if running via Docker).**
- **Required Data Files: Ensure all necessary Excel/CSV files are placed in an accessible directory.**

2. Building the Docker Image

The application is packaged inside a Docker container for consistent execution.

1. **Navigate to the Project Directory:**

   ```
   sh
   CopyEdit
   cd /path/to/your/project
   ```

2. **Create a Dockerfile (if not already created):**

   ```
   dockerfile
   CopyEdit
   # Use an official lightweight Ubuntu image with GCC
   FROM ubuntu:22.04
   # Install necessary dependencies
   RUN apt-get update && apt-get install -y g++ cmake make xlsxio-utils
   libxlsxio-dev
   # Set the working directory inside the container
   WORKDIR /app
   # Copy the source code into the container
   COPY . /app
   # Compile the C++ program
   RUN g++ -std=c++17 -o sod_analyzer main.cpp parser.cpp
   privilege_lookup.cpp role_lookup.cpp user_lookup.cpp utils.cpp -
   pthread -lxlsxio
   # Define command to run the application
   ENTRYPOINT ["/app/sod_analyzer"]
   ```

3. **Build the Docker Image:**
   Run the following command inside the project directory where the `Dockerfile` is located:

   ```
   sh
   CopyEdit
   docker build -t sod_analysis .
   ```

3. Running the Docker Container

To execute the application, provide input and output directory mappings using Docker's volume feature.

1. **Prepare the Input Files**
   Ensure all required input Excel/CSV files are placed inside a directory (e.g., `test_data`):

   ```
   CopyEdit
   test_data/
   ```

```
├── XX_2_USER_DETAILS_RPT.xlsx
├── XX_3_USER_ROLE_MAPPING_RPT.xlsx
├── XX_4_ROLE_MASTER_DETAILS_RPT.xlsx
├── XX_5_ROLE_TO_ROLE_HIER_RPT.xlsx
├── XX_6_PVLG_TO_ROLE_RELATION_RPT.xlsx
├── XX_7_PVLGS_MASTER_RPT.xlsx
├── SOD_Ruleset.xlsx
```

2. **Run the Docker Container with Input and Output Paths:**

```
sh
CopyEdit
docker run -v "$(pwd)/test_data:/input" -v "$(pwd)/output:/output"
sod_analysis /input/XX_3_USER_ROLE_MAPPING_RPT.xlsx
/input/XX_5_ROLE_TO_ROLE_HIER_RPT.xlsx
/input/XX_6_PVLG_TO_ROLE_RELATION_RPT.xlsx /input/SOD_Ruleset.xlsx
/output/results.csv
```

   - `-v "$(pwd)/test_data:/input"` → Mounts the test data directory as `/input` in the container.
   - `-v "$(pwd)/output:/output"` → Mounts the output directory as `/output` in the container.
   - Arguments: The paths provided to the program specify the required input files and the output file location.

3. **Verify the Output:**
   After execution, check the output directory:

```
sh
CopyEdit
cat output/results.csv
```

Expected format:

```
pgsql
CopyEdit
User, Conflict Rule
JohnDoe, Create & Approve Payables
JaneSmith, Modify & Approve Payments
```

4. Debugging and Troubleshooting

Check Running Containers:

```
sh
CopyEdit
docker ps
```

```
Check Container Logs (if errors occur):
sh
CopyEdit
docker logs <container_id>
```

Remove Unused Containers and Images:

```
sh
CopyEdit
docker system prune -f
```

This ensures a fully reproducible environment for running the analysis without modifying the source code.