**Static Security Vulnerability Report: Wuta Camera Android Application**

**Executive Summary**

This report outlines the results of a security analysis conducted on the Wuta Camera Android application. Using static analysis tools such as MobSF and reverse engineering with JADX and bytecode inspection, multiple security vulnerabilities were identified that could potentially expose users to data leakage, exploitation, and other malicious activities. The findings are presented in detail, with explanations of where each vulnerability resides, why it poses a threat, and professional recommendations for mitigation.

**Vulnerability Assessment**

| **#** | **Vulnerability** | **Source** | **Where It's Found** | **Why It’s a Vulnerability** |
| --- | --- | --- | --- | --- |
| 1 | **Cleartext Traffic Enabled** | AndroidManifest.xml | android:usesCleartextTraffic="true" | Allows unencrypted HTTP traffic. This makes it easy for attackers to intercept or alter communication, violating confidentiality and integrity. |
| 2 | **Insecure Random Number Generator** | MobSF Report | Code uses java.util.Random | Predictable random values can allow attackers to guess session tokens or keys, compromising authentication and encryption. |
| 3 | **Hardcoded Sensitive Information** | MobSF Report | Files in assets/ such as grs\_sp.bks and hmsrootcas.bks | Hardcoded keys or credentials can be extracted by reverse engineering the APK, allowing unauthorised access. |
| 4 | **Improperly Exported Components** | AndroidManifest.xml | Activities like WXPayEntryActivity, VendorMsgClickActivity | Exported components without proper permission checks can be accessed by other apps, leading to privilege escalation or injection attacks. |
| 5 | **Weak Hash Algorithm (MD5)** | MobSF Report | Code references MD5 | MD5 is cryptographically broken and vulnerable to collision attacks, making it unreliable for verifying integrity. |
| 6 | **Weak Encryption (AES-CBC + PKCS5)** | MobSF Report | Found in decompiled code | Vulnerable to padding oracle attacks; lacks strong integrity verification like GCM. |
| 7 | **Potential SQL Injection** | MobSF Report | Raw SQL queries found in code | If not properly sanitized, this can allow attackers to manipulate queries and access sensitive database content. |
| 8 | **Janus Vulnerability (V1 Signature Only)** | MobSF Report | App signed using only V1 | Allows modification of APK files without affecting the signature, affecting Android 5.0 to 7.0 users. |
| 9 | **Improper Data Storage on External Media** | Static Analysis | openFileOutput, getExternalFilesDir, getExternalStorageDirectory | Exposes sensitive data to other apps on the device. This violates data confidentiality. |
| 10 | **Sensitive Data Logging** | MobSF + Static Review | Use of Log.d, Log.e, System.out.println | If an attacker gains access to logs, they may retrieve user information or debug data that should be protected. |
| 11 | **Unsafe WebView Handling** | Reverse Engineering | WebView used in multiple classes | WebViews can be exploited if JavaScript is enabled or URLs are loaded without validation. May enable phishing or code injection. |
| 12 | **Deprecated HTTP Client Usage** | Decompiled Code | Use of DefaultHttpClient, HttpURLConnection | These classes are outdated or insecure. They lack modern TLS protections and proper hostname verification. |
| 13 | **Potential Insecure IP-based Communication** | Decompiled Code | http://8.136.104.193:8888/upload.php via OkHttpClient | Usage of a hardcoded IP address over HTTP could allow MitM attacks and violates secure communication practices. |
| 14 | **Unvalidated Input/Output** | Static Analysis | Dynamic string concatenation in logs and network requests | Lacks input validation, which may open the door to injection attacks or logic bugs. |

**Explanation of Selected Key Vulnerabilities**

**1. Cleartext Traffic Enabled**

The android:usesCleartextTraffic="true" flag in the manifest allows the application to send data over HTTP instead of HTTPS. This means an attacker on the same network (e.g., public Wi-Fi) could intercept or manipulate this data. This is especially concerning if sensitive information is transmitted.

**Mitigation**: Enforce HTTPS using android:networkSecurityConfig and disable cleartext traffic.

**3. Hardcoded Keys or Credentials**

The presence of .bks files in the assets folder is suspicious. If these contain hardcoded cryptographic material, they can be easily extracted via reverse engineering and used by attackers to spoof trusted sources.

**Mitigation**: Never embed sensitive keys in the APK. Use secure keystores or remote token retrieval.

**4. Janus Vulnerability**

The APK uses only the V1 signature scheme. This makes it vulnerable to the Janus vulnerability, where an attacker can add malicious code without invalidating the signature on affected Android versions.

**Mitigation**: Sign the application using both V1 and V2 signature schemes or higher.

**10. Sensitive Data Logging**

Logging calls such as Log.d, Log.e, and System.out.println were found across the codebase. These often include user activity and state information, which, if accessed by malicious apps or during runtime, can result in data leakage.

**Mitigation**: Remove or sanitize logs in production builds. Never log personally identifiable information or tokens.

**11. Unsafe WebView Handling**

Multiple WebView classes were identified. If JavaScript is enabled and URLs are not properly validated, this opens the door for XSS attacks or phishing by loading malicious content inside the app.

**Mitigation**: Disable JavaScript unless necessary. Validate URLs strictly before loading.

**13. Insecure Communication Endpoints**

The app connects to an IP address (8.136.104.193:8888) using OkHttpClient over cleartext HTTP. This is extremely dangerous in production apps.

**Mitigation**: Always use domain names and secure them with HTTPS and TLS. Avoid using raw IPs.

**General Recommendations**

* **Apply secure coding practices**: Avoid hardcoding secrets; use well-reviewed cryptographic libraries.
* **Implement proper access control**: Ensure all exposed components are protected by permissions.
* **Use HTTPS for all communications**: Enforce TLS for secure data transmission.
* **Upgrade security protocols**: Use AES-GCM over CBC, and avoid outdated hashes like MD5.
* **Secure user data**: Avoid logging sensitive information and do not store sensitive files on external storage.
* **Use modern libraries**: Replace DefaultHttpClient and HttpURLConnection with OkHttpClient or other maintained libraries.
* **Sign with the latest signature scheme**: Ensure compatibility with V2 or V3 signatures for newer Android versions.

**Conclusion**

The analysis reveals that Wuta Camera contains several serious security vulnerabilities, including improper data handling, outdated cryptographic practices, unsafe component exposure, and insecure networking practices. These weaknesses could potentially be exploited by malicious actors to compromise user data and app functionality. Immediate remediation is necessary to ensure user safety and regulatory compliance.