Mobile Application Penetration Testing Report: Diva-Beta.apk

1. Introduction

This report provides a comprehensive analysis of the mobile application *Diva-Beta.apk* after conducting a static analysis using various tools and techniques. The purpose of this analysis was to identify potential security vulnerabilities within the application. Although the dynamic analysis could not be performed due to resource limitations, the static analysis revealed several significant security issues.

2. Tools and Methodologies

- **Decompiling and Code Review Tools:**
  - **APKTool**: Used for decompiling the APK to access the underlying resources and code structure.
  - **JADX GUI**: Enabled decompilation of the APK into Java source code for an in-depth code review.
  - **Kali Linux** (Virtual Box): Provided the necessary environment and tools to perform the analysis.
  - **Android Emulator**: Simulated a real Android environment to understand how the application interacts with the system.
- **Security Analysis Tool:**
  - **MobSF (Mobile Security Framework)**: Automated tool used to perform static analysis, including scanning for known vulnerabilities, reviewing code for security issues, and identifying insecure practices.

3. Findings

3.1 Insecure Logging of Data

- The application was found to log sensitive information insecurely. Log files are accessible in the file system and may contain data such as usernames, passwords, or other sensitive user inputs. This is a significant issue, as attackers with access to the device could retrieve this information from the logs.
- **Recommendation:** Ensure that sensitive information is either not logged or that logging is securely managed by masking or encrypting sensitive data.

3.2 Insecure Storage

- *Diva-Beta.apk* was discovered to store sensitive information insecurely on the device. For example, user credentials and tokens were stored in plain text within the application's

local storage. This vulnerability could lead to unauthorized access if an attacker gains physical or remote access to the device.
- **Recommendation:** Ensure that sensitive information is either not logged or that logging is securely managed by masking or encrypting sensitive data.

3.3 Hardcoding of Secrets

- Hardcoded secrets such as API keys, credentials, and other sensitive configurations were found within the source code. These secrets are easily retrievable by anyone with access to the decompiled source code, posing a significant risk if they are used in a production environment.
- **Recommendation:** Remove hardcoded secrets from the application and consider using secure storage mechanisms or environment variables to manage them.

3.4 Insecure Access to Data

- The application was also vulnerable to unauthorized data access due to poorly implemented access controls. For instance, sensitive data was accessible through components such as Content Providers without adequate permissions or validation. This could allow other applications or attackers to retrieve sensitive information without proper authorization.
- **Recommendation:** Implement proper access controls to ensure that only authorized users or components can access sensitive data. Review and enforce permissions and API security.

3.5 Vulnerability to SQL Injection

- The code review revealed that the application is susceptible to SQL injection attacks. This vulnerability arises from the direct inclusion of user input in SQL queries without proper sanitization or parameterization. An attacker could exploit this to execute arbitrary SQL commands, potentially leading to data leakage, unauthorized access, or data manipulation.

4. Conclusion

The static analysis of *Diva-Beta.apk* revealed multiple critical vulnerabilities, including insecure logging, insecure storage, hardcoding of secrets, insecure access to data, and susceptibility to SQL injection attacks. These issues highlight the importance of secure coding practices and proper input validation in mobile application development.

While dynamic analysis could not be performed due to resource constraints, the findings from the static analysis are significant and should be addressed to ensure the security of the application and its users.

Recommendations:

1. **Secure Logging Practices**: Avoid logging sensitive data. Ensure that any necessary logging is secured and obfuscated.
2. **Secure Storage**: Use secure storage methods, such as Android's Keystore system, to store sensitive information.
3. **Remove Hardcoded Secrets**: Avoid embedding secrets in the code; instead, consider using secure storage or environment variables.
4. **Implement Access Controls**: Ensure proper access controls are in place for all sensitive data, and restrict access to authorized components only.
5. **Prevent SQL Injection**: Sanitize all user inputs and use prepared statements or ORM frameworks to mitigate SQL injection risks.

Addressing these vulnerabilities is crucial to safeguarding the application's integrity and protecting user data from potential security threats.