Static Code Analysis Report

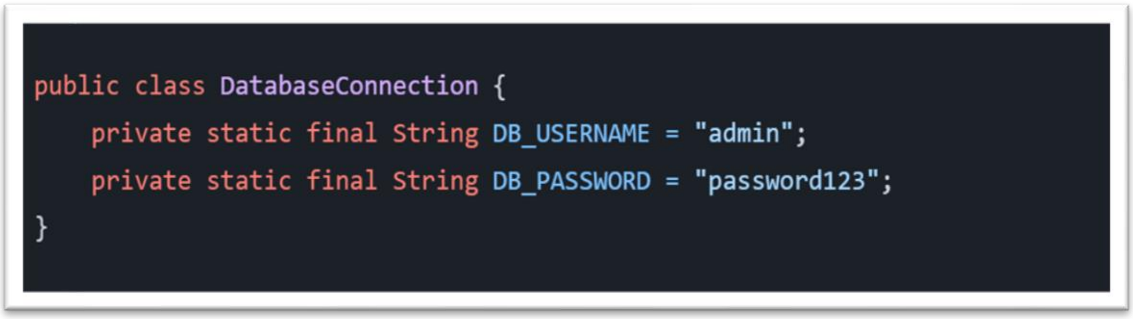
Executive Summary

This report presents a static code analysis of the **Budget Manager** application, focusing on security vulnerabilities, code quality issues, and best practices. The analysis identified several critical issues, including hardcoded credentials, potential SQL injection vulnerabilities, and unencrypted data storage. Addressing these vulnerabilities is essential to ensure the security and integrity of user financial data.

Findings

Finding 1: Hardcoded Credentials

- **CVSS Score:** 8.0
- **Severity:** High
- **CVSS Vector:** AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
- **Description of the finding:** The source code contains hardcoded database credentials, making them easily discoverable for attackers.
- **Screenshots and Proof of Concepts:**



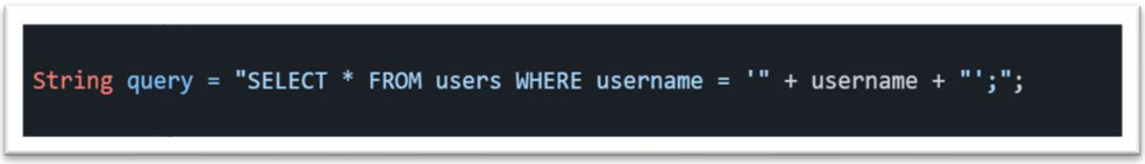
```
public class DatabaseConnection {
    private static final String DB_USERNAME = "admin";
    private static final String DB_PASSWORD = "password123";
}
```

- **Impact:** Exposure of database credentials can lead to unauthorized access, data breaches, and manipulation of user data.
- **Recommendations:** Utilize environment variables or secure vaults to manage sensitive information instead of hardcoding it in the source code.
- **References:** OWASP - Hardcoded Secrets

Finding 2: SQL Injection Vulnerability

- **CVSS Score:** 7.5
- **Severity:** High
- **CVSS Vector:** AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N
- **Description of the finding:** Use of string concatenation in SQL queries without parameterized statements increases the risk of SQL injection.

- **Screenshots and Proofs of Concept:**

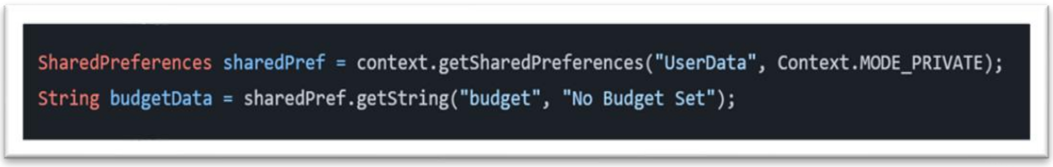


```
String query = "SELECT * FROM users WHERE username = '" + username + "';";
```

- **Impact:** This vulnerability could allow attackers to manipulate SQL queries, leading to unauthorized data access or modification.
- **Recommendations:** Implement prepared statements or ORM frameworks to prevent SQL injection through parameterized queries.
- **References:** [OWASP - SQL Injection]()

Finding 3: Insecure Data Storage

- **CVSS Score:** 6.0
- **Severity:** Medium
- **CVSS Vector:** AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N
- **Description of the finding:** User financial data is stored unencrypted on the device, making it susceptible to unauthorized access.
- **Screenshots and Proofs of Concept:**



```
SharedPreferences sharedPref = context.getSharedPreferences("UserData", Context.MODE_PRIVATE);
String budgetData = sharedPref.getString("budget", "No Budget Set");
```

- **Impact:** Unencrypted sensitive data can be accessed by malicious applications or users, risking the user's financial privacy.
- **Recommendations:** Use Android's EncryptedSharedPreferences or other encryption libraries to secure sensitive data stored on devices.
- **References:** [OWASP - Insecure Storage]()

Conclusion

The static code analysis of the Budget Manager application revealed several critical issues that need immediate attention. Particularly, hardcoded credentials and SQL injection vulnerabilities pose significant risks to user data and application security. Implementing the recommended practices will greatly enhance the application's security posture and protect users' sensitive information.