VAPT For Green Commune

(Sustainability COE APP)

January 2023

Document History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. Rel. No.** | **Release Date** | **Prepared. By Prepared. Dt.** | **Reviewed By** | **Approved By** | **Remarks/Revision Details** |
| 1.0 | 7/01/2023 | Amit Kumar Pradhan | Sunil M.C |  | Version 1 |

**Contents**

1. Overview of the project
2. Objective of the security assessment
3. Approach and test strategy
4. Tools
5. Key security policies
6. Summary of findings
7. Vulnerabilities finding details with description, impact and remediation.
8. Table of vulnerabilities count/findings.
9. **Overview of the project**

This assessment was aimed to perform Vulnerability Assessment and Penetration Testing (VAPT) on Sustainability App (Green Commune) is launched for the employee to use and share their views with respect to sustainable living App usage is expected to motivate the employees to initiate lifestyle changes and ensure their efforts are channeled towards sustainable lifestyle.

Green coins are allotted for lifestyles towards sustainable living, and this is calculated based on LTTS Green thinkers’ recommendation. This will be further improvised based on scientific approach in future.

Green coins allotment is currently a measure to monitor one’s performance towards sustainable lifestyle. Employees participating in this measurement and monitoring tool is to resonate with organization`s sustainability initiative.

The Security testing scope is limited to Sustainability App (Green Commune).

1. **Objective of the security assessment:**

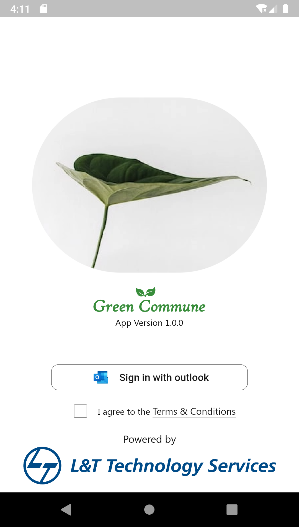
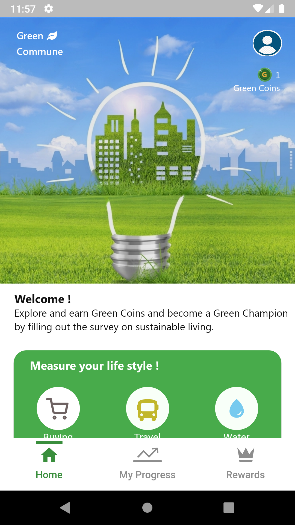
L&T Technology Services (LTTS) has conducted Mobile APP Security Assessment of Sustainability App (Green Commune). The purpose of the assessment is to evaluate the security of the application against common vulnerabilities with the primary reference being guidelines of OWASP Mobile Top 10.

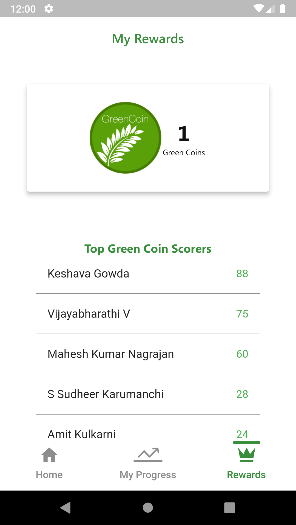
1. **Approach & Test Strategy:**

The following approach was taken to make sure the target was assessed against OWASP Mobile Top 10 from all possible security perspectives:

* + 1. Testing Environment Details

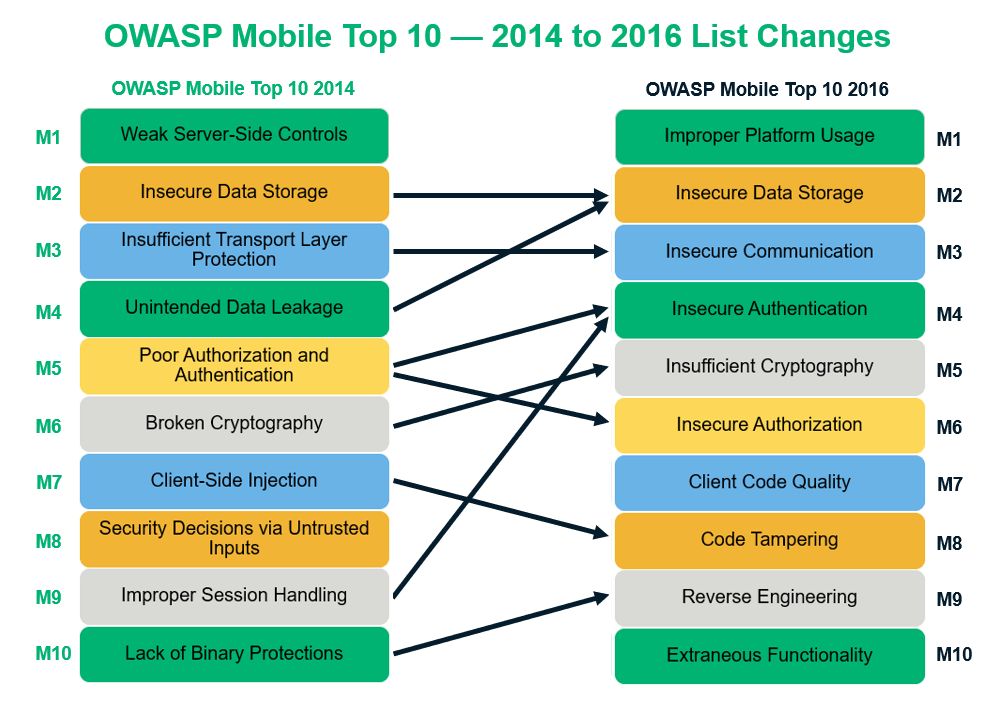
Graphical user interface, application

Description automatically generated   Chart, line chart

Description automatically generated 

Screen 1 Screen 2 Screen 3 Screen 4 Screen 5

|  |  |  |
| --- | --- | --- |
| Screens | Testing Status | Remark |
| 1 Splash | Completed | NA |
| 2 Login with outlook | Completed | NA |
| 3 Home | Completed | NA |
| 4 Progress | Completed | NA |
| 5 Rewards | Completed | NA |
|  |  |  |
|  |  |  |



The test environment is taken for assessment which has the Android and IOS APP.

1. **Tools:**

Some of the tools which were used are listed below:

|  |  |  |
| --- | --- | --- |
| S.no | Tool | Description |
| 1 | Dex2jar | A free tool for converting bytecode from the .dex format into Java class files. |
| 2 | JD-GUI | One of the tools created by the Java De-Compiler project. This graphical utility makes java code readable, displaying it as java class files. |
| 3 | MobSF | Mobile Security Framework is and open-source automated android pen-testing, malware analysis, and security assessment framework capable of performing static and dynamic analysis. |
| 4 | Adb | Android Debug Bridge (adb) is a versatile command-line tool that lets you communicate with a device. |
| 6 | Burp Suite | It is an integrated platform for performing security testing of mobile applications. |
| 7 | Wireshark | Wireshark is a free and open source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, and education |

1. Key Security Policies

Mobile APP Testing based on the OWASP Mobile Top 10.

The following key security aspects were checked:

1. M1: Improper Platform Usage
2. M2: Insecure Data Storage
3. M3: Insecure Communication
4. M4: Insecure Authentication
5. M5: Insufficient Cryptography
6. M6: Insecure Authorization
7. M7: Client Code Quality
8. M8: Code Tampering
9. M9: Reverse Engineering
10. M10: Extraneous Functionality
11. **Summary of Findings**

The graph below shows a summary of the number of vulnerabilities found for each impact level for the Application Security Assessment. Vulnerabilities found are addressed according to priority, findings, analysis and recommendations from the assessment.

**Format of the vulnerability findings.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Vulnerability Name | | | | | |
| **Name** |  | **Impact** | High | **Risk Rating** | Low / High / Moderate / Critical |
| **Ease of Exploit** | Easy/Difficult/Theoretical/Automated Tools Available | **Likelihood** | High |
| **Category** | Vulnerability category | | | | |
| **URL / Impacted System** | --url-- | | | | |
| **Description** | | | | | |
| --description-- | | | | | |
| **Impact** | | | | | |
| --impact-- | | | | | |
| **Remediation** | | | | | |
| --remediation- | | | | | |
| **How to reproduce the Security defect** | | | | | |
| --steps-- | | | | | |
| **CVSS Base Score :** | | --Img-- | | | |
| **Evidence** | | | | | |
| --img-- | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1. **Improper Export of Android Application Components** | | | | | |
| **Name** | Improper Export of Android Application Components | **Impact** | High | **Risk Rating** | **HIGH** |
| **Ease of Exploit** | Automated Tools Available | **Likelihood** | High |
| **Category** | Improper Platform Usage | | | | |
| **URL / Impacted System** | Android App | | | | |
| **Description** | | | | | |
| Improper Platform Usage covers mainly the misusing of platform features or failing to use platform security controls provided and documented by the platform and it’s community. On Android-based applications, the AndroidManifest.xml file is certainly as well as information about the platform configuration and how the application should behave, this file is, indeed, a must-check for security misconfiguration. The Android application exports a component for use by other applications but does not properly restrict which applications can launch the component or access the data it contains.  CWE-926 ( <https://cwe.mitre.org/data/definitions/926.html> ) | | | | | |
| **Impact** | | | | | |
| * Other applications, possibly untrusted, can launch the Activity. * Other applications, possibly untrusted, can bind to the Service. * Other applications, possibly untrusted, can read or modify the data that is offered by the Content Provider. | | | | | |
| **Remediation** | | | | | |
| 1. In application all activities should be exported false 2. If any intent filter is used in application use custom permission 3. Application should not run-on rooted device. | | | | | |
| **How to reproduce the Security defect** | | | | | |
| The components have intent filters but have not explicitly set 'android:exported=false' elsewhere in the manifest, they are automatically exported so that any other application can launch them. This may lead to unintended behavior or exploits. | | | | | |
| **CVSS Base Score :** | |  | | | |
| **Evidence** | | | | | |
|  | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1. **Reliance on Obfuscation** | | | | | |
| **Name** | Encryption of Security-Relevant Inputs without Integrity Checking | **Impact** | High | **Risk Rating** | HIGH |
| **Ease of Exploit** | Automated Tools Available | **Likelihood** | High |
| **Category** | Insufficient Cryptography | | | | |
| **URL / Impacted System** | Android App | | | | |
| **Description** | | | | | |
| * Insecure use of cryptography is common in most mobile apps that leverage encryption. There are two fundamental ways that broken cryptography is manifested within mobile apps. * First, the mobile app may use a process behind the encryption / decryption that is fundamentally flawed and can be exploited by the adversary to decrypt sensitive data. * Second, the mobile app may implement or leverage an encryption / decryption algorithm that is weak in nature and can be directly decrypted by the adversary. * When an application relies on obfuscation or incorrectly applied / weak encryption to protect client-controllable tokens or parameters, that may influence the user state, system state, or some decision made on the server. Without protecting the tokens/parameters for integrity, the application is vulnerable to an attack where an adversary traverses the space of possible values of the said token/parameter to attempt to gain an advantage. The goal of the attacker is to find another admissible value that will somehow elevate their privileges in the system, disclose information or change the behavior of the system in some way beneficial to the attacker. If the application does not protect these critical tokens/parameters for integrity, it will not be able to determine that these values have been tampered with. Measures that are used to protect data for confidentiality should not be relied upon to provide an integrity service.   CWE-649 (<https://cwe.mitre.org/data/definitions/649.html> ) | | | | | |
| **Impact** | | | | | |
| * The problem with CBC mode is the padding. When there is a padding error, the server must respond to a message back to you so that you can send the message back again or encrypt the message from the beginning. * The padding oracle attack is solely based on this idea. The attacker changes the byte and looks at the response of the server to execute the attack. * As we can see the attacker needs an oracle to execute the padding oracle attack. Data on-rest or encrypted databases has no oracle. Therefore, they are not vulnerable to padding oracle attacks. * CBC mode has no integrity and authentication like all basic (archaic) block cipher mode of operations, and it is usually used with HMAC. CBC is removed from TLS 1.3. | | | | | |
| **Remediation** | | | | | |
| 1. The modern, safe option is [authenticated encryption](https://en.wikipedia.org/wiki/Authenticated_encryption), e.g. AES/GCM/NoPadding in modern javax.crypto.Cipher 2. Use encryption standards that you know will hold their own for at least 10 years into the future. 3. Avoid storing sensitive data unencrypted. 4. Avoid using easily guessable encryption keys. 5. Follow NIST guidelines on recommended algorithms. | | | | | |
| **How to reproduce the Security defect** | | | | | |
| MOBSF scanning report. | | | | | |
| **CVSS Base Score :** | |  | | | |
| **Evidence** | | | | | |
|  | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1. **Improper APK signing schemes** | | | | | |
| **Name** | Android Code with improper APK signing schemes | **Impact** | Low | **Risk Rating** | MEDIUM |
| **Ease of Exploit** | Theoretical / Automated Tools Available | **Likelihood** | Medium |
| **Category** | Client Code Quality Information | | | | |
| **URL / Impacted System** | Android APP | | | | |
| **Description** | | | | | |
| * Android requires all APKs to be digitally signed with a certificate before they are installed or run. The digital signature is used to verify the owner's identity for application updates. This process can prevent an app from being tampered with or modified to include malicious code. * When an APK is signed, a public-key certificate is attached to it. This certificate uniquely associates the APK with the developer and the developer's private key. When an app is being built in debug mode, the Android SDK signs the app with a debug key created specifically for debugging purposes. An app signed with a debug key is not meant to be distributed and won't be accepted in most app stores, including the Google Play Store. * The final release build of an app must be signed with a valid release key. In Android Studio, the app can be signed manually or via creation of a signing configuration that's assigned to the release build type. * Prior Android 9 (API level 28) all app updates on Android need to be signed with the same certificate, so a validity period of 25 years or more is recommended. Apps published on Google Play must be signed with a key that that has a validity period ending after October 22th, 2033.   Three APK signing schemes are available:   * JAR signing (v1 scheme), * APK Signature Scheme v2 (v2 scheme), * APK Signature Scheme v3 (v3 scheme). * APK Signature Scheme v4 (v4 scheme). | | | | | |
| **Impact** | | | | | |
| JAR signing (v1 signature) does have additional attack vector:   * malicious mangling of the metadata, or of compressed (signed) entries.   For example, uncompressing a specially forged entry, which occurs in trusted context, could cause dangerous side-effects for your system. | | | | | |
| **Remediation** | | | | | |
| * Make sure that the release build has been signed via both the v1 and v2 schemes for Android 7.0 (API level 24) and above and via all the three schemes for Android 9 (API level 28) and above, and that the code-signing certificate in the APK belongs to the developer. | | | | | |
| **How to reproduce the Security defect** | | | | | |
| Path: C:\Users\40018089\AppData\Local\Android\Sdk\build-tools\33.0.1   * Open cmd * apksigner verify --verbose C:\Users\40018089\Desktop\APPS\app-release.apk | | | | | |
| **CVSS Base Score :** | |  | | | |
| **Evidence** | | | | | |
|  | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1. **Unsafe debug certificate** | | | | | |
| **Name** | Improper Android Code Quality with Debug certificate. | **Impact** | High | **Risk Rating** | HIGH |
| **Ease of Exploit** | Automated Tools Available | **Likelihood** | High |
| **Category** | Client Code Quality | | | | |
| **URL / Impacted System** | Android APP | | | | |
| **Description** | | | | | |
| Debug certificate is created by the build tools and is insecure by design, most app stores (including the Google Play Store) do not accept apps signed with a debug certificate for publishing. | | | | | |
| **Impact** | | | | | |
| An app signed with a debug key is not be meant for distribution and won't be accepted in most app stores, including the Google Play Store. To prepare the app for final release, the app must be signed with a release key belonging to the developer. | | | | | |
| **Remediation** | | | | | |
| * Developers need to make sure that release builds are signed with the appropriate certificate from the release keystore. In Android Studio, this can be done manually or by creating a signing configuration and assigning it to the release build type. * The signing configuration can be managed through the Android Studio GUI or the signingConfigs {} block in build.gradle. The following values need to be set to activate both v1 and v2 scheme: * v1SigningEnabled true * v2SigningEnabled true | | | | | |
| **How to reproduce the Security defect** | | | | | |
| The debug certificate does not have a trust anchor, so the certificate can easily be replaced.  Path: C:\Program Files\Java\jdk-11.0.13\bin>   * Open CMD * jarsigner -verify -verbose -certs C:\Users\40018089\Desktop\APPS\app-release.apk | | | | | |
| **CVSS Base Score :** | |  | | | |
| **Evidence** | | | | | |
|  | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1. **Vulnerable to Janus vulnerability** | | | | | |
| **Name** | Janus vulnerability | **Impact** | High | **Risk Rating** | HIGH |
| **Ease of Exploit** | Easy | **Likelihood** | High |
| **Category** | Poor Code Quality | | | | |
| **URL / Impacted System** | Android APP | | | | |
| **Description** | | | | | |
| * Janus is an Android vulnerability that allows an attacker to modify an application undetected. This is achieved by adding a malicious Dalvik executable (DEX) file to an Android Package Kit (APK) file. * Janus vulnerability (CVE-2017-13156) in Android allows attackers to modify the code in applications without affecting their signatures. The report goes onto say that the root of the vulnerability is that a file can be a valid APK file and a valid DEX file at the same time. * Application is signed with v1 signature scheme, making it vulnerable to Janus vulnerability on Android 5.0-8.0, if signed only with v1 signature scheme. Applications running on Android 5.0-7.0 **signed with v1, and v2/v3 scheme is also vulnerable.**   CVE-2017-13156 (<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-13156> ) | | | | | |
| **Impact** | | | | | |
| * An attacker can replace a trusted application with high privileges (a system app, for instance) by a modified update to abuse its permissions. * Depending on the targeted application, this could enable the hacker to access sensitive information stored on the device or even take over the device completely. * Alternatively, an attacker can pass a modified clone of a sensitive application as a legitimate update which can look and behave like the original application but inject malicious behavior. | | | | | |
| **Remediation** | | | | | |
| * Verify the schemes for the latest Android software versions and map the flag as per the requirements. * Consider whitelisting applications on corporate devices.   NOTE:   1. On device running android version 5.X & 6.X, Apk signature was verified using only v1 scheme Janus vulnerability was found in v1 signing scheme and google released a patch on 2017 December 01 to fix Janus on these android versions.so even though apk is signed only with v1 which can be run on 5.x & 6.x, if 2017 December 01 patch is installed in the device Janus cannot be exploited. 2. After releasing the patch for Janus, Google released v2 signing scheme and prioritized device to use v2 scheme over v1 scheme if v2 was used along with v1 scheme but they did not integrate the patch into the system until android 8.1.This made possible to exploit Janus on android 7.x & 8.0 also, as long as 2017 December patch is not installed. 3. Applications signed with v2 or v3 along with v1 are also vulnerable to Janus if they are made to run on android versions 5.x & 6.x as it verifies only v1 scheme without installing the patch. 4. Finally v1 scheme was removed and V4 scheme has been introduced in Android 11, Applications that are only signed with v1 scheme will not run on Android 11 | | | | | |
| **How to reproduce the Security defect** | | | | | |
| --steps-- | | | | | |
| **CVSS Base Score :** | |  | | | |
| **Evidence** | | | | | |
|  | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1. **Improper utilization of Android attribute** | | | | | |
| **Name** | Android Backup Attribute Missing | **Impact** | Low | **Risk Rating** | HIGH |
| **Ease of Exploit** | Easy | **Likelihood** | High |
| **Category** | Poor Code Quality | | | | |
| **URL / Impacted System** | Android APP | | | | |
| **Description** | | | | | |
| The android:allowBackup attribute defines whether application data can be backed up and restored by a user who has enabled usb debugging. If backup flag is set to true, it allows an attacker to take the backup of the application data via adb even if the device is not rooted.  CVE-2017-16835 (<https://nvd.nist.gov/vuln/detail/CVE-2017-16835> ) | | | | | |
| **Impact** | | | | | |
| * The flag [android:allowBackup] (flag is missing) by default it is set to true and allows anyone to backup your application data via adb. * It allows users who have enabled USB debugging to copy application data off of the device. | | | | | |
| **Remediation** | | | | | |
| * Add [android:allowBackup ] attributes in the AndroidManifest.xml file and mark it as false. | | | | | |
| **How to reproduce the Security defect** | | | | | |
| --steps-- | | | | | |
| **CVSS Base Score :** | |  | | | |
| **Evidence** | | | | | |
|  | | | | | |

**Table of Findings**

|  |  |  |
| --- | --- | --- |
| **S.no** | **Vulnerabilities** | **Severity** |
| 1 | Improper Export of Android Application Components | HIGH |
| 2 | Encryption of Security-Relevant Inputs without Integrity Checking | HIGH |
| 3 | Android Code with improper APK signing schemes | MEDIUM |
| 4 | Unsafe debug certificate | HIGH |
| 5 | Janus vulnerability | HIGH |
| 6 | Android Backup Attribute Missing | HIGH |
|  |  |  |

MOBSF report: 