**Zomato’s Static Code Analysis:**

**Executive Summary:**  
This report provides a static code analysis of the Zomato application, a popular food delivery and restaurant discovery app with Approax. 1,03,29,000 downloads on Google Play Store. The analysis was conducted to identify potential security vulnerabilities within the application without executing it. The following findings were observed, with corresponding CVSS scores, severity levels, and recommendations to mitigate risks.

**Findings:**

**Finding 1: Hardcoded API Keys and Sensitive Information**  
 **CVSS Score:** 7.5  
 **Severity:** High  
 **CVSS Vector:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N  
 **Description of the Finding:**  
During the analysis of Zomato's decompiled APK, hardcoded API keys were found in the application's source code. Hardcoding sensitive credentials can expose them to attackers who decompile the app and use them for unauthorized API access.  
 **Screenshots and Proof of Concept:**

**Impact:**  
An attacker can exploit the hardcoded API keys to make unauthorized requests, potentially leading to data leaks, service disruptions, and increased costs for Zomato.  
 **Recommendations:**

* Remove hardcoded keys and use secure environment variables or backend API authentication.
* Implement a secure vault for storing API keys.

**References:**

1. OWASP Mobile Security Testing Guide

**Finding 2: Insecure Data Storage**  
 **CVSS Score:** 6.8  
 **Severity:** Medium  
 **CVSS Vector:** CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N  
 **Description of the Finding:**  
The application stores sensitive user information, such as session tokens and personal details, in local storage (SharedPreferences and SQLite) without encryption. This poses a risk if the device is compromised or rooted.  
 **Screenshots and Proof of Concept:**

**Impact:**  
An attacker with local access to the device can extract sensitive user data, leading to account takeovers or identity theft.  
 **Recommendations:**

* Encrypt sensitive data before storing it on the device.
* Use Android’s EncryptedSharedPreferences and SQLCipher for database encryption.

**References:**

1. OWASP Mobile Security Best Practices

**Finding 3: Improper SSL Pinning Implementation**  
 **CVSS Score:** 7.4  
 **Severity:** High  
 **CVSS Vector:** CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N  
 **Description of the Finding:**  
The application does not properly implement SSL pinning, making it susceptible to Man-in-the-Middle (MITM) attacks. During static analysis, it was observed that the app relies solely on the default Android trust store without verifying pinned certificates.  
 **Screenshots and Proof of Concept:**

**Impact:**  
An attacker can intercept user data, including login credentials and payment information, through MITM attacks.  
 **Recommendations:**

* Implement SSL pinning using Certificate Transparency.
* Use a robust library like OkHttp with a pinned certificate list.

**References:**

1. OWASP MASVS: Mobile App Security Verification Standard

**Finding 4: Exposure of Debug Logs in Production**  
 **CVSS Score:** 5.3  
 **Severity:** Medium  
 **CVSS Vector:** CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N  
 **Description of the Finding:**  
Debug logs containing sensitive information were found in the application’s source code. Logging sensitive data in a production environment can lead to information disclosure if logs are exposed.  
 **Screenshots and Proof of Concept:**

**Impact:**  
An attacker with access to logs can extract confidential data, leading to security breaches.  
 **Recommendations:**

* Disable verbose logging in production builds.
* Implement log redaction for sensitive information.

**References:**

1. OWASP Logging Security Guidelines