Bell-LaPadula Model Implementation in Java

Team: Save Water

- Jacob Pugh
- Brandon Magana
- Eli Manning

Assignment Overview

This project implements the Bell-LaPadula (BLP) security model in Java, demonstrating confidentiality-focused access control through the enforcement of two fundamental security properties: - **Simple Security Property** (“no read up”): Users cannot read objects with a higher security level - **Star Property (★-property)** (“no write down”): Users cannot write to objects with a lower security level

Project Structure

```
src/
├── Main.java
└── models/
    ├── BaseModel.java
    ├── MissionSpecModel.java
    ├── WeaponsSpecModel.java
    ├── User.java
    └── SecurityLevel.java
```

System Components

Security Levels

The system implements four hierarchical security levels (lowest to highest): 1. **Unclassified** (Level 1) 2. **Confidential** (Level 2) 3. **Secret** (Level 3) 4. **Top Secret** (Level 4)

Classes Description

`SecurityLevel.java`

- Enumeration defining the four security classification levels
- Each level has a name and numerical clearance level for comparison

`User.java`

- Represents system users with security clearances
- Contains: first name, last name, unique UUID, and security level

`BaseModel.java`

- Abstract base class for all secure objects
- Implements BLP access control logic
- Contains:
  - Generic Field system for type-safe dynamic attributes

    - `get()` method enforcing “no read up” rule
    - `set()` method enforcing “no write down” rule
    - Security level, UUID, and timestamps

`MissionSpecModel.java`

- Extends BaseModel to represent mission specification documents
- Fields: Title, Content

`WeaponsSpecModel.java`

- Extends BaseModel to represent weapons specification documents
- Fields: Name, Description, Classification

`Main.java`

- Interactive simulation driver
- Manages collections of users and objects
- Provides command-line interface for testing BLP rules

Pre-configured Test Data

Users

Name	Security Level	UUID (generated at runtime)
Valentine Davis	Unclassified	Auto-generated
Robert Yestur	Confidential	Auto-generated
Brandon Lee	Secret	Auto-generated
Pauline Pugh	Top Secret	Auto-generated

Objects

Object	Type	Security Level	UUID (generated at runtime)
Ash	Mission Spec	Top Secret	Auto-generated
Woods	Mission Spec	Confidential	Auto-generated
Miracle	Weapon Spec	Secret	Auto-generated
Bombastic Side Eye	Weapon Spec	Unclassified	Auto-generated

How to Compile and Run

Prerequisites

- Java Development Kit (JDK) 8 or higher
- Command line access (Terminal, Command Prompt, or PowerShell)

Compilation Steps

1. **Navigate to the project directory:**

   ```
   cd path/to/project
   ```

2. **Compile all Java files:**

   ```
   javac -d bin src/Main.java src/models/*.java
   ```

   Or, if not using a bin directory:

```
javac src/Main.java src/models/*.java
```

3. **Run the program:**

   ```
   java -cp bin Main
   ```

   Or, if compiled without bin directory:

   ```
   cd src
   java Main
   ```

Alternative: Using an IDE

1. Import the project into your IDE (Eclipse, IntelliJ IDEA, NetBeans, VS Code)
2. Ensure the source folder is properly configured
3. Run `Main.java`

Usage Instructions

When the program starts, it will: 1. Display all available users with their UUIDs and security levels 2. Display all available objects with their UUIDs and security levels

Interactive Simulation

The program will prompt you for the following inputs in sequence:

1. **User ID**: Enter the UUID of the user you wish to simulate
   - Copy/paste a UUID from the displayed user list
2. **Object ID**: Enter the UUID of the object you wish to access
   - Copy/paste a UUID from the displayed object list
3. **Operation**: Enter either `READ` or `WRITE`
   - Case-insensitive
4. **Field**: Select which field of the object to access
   - Available fields are displayed for the selected object
   - Field names are case-sensitive (e.g., “Title”, “Name”)
5. **Value** (for WRITE operations only): Enter the value to write

Example Interaction

```
Enter the User Id you wish to simulate as: a1b2c3d4-e5f6-7890-abcd-
ef1234567890

Enter the Object Id you wish to touch: 12345678-90ab-cdef-1234-
567890abcdef

Enter the operation you wish to preform (READ, WRITE): READ

Mission Spec:
Title (class java.lang.String)
Content (class java.lang.String)

Enter the field you wish to preform the operation on (case-sensitive):
Title

Ash
```

BLP Rules Implementation

Simple Security Property (No Read Up)

Implemented in `BaseModel.hasReadAcess()`:

```
return currentUser.getUserSecurityLevel().getClearanceLevel()
       >= this.securityLevel.getClearanceLevel();
```

- Users can only read objects at their level or below
- Prevents information flow from high to low through reads

Star Property (No Write Down)

Implemented in `BaseModel.hasWriteAcess()`:

```
return currentUser.getUserSecurityLevel().getClearanceLevel()
       <= this.securityLevel.getClearanceLevel();
```

- Users can only write to objects at their level or above
- Prevents information flow from high to low through writes

Key Features

Type-Safe Field System

- Generic `Field<T>` class ensures type safety for object attributes
- Dynamic field mapping with compile-time type checking
- Null-safe casting mechanism

UUID-Based Identification

- All users and objects identified by unique UUIDs
- Prevents conflicts and ensures uniqueness

Timestamp Tracking

- Objects track creation time and last modification time
- Foundation for audit logging capabilities

Error Handling

- Invalid UUID detection with user-friendly prompts
- Invalid operation validation
- Invalid field name handling
- Comprehensive access denial messages explaining BLP violations

Limitations and Future Enhancements

Current Limitations

- Security levels are fixed at object/user creation
- No persistence (data lost when program exits)
- No audit logging of access attempts

Dependencies

- Java Standard Library only
- No external dependencies required