



{Client Name}

{Date}

{Client/Application Name} Android Application Security Assessment

Client Details

Company Name: {Client Name}

Contact Person: {Person Name}

Address: {Client Address}

Email: {Email Address}

Telephone: {Telephone Number}

Document History

| Version | Date | Author | Remark |
|---------|--------|----------|-------------------|
| 1.0 | {Date} | {Author} | Document Creation |
| | | | |
| | | | |

Table of Contents

| | |
|--|----|
| 1. About Payatu | 4 |
| 2. Project Details | 5 |
| 2.1 Executive Summary | 5 |
| 2.2 Scope and Objective | 5 |
| 2.3 Project Timeline | 5 |
| 2.4 Technological Impact Summary | 6 |
| 2.5 Business Impact Summary | 6 |
| 2.6 Testing Environment | 6 |
| 2.7 Vulnerability Chart | 8 |
| 2.8 Vulnerabilities by Category | 9 |
| 2.9 Table of Findings | 10 |
| 2.10 Application Strengths | 10 |
| 2.11 Application Weaknesses | 11 |
| 3. Technical Findings | 12 |
| 3.1 PY-CL-001: Application logs user credentials into the device logs | 12 |
| 3.2 PY-CL-002: Application contains hardcoded secrets in the application source code | 14 |
| 3.3 PY-CL-003: Application uses broken cryptography for storing user credentials in the shared storage | 16 |
| 3.4 PY-CL-004: Application source code is missing obfuscation | 19 |
| 3.5 PY-CL-005: Application has not implemented SSL Pinning | 21 |
| 3.6 PY-CL-006: Application has Android backup functionality enabled | 24 |
| 3.7 PY-CL-007: Application can be installed on a rooted device | 26 |
| 4. We Prescribe | 28 |
| 5. Appendix | 30 |
| 5.1 Observations | 30 |
| 5.1.1 Android Permissions mentioned in AndroidManifest.xml | 30 |
| 5.2 References | 32 |
| 5.3 Failed Test Cases | 33 |
| 5.3.1 Attacking Android Intents | 33 |
| 5.3.2 SQL injection attacks at Client Side | 35 |
| 5.3.3 Cross-Site Scripting (XSS) attacks at Client Side | 37 |

1. About Payatu

Payatu is a research-focused security testing service organization specialized in IoT and embedded products, web, mobile, cloud & infrastructure security assessments, and in-depth technical security training. Our state-of-the-art research, methodologies, and tools ensure the safety of our client's assets.

At Payatu, we believe in following one's passion, and with that thought, we have created a world-class team of researchers and executors who are bending the rules to provide the best security services. We are a passionate bunch of folks working on the latest and leading-edge security technology.

We are proud to be part of a vibrant security community and don't miss any opportunity to give back. Some of the contributions in the following fields reflect our dedication and passion.

- nullcon - nullcon security conference is an annual security event held in Goa, India. After years of efforts put in the event, it has now become a world-renowned platform to showcase the latest and undisclosed research.
- hardware.io - Hardware security conference is an annual hardware security event held in The Hague, Netherlands. It is being organized to answer emerging threats and attacks on hardware. We aim to make it the largest platform, where hardware security innovation happens.
- Dedicated fuzzing infrastructure - We are proud to be one of the few security research companies to own an in-house infrastructure and hardware for distributed fuzzing of software such as browsers, client and server applications.
- null - It all started with null - The open security community. It's a registered non-profit society and one of the most active security community. null is driven totally by passionate volunteers.
- Open source - Our team regularly authors open source tools to aid in security learning and research.
- Talks and Training: Our team delivers talks/highly technical training in various international security and hacking conference, i.e., DEFCON Las Vegas, BlackHat Las Vegas, HITB Amsterdam, Consecwest Vancouver, nullcon Goa, HackinParis Paris, Brucon Belgium, zer0con Seoul, PoC Seoul to name few.

We are catering to a diverse portfolio of clients across the world who are leaders in banking, finance, technology, healthcare, manufacturing, media houses, information security, and education, including government agencies. Having various empanelment and accreditations, along with a strong word of mouth has helped us win new customers, and our thorough professionalism and quality of work, have brought repeat business from our existing clients.

We thank you for considering our security services and requesting a proposal. We look forward to extending the expertise of our passionate, world-class professionals to achieve your security objectives.

2. Project Details



2.1 Executive Summary

Security Assessment of {Client Name} Android application has been performed, considering below common security issues:

- ✓ If proper access control is implemented across teams.
- ✓ If the proper Authorization & Authentication System is implemented.
- ✓ If proper error handling is done to avoid exposing API Keys.
- ✓ If the user input is properly escaped.

Overall security postures of the application are moderate. However, some of the security controls/measures have not been properly thought of/implemented during the design and coding of the application; therefore, exploitation is not very likely due to randomness in identifiers.

The security assessment revealed 0 critical severity security issues, 3 high severity security issues, 2 medium severity issues, and 2 low severity issues in this application. The consolidated summary of the assessment has been presented in the Executive Summary section. Additional information is contained within the Detailed Vulnerability Information section of this report.



2.2 Scope and Objective

The scope of this assessment was limited to Android applications created by {Client Name}. The application version under review was {Version Number X.Y.Z} available for download from Google Play Store at

- {Google Play Store Application URL }



2.3 Project Timeline

The security assessment was performed for {Number of days} days from {Start Date} to {End Date}.



2.4 Technological Impact Summary

The Payatu security team performs real-time security assessments on the mobile application. These assessments aim is to uncover any security issues in the assessed mobile application, explain the impact and risks associated with the found issues, and provide guidance in the prioritization and remediation steps.

It was identified that the mobile application logs user credentials in the device logs. The developer has hardcoded cryptographic key, IV, and mode in the source code. The attacker uses this hardcoded information to decrypt the user credentials stored in the application shared storage. The application has not implemented any obfuscation layer in the application source code. The application also doesn't implement SSL pinning to prevent MITM attacks. The mobile application has Android backup functionality enabled. The mobile application can be installed on the rooted device.



2.5 Business Impact Summary

We identified the following business impacts:

- ▶ An attacker can take over the user account by just accessing the android device logs.
- ▶ An attacker can access the hardcoded secret keys and can use it to get decrypted user credentials.
- ▶ An attacker can perform a MITM attack and can sniff the traffic between the mobile device and the server.



2.6 Testing Environment

To perform {Client} android application assessment, we have set up our testing environment by either using a rooted device or using any Android Emulator.

Detail instructions for Rooting Android Device can be found at:

<https://www.xda-developers.com/root/>

There are many android emulators like Android Studio, Genymotion, etc. Details instruction for setting up and download emulators can be found at:

- <https://www.genymotion.com/download/>

- <https://docs.expo.io/workflow/android-studio-emulator/>

We have also set up BurpSuite proxy tool, which intercepts the HTTP/HTTPS traffic originated from browser to the server



Detailed instruction to configure BurpSuite proxy tool in android:

<https://portswigger.net/support/configuring-an-android-device-to-work-with-burp>

Apart from Burp Suite, we have set-up a few more tools like Android Debug Bridge (ADB), Drozer, Frida, Objection, Dex2jar, and Jdgui.

The details instructions to configure and setup the tools used can be found in the below links:

- Android Debug Bridge(adb) :
 - <https://www.xda-developers.com/install-adb-windows-macos-linux/>
- Drozer:
 - <https://github.com/FSecureLABS/drozer>
- Dex2jar:
 - <https://github.com/pxb1988/dex2jar>
- Jd-GUI:
 - <http://java-decompiler.github.io/>
- Frida:
 - <https://frida.re/docs/android/>
- Objection:
 - <https://github.com/sensepost/objection>

For all the mentioned reproduction steps in this report, we assume the Burp Suite and Android Debug Bridge testing environment setup is done. Also, we assume all the testing is done in the rooted device or an Android device emulator.



2.7 Vulnerability Chart

The discovered vulnerabilities table and chart illustrated below provides a snapshot view of the number and severity of issues discovered during this security assessment.

CRITICAL

This issue can impact the application severely and should be addressed immediately. Attackers can gain root or superuser access or severely impact system operation.

HIGH

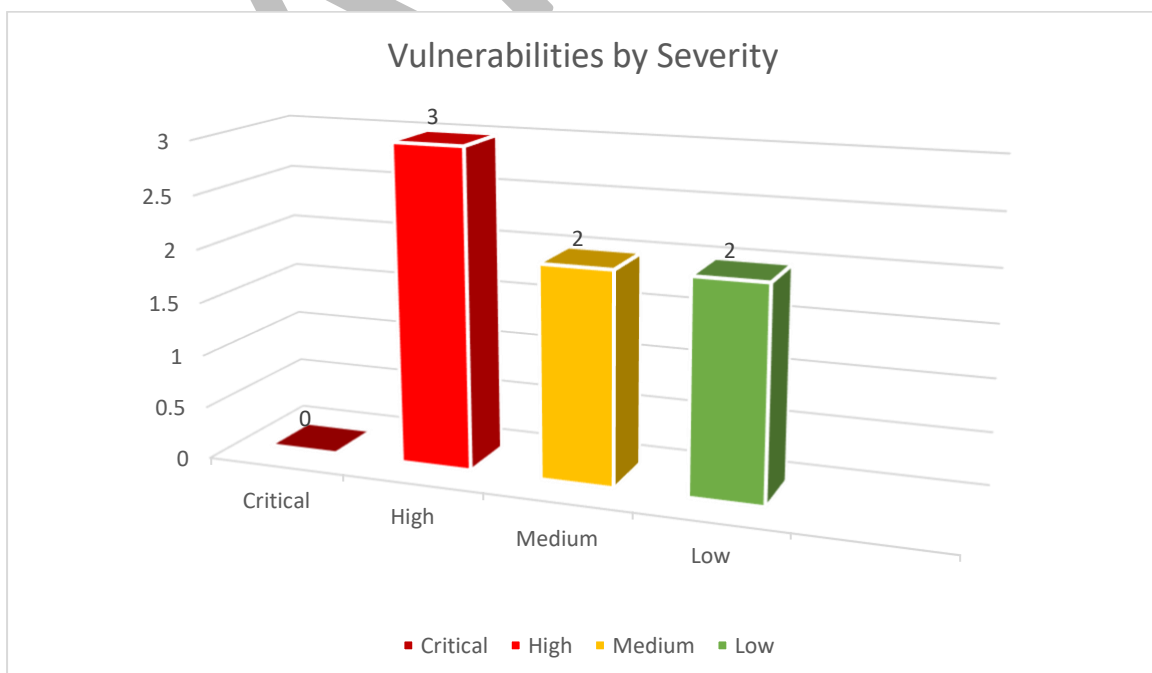
This issue can cause a problem like unprivileged access and should be addressed as soon as possible.

MEDIUM

This issue may pose a significant threat over a longer period of time.

LOW

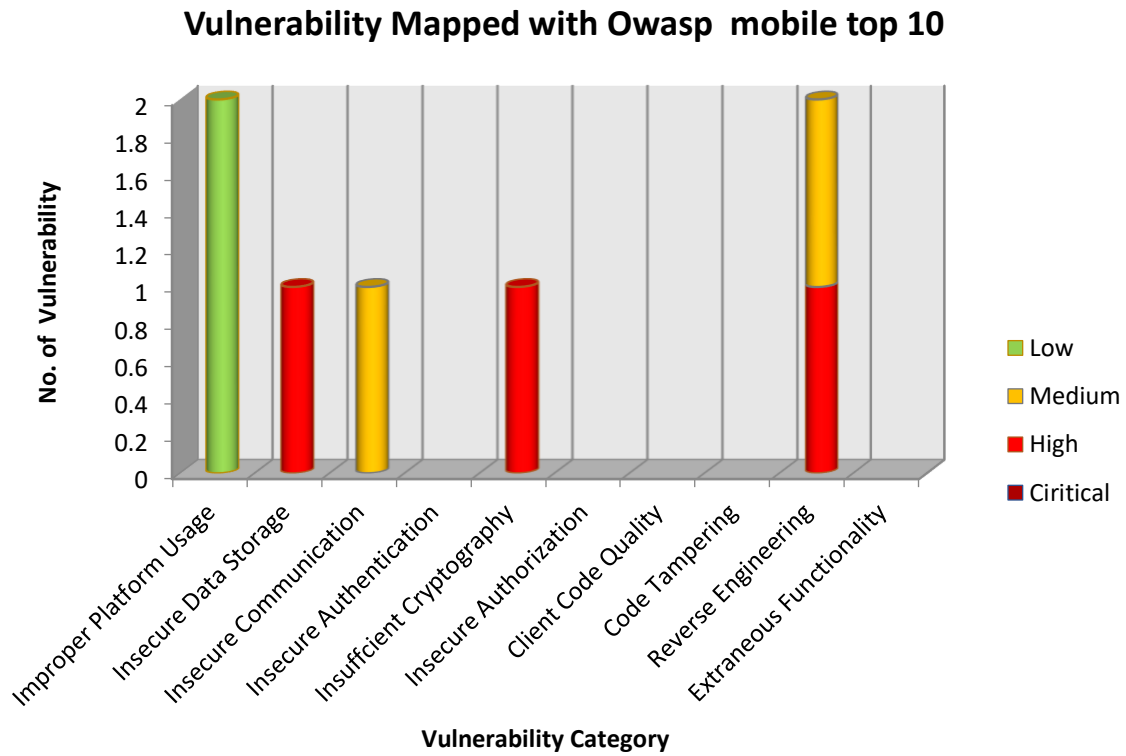
This issue is more likely an information disclosure and may be an acceptable threat.





2.8 Vulnerabilities by Category

Below given chart shows the vulnerability matrix based on the category of vulnerabilities.



SA



2.9 Table of Findings

| Vuln ID | Finding | Severity | Status |
|------------|---|----------|-----------|
| PY-CL-001 | Application logs user credentials into device logs | HIGH | Not Fixed |
| PY-CL-002 | Application contains Hardcoded secrets in the application source code | HIGH | Not Fixed |
| PY-CL-003 | Application uses broken cryptography for storing user credentials in the shared storage | HIGH | Not Fixed |
| PY-CL-004 | Application source code is missing obfuscation | MEDIUM | Not Fixed |
| PY-CL -005 | Application has not implemented SSL Pinning | MEDIUM | Not Fixed |
| PY-CL-006 | Application has Android backup functionality enabled | LOW | Not Fixed |
| PY-CL-007 | Application can be installed on a rooted device | LOW | Not Fixed |



2.10 Application Strengths

During our assessment, we observed the following properties of the application that are well designed and serve towards its strengths:

- ✓ The application implements strong authorization and authentication checks through JWT tokens that cannot be tampered. This reduces the risk of access control violations, privilege escalation attacks as well as CSRF attacks.
- ✓ The application implements appropriate user input sanitization and output encoding that reduces the risk of server-side injection attacks as well as XSS attacks.
- ✓ The application does not leak any sensitive data via unhandled error messages
- ✓ During our testing, we did not find any session management issues. The application allocates new JWT tokens every time and immediately expires any JWT token after logout. The application also has provision to expire a session after a certain period of inactivity
- ✓ The application does not use any 3rd party open source component that is affected by any known vulnerabilities.



2.11 Application Weaknesses

- ▶ The application fails to store user credentials using a proper cryptography algorithm.
- ▶ The application fails to obfuscate the application source code.
- ▶ The application has hardcoded secrets in the application source code.
- ▶ The application logs user credentials in the device logs.
- ▶ The application has not implemented SSL Pinning to protect the network traffic.
- ▶ The application has Android backup functionality enabled.
- ▶ The application can be installed on the rooted device.

These vulnerabilities were identified and verified by Payatu during the process of this Android Application Penetration Test for the client. Retesting should be planned following the remediation of these vulnerabilities.

3. Technical Findings

3.1 PY-CL-001: Application logs user credentials into the device logs

Potential Impact: **HIGH**

Description:

During our assessment, we observed that the application logs sensitive information like the username and password in device logs.

Affected Hosts:

- {Client/App Name} Android application
 - Android Device Logs

CVSS Score:

CVSS Base Score: 7.8
CVSS Temporal Score: 7.5
CVSS Environmental Score: 7.5

CVSS Vector:

CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H/E:H/RL:O/RC:C/CR:H/IR:H/AR:H/MAV:L/MAC:L/MPR:N/MUI:R/MS:U/MC:H/MI:H/MA:H

Business Risk:

- Loss of Customer Data
- Reputation Damage
- Fraud
- Material Loss

Technical Risk: Successful exploitation of this vulnerability allows an attacker to get access to sensitive data like user credentials leading to user account takeover by just accessing the android device logs.

Remediation:

- Do not write sensitive data in the application log.
- Check source code for the below keywords and remove any occurrence that might log sensitive data in the logs.
 - Log, Logger classes
 - Log.d, Log.e, Log.i ... functions


- System.out.print, System.err.print functions
 - printStackTrace
-

Steps to Reproduce:

1. Connect the device to the PC using the USB cable.
2. We will be using the Android inbuilt command line tool – logcat – to view android device logs.

Cmd: adb logcat

3. Start the application and enter the credentials into the application.
4. Observe the logs, the application has logged user credentials into the device logs.



```
07-08 16:37:17.442 11674 12152 D Successful Login:: , account=dinesh:Dinesh@123$
```

The screenshot shows a logcat output with a red box highlighting the message: "Successful Login:: , account=dinesh:Dinesh@123\$". The background of the logcat window is dark, and the text is white. A large, light gray "SAMPLE" watermark is visible diagonally across the page.

3.2 PY-CL-002: Application contains hardcoded secrets in the application source code

Potential Impact: **HIGH**

Description:

During our assessment, we observed that the developer had written some hardcoded secrets in the application source code. The attacker can reverse engineer the application package and access this hardcoded secret.

Affected Hosts:

- {Client/App Name} Android application
 - Android application source code(apk file)

CVSS Score:

CVSS Base Score: 8.0
CVSS Temporal Score: 7.2
CVSS Environmental Score: 7.2

CVSS Vector:

CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:U/C:H/I:H/A:H/E:P/RL:O/RC:C/CR:H/IR:H/AR:H/MAV:N/MAC:L/MPR:L/MUI:R/MS:U/MC:H/MI:H/MA:H

Business Risk:

- Intellectual Property theft;
- Reputational Damage;
- Identity Theft; or
- Compromise of Backend Systems

Technical Risk: Successful exploitation of this vulnerability can lead to the exposure of resources or functionality to unintended actors, possibly providing attackers with sensitive information or even execute arbitrary code.

Remediation:

- The passwords or keys should be limited at the back end to only performing actions valid for the front end, as opposed to having full access.
- Developers should review the contents of all the files present in the application package for any other sensitive data being hardcoded in the files and should remove them from the application package.

Steps to Reproduce:

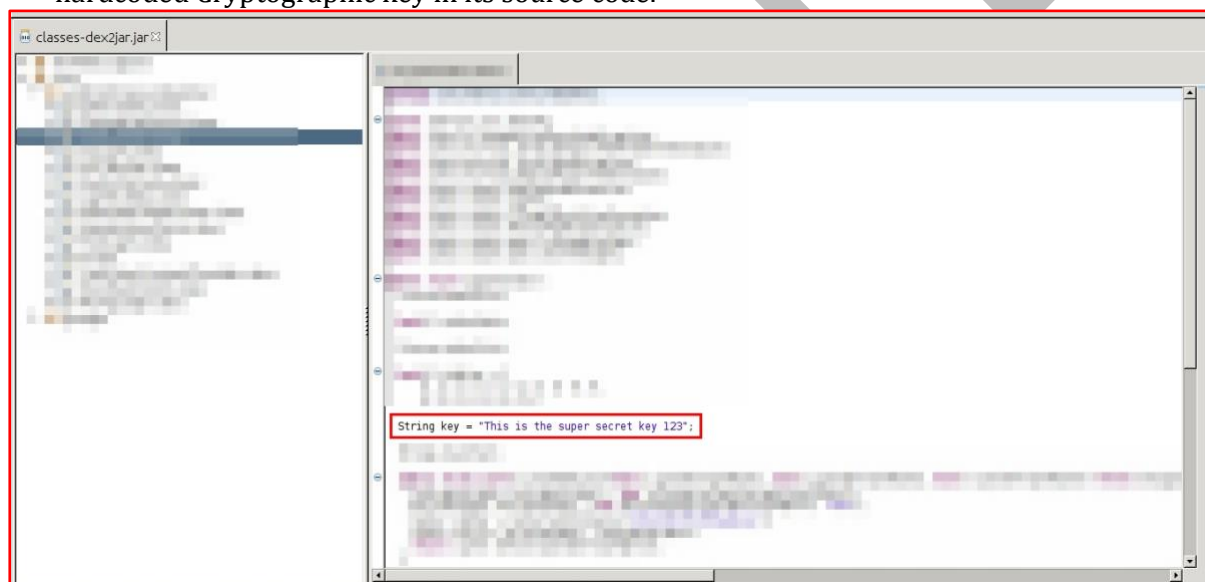
1. Make sure we have .apk file.
2. If the application has been downloaded from the Google Play store, we can get the .apk file by pulling it from the device using the Android inbuilt command line tool- pull – to transfer files from the mobile device to PC. Make sure the device is connected to the PC using the USB cable.

Cmd: adb pull {path-to-apk-file}

3. Unzip the application (.apk) file.
4. Open the terminal and navigate to the path of the extracted folder. This extracted folder might contain classes.dex file.
5. Using d2j-dex2jar, convert the classes.dex file from dex to jar.

Cmd: d2j-dex2jar.sh classes.dex

6. Now use Jd-gui to open the converted jar file. This jar file will contain the application Java pseudo-code.
7. Open class file {Name-of-class}. On viewing the code, it is visible that the application has a hardcoded Cryptographic key in its source code.



3.3 PY-CL-003: Application uses broken cryptography for storing user credentials in the shared storage

Potential Impact: **HIGH**

Description:

During our assessment, we observed that the application is using weak cryptography in the application for encrypting and decrypting the user credentials (username and password) to store it in the shared storage of the device. The application is using AES-256-CBC encryption and decryption algorithm. The developer has hardcoded the key and IV used to encrypt the username and the password. Using this information, the attacker can easily crack the encrypted user credentials stored in the shared storage.

Affected Hosts:

- {Client/App Name} Android application
 - Android application source code(apk file)

CVSS Score:

CVSS Base Score: 7.3
 CVSS Temporal Score: 6.6
 CVSS Environmental Score: 6.6

CVSS Vector:

CVSS:3.1/AV:L/AC:L/PR:L/UI:R/S:U/C:H/I:H/A:H/E:P/RL:O/RC:C/CR:H/IR:H/AR:H/MAV:L/MAC:L/MPR:L/MUI:R/MS:U/MC:H/MI:H/MA:H

Business Risk:

- Privacy Violations;
- Information Theft;
- Code Theft;
- Intellectual Property Theft; or
- Reputational Damage.

Technical Risk: Successful exploitation of this vulnerability will result in the unauthorized retrieval of sensitive information from the mobile device.

Remediation:

- Avoid the storage of any sensitive data on a mobile device where possible.
- Do not store cryptographic keys on the client-side (application source code).
- Use secure one-way hashing algorithms.
- Do not encrypt and decrypt the user credentials on the client-side.

- Apply cryptographic standards that will withstand the test of time for at least 10 years into the future; and
- Follow the NIST guidelines on the recommended algorithms.
 - <https://csrc.nist.gov/csrc/media/publications/sp/800-175b/archive/2016-03-11/documents/sp800-175b-draft.pdf>

Steps to Reproduce:

1. Launch the application in the mobile device and perform the login functionality.
2. Connect the android device to PC using the USB cable.
3. Open terminal on the PC. We will be using the Android inbuilt command line tool – shell - to get the shell terminal of the device.

Cmd: adb shell

4. Now in the shell enter command – su – to get the superuser access.

Cmd: su

5. Navigate to the application sandbox by entering below command

Cmd: cd /data/data/{application-package-name}/

6. Navigate to the shared_prefs folder, which is used to store a small amount of primitive data as key/value pairs in the file on the device.

Cmd: cd shared_prefs/

7. Open the file {File-name}. The following screenshot shows that the username and the password were stored in encrypted format in the file. Note the value of the "superSecurePassword".



```
<map>
  <string name="superSecurePassword">DTw2VXjSoFdq0e61fHxJg== </string>
  <string name="EncryptedUsername">ZGluZXNo </string>
</map>
```

8. Make sure we have .apk file.
9. If the application has been downloaded from the Google Play store, we can get the .apk file by pulling it from the device using the Android inbuilt command line tool- pull – to transfer files from the mobile device to PC. Make sure the device is connected to the PC using the USB cable.

Cmd: adb pull {path-to-apk-file}

10. Unzip the application (.apk) file.
11. Open the terminal and navigate to the path of the extracted folder. This extracted folder might contain classes.dex file.
12. Using d2j-dex2jar, convert the classes.dex file from dex to jar.

Cmd: d2j-dex2jar.sh classes.dex

13. Now use Jd-gui to open the converted jar file. This jar file will contain the application Java pseudo-code.

14. Open class file {Name-of-class}. On viewing the code, we can get the values of key, IV, and ciphertext to reverse the encryption.

```
byte[] ivBytes = new byte[] {(byte) 0, (byte) 0, (byte) 0, (byte) 0, (byte) 0, (byte) 0,
String key = "This is the super secret key 123";
String plaintext;

public static byte[] aes256encrypt(byte[] ivBytes, byte[] keyBytes, byte[] textBytes) {
    AlgorithmParameterSpec ivSpec = new IvParameterSpec(ivBytes);
    SecretKeySpec newKey = new SecretKeySpec(keyBytes, "AES");
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(1, newKey, ivSpec);
    return cipher.doFinal(textBytes);
}
```

15. Now we can any third party tool or online tool to decrypt this password.

AES Online Decryption

Enter text to be Decrypted

Input Text Format: ☒ Base64 ☐ Hex

Select Mode

CBC

Enter IV Used During Encryption(Optional)

Key Size in Bits

256

Enter Secret Key

This is the super secret key 123

Decrypt

AES Decrypted Output (Base64):

Decode to Plain Text

D

16. So, we have successfully decrypted the user password.

3.4 PY-CL-004: Application source code is missing obfuscation

Potential Impact: MEDIUM

Description:

During our assessment, we observed that the application does not implement any binary protection, and it's trivially possible to reverse engineer and obtain the application source code. Having access to source code allows an adversary to understand the inner working of the application and study potentially hidden functions as well. By reversing the application source code, we were able to understand the application logic and also the hardcoded secrets in the application.

Affected Hosts:

- {Client/App Name} Android application
 - Android application source code(apk file)

CVSS Score:

CVSS Base Score: 6.6
CVSS Temporal Score: 5.9
CVSS Environmental Score: 5.9

CVSS Vector:

CVSS:3.1/AV:P/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H/E:P/RL:O/RC:C/CR:H/IR:H/AR:H/MAV:P/MAC:L/MPR:L/MUI:N/MS:U/MC:H/MI:H/MA:H

Business Risk:

- Intellectual Property theft;
- Reputational Damage;
- Identity Theft; or
- Compromise of Backend Systems.

Technical Risk: An attacker may exploit reverse engineering to achieve any of the following:

- Reveal information about back end servers;
- Reveal cryptographic constants and ciphers;
- Steal intellectual property;
- Perform attacks against back end systems; or
- Gain intelligence needed to perform subsequent code modification.

Remediation:

- Use of obfuscation tools to obfuscate the application source code is recommended.
- Below reference can be used:
 - <https://developer.android.com/studio/build/shrink-code>
- A good obfuscator will have the following abilities:
 - Narrow down what methods/code segments to obfuscate;
 - Tune the degree of obfuscation to balance performance impact;
 - Withstand de-obfuscation from tools like IDA Pro and Hopper;
 - Obfuscate string tables as well as methods

Steps to Reproduce:

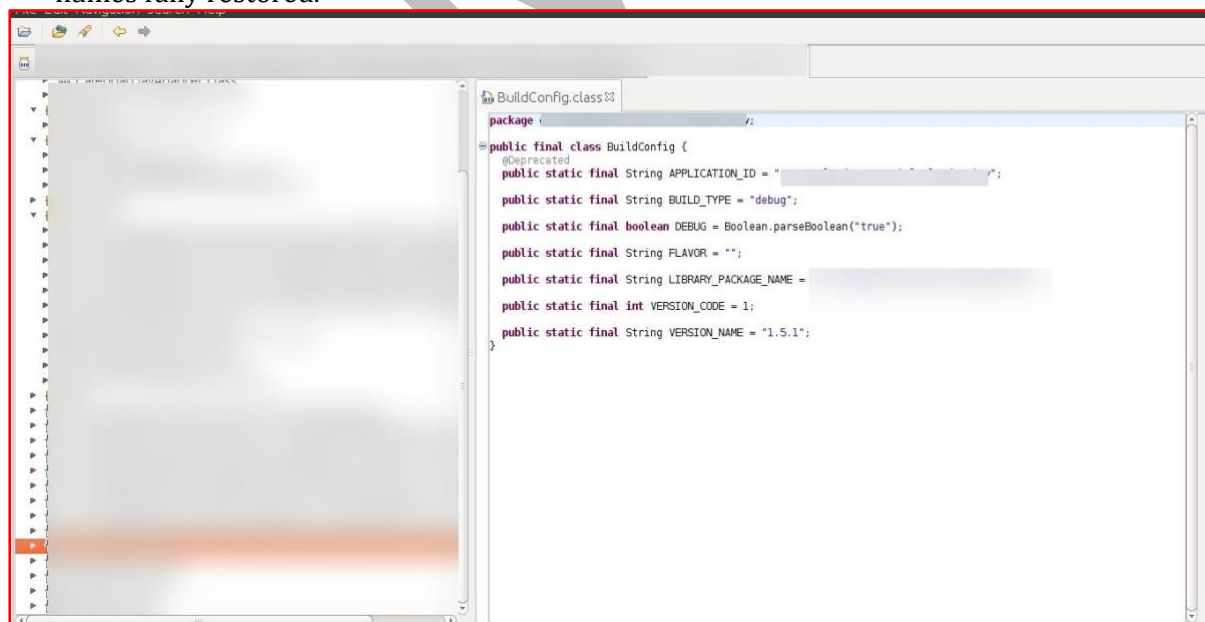
1. Make sure we have .apk file.
2. If the application has been downloaded from the Google Play store, we can get the .apk file by pulling it from the device using the Android inbuilt command line tool- pull – to transfer files from the mobile device to PC. Make sure the device is connected to the PC using the USB cable.

Cmd: adb pull {path-to-apk-file}

3. Unzip the application (.apk) file.
4. Open the terminal and navigate to the path of the extracted folder. This extracted folder might contain classes.dex file.
5. Using d2j-dex2jar, convert the classes.dex file from dex to jar.

Cmd: d2j-dex2jar.sh classes.dex

6. Now use Jd-gui to open the converted jar file. This jar file will contain the application Java pseudo-code.
7. We can observe that tool has the decompiled java source code list with file and function names fully restored.



3.5 PY-CL-005: Application has not implemented SSL Pinning

Potential Impact: MEDIUM

Description:

During our assessment, we observed that the application had not implemented SSL Pinning. So it was possible to intercept the traffic of the application when we configure proxy into the mobile device, and the application does not show any error or message in the application.

Affected Hosts:

- {Client/App Name} Android application
 - Application Data

CVSS Score:

CVSS Base Score: 6.1
CVSS Temporal Score: 5.5
CVSS Environmental Score: 6.1

CVSS Vector:

CVSS:3.1/AV:N/AC:L/PR:H/UI:R/S:U/C:H/I:H/A:N/E:P/RL:O/RC:C/CR:H/IR:H/MAV:N/MAC:L/MPR:H/MUI:R/MS:U/MC:H/Mi:H/MA:N

Business Risk: The interception of sensitive data through a communication channel will result in a privacy violation. The violation of a user's confidentiality may result in:

- Identity theft;
- Fraud, or
- Reputational Damage.

Technical Risk: Successful exploitation of this vulnerability will allow the attacker to perform a MITM attack and analyze the traffic.

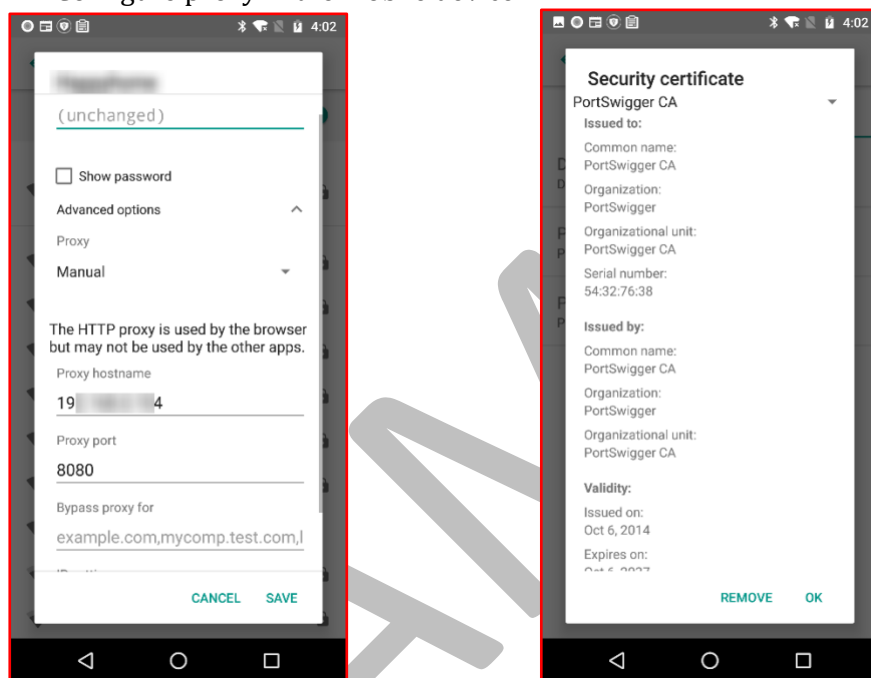
Remediation:

- Use strong, industry-standard cipher suites with appropriate key lengths.
- Use certificates signed by a trusted CA provider.
- Never allow self-signed certificates and consider certificate pinning for security-conscious applications.
- Always require SSL chain verification.
- Only establish a secure connection after verifying the identity of the endpoint server using trusted certificates in the key chain.
- Alert users through the UI if the mobile app detects an invalid certificate.

- If possible, apply a separate layer of encryption to any sensitive data before it is given to the SSL channel. If future vulnerabilities are discovered in the SSL implementation, the encrypted data will provide a secondary defense against confidentiality violation.
- References on implementing a fix for this vulnerability
 - <https://developer.android.com/training/articles/security-config>
 - <https://medium.com/@appmattus/android-security-ssl-pinning-1db8acb6621e>

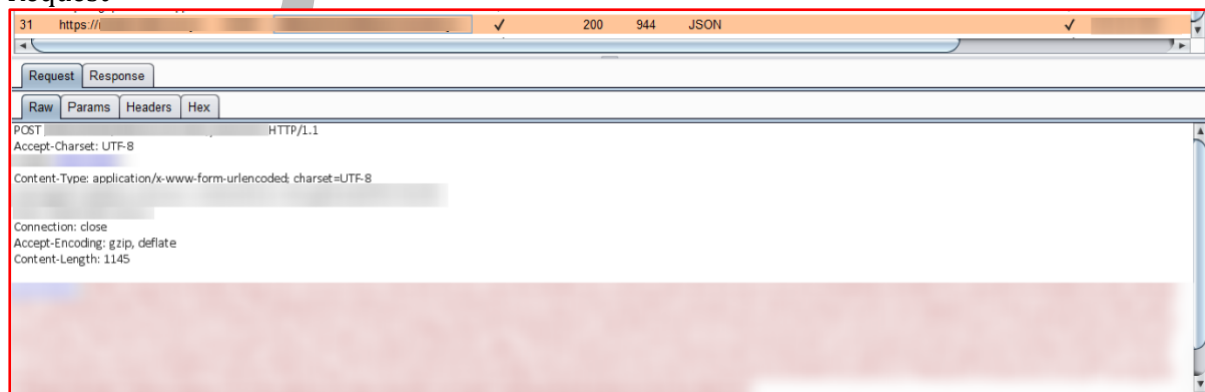
Steps to Reproduce:

1. Configure proxy in the mobile device.

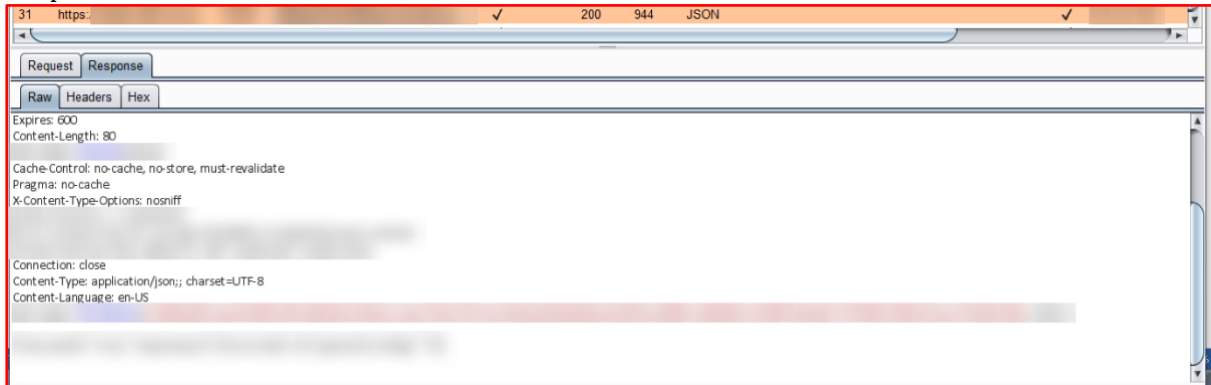


2. Launch the application and go through the application functionalities.
3. Now open burp suite and observe the traffic.

Request



Response



3.6 PY-CL-006: Application has Android backup functionality enabled

Potential Impact: LOW

Description:

During our assessment, we observed that the application had set `android:allowBackup=true` in the `AndroidManifest.xml`. The flag `[android:allowBackup]` should be set to false. It allows anyone to back up the application data via adb. It allows users who have enabled USB debugging to copy application data off of the device. If there is sensitive data and an attacker gets physical access to the device, then the data can be backed up by the attacker. So, according to security standards, it should be set to false.

Affected Hosts:

- {Client/App Name} Android application
 - Application Data

CVSS Score:

CVSS Base Score: 2.9
 CVSS Temporal Score: 2.6
 CVSS Environmental Score: 1.6

CVSS Vector:

CVSS:3.1/AV:L/AC:H/PR:H/UI:R/S:U/C:L/I:L/A:N/E:P/RL:O/RC:C/CR:L/IR:L/AR:L/MAV:L/MAC:H/MPR:H/MUI:R/MS:U/MC:L/MI:L/MA:N

Business Risk: This vulnerability can have several different business impacts.

- Privacy Violations;
- Information Theft;
- Intellectual Property Theft; or
- Reputational Damage.

Technical Risk: If there is sensitive data and an attacker gets physical access to the device, then the data can be backed up by the attacker.

Remediation:

- Set `android:allowBackup` to false in the Android Manifest file.
- Do not store any sensitive data in the shared storage.

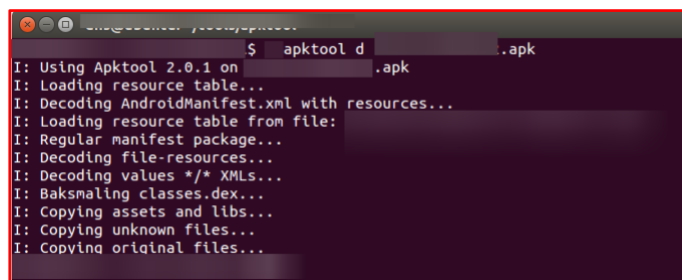
Steps to Reproduce:

1. Make sure we have .apk file.
2. If the application has been downloaded from the Google Play store, we can get the .apk file by pulling it from the device using the Android inbuilt command line tool- pull – to transfer files from the mobile device to PC. Make sure the device is connected to the PC using the USB cable.

Cmd: adb pull {path-to-apk-file}

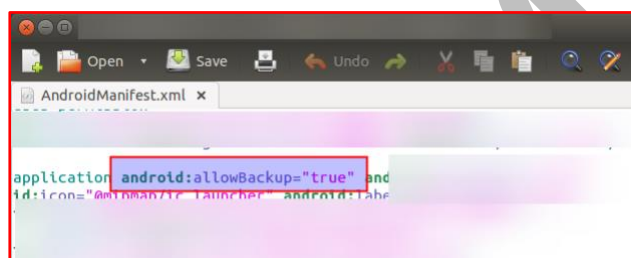
3. Open the terminal and using apktool decompile the .apk file.

Cmd: apktool d {Application-apk-name}.apk



```
$ apktool d [redacted].apk
I: Using Apktool 2.0.1 on [redacted].apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file:
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

4. Navigate to the folder created after the de-compilation of the .apk file.
5. Open the decompiled AndroidManifest.xml file. In this file, we can see that android:allowBackup flag is set to true.



```
application android:allowBackup="true" and
id:icon="@mipmap/ic_launcher" android:label
```

3.7 PY-CL-007: Application can be installed on a rooted device

Potential Impact: LOW

Description:

During our assessment, we observed that the application can be installed in the rooted device. Also, the application is running in the device without prompting any warning or error message to the user.

Affected Hosts:

- {Client/App Name} Android application

CVSS Score:

CVSS Base Score: 2.7
CVSS Temporal Score: 2.5
CVSS Environmental Score: 1.4

CVSS Vector:

CVSS:3.1/AV:P/AC:H/PR:H/UI:R/S:U/C:L/I:L/A:N/E:P/RL:O/RC:C/CR:L/IR:L/AR:L/MAV:P/MAC:H/MPR:H/MUI:R/MS:U/MC:L/MI:L/MA:N

Business Risk: An attacker can leverage this to attack the infrastructure or the application itself. This can also facilitate in intellectual property theft.

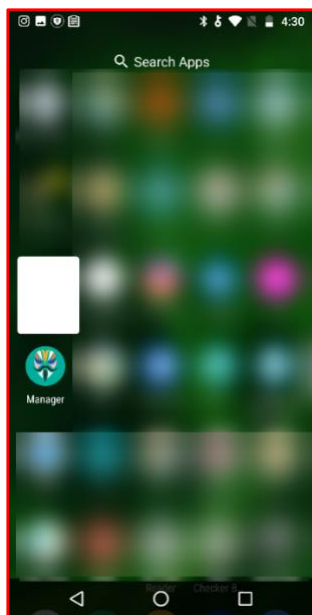
Technical Risk: An attacker can use a rooted device to investigate the application by reversing the application and modifying control-flow. This also allows an attacker to view the application sandbox.

Remediation:

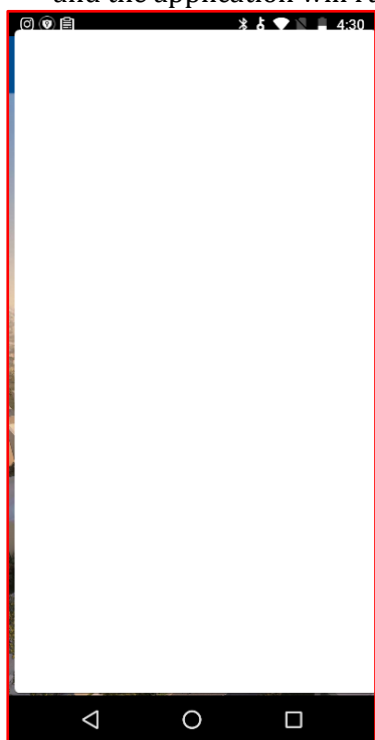
- The application should check whether the device is rooted or not, and if rooted, the application should warn the user about it or should not run the app.
- References on implementing a fix for this vulnerability:
 - <https://stackoverflow.com/a/8097801>
 - <https://github.com/scottyab/rootbeer>

Steps to Reproduce:

1. Install the application in the rooted device.



2. Now open the application. There will be no warning/error message prompted to the user and the application will run smoothly.



4. We Prescribe

The below table provides an at-a-glance view of the remediation solution and the best practices that should be taken into consideration to improve the security posture of the scoped-in application:

| Vuln ID | Recommendation & Best Practices |
|-----------|---|
| PY-CL-001 | <ul style="list-style-type: none"> Do not write sensitive data in the application log. Check source code for below keywords and remove any occurrence which might be logging sensitive data in the logs. <ul style="list-style-type: none"> Log, Logger classes Log.d, Log.e, Log.i ... functions System.out.print, System.err.print functions printStackTrace |
| PY-CL-002 | <ul style="list-style-type: none"> The passwords or keys should be limited at the back end to only performing actions valid for the front end, as opposed to having full access. Developers should review the contents of all the files present in the application package for any other sensitive data being hardcoded in the files and should remove them from the application package. |
| PY-CL-003 | <ul style="list-style-type: none"> Avoid the storage of any sensitive data on a mobile device where possible. Do not store cryptographic keys on the client-side (application source code). Use secure one-way hashing algorithms. Do not encrypt and decrypt the user credentials on the client-side. Apply cryptographic standards that will withstand the test of time for at least 10 years into the future; and Follow the NIST guidelines on the recommended algorithms. <ul style="list-style-type: none"> https://csrc.nist.gov/csrc/media/publications/sp/800-175b/archive/2016-03-11/documents/sp800-175b-draft.pdf |
| PY-CL-004 | <ul style="list-style-type: none"> Use of obfuscation tools to obfuscate the application source code is recommended. Below reference can be used: <ul style="list-style-type: none"> https://developer.android.com/studio/build/shrink-code A good obfuscator will have the following abilities: <ul style="list-style-type: none"> Narrow down what methods/code segments to obfuscate; Tune the degree of obfuscation to balance performance impact; Withstand de-obfuscation from tools like IDA Pro and Hopper; |

| | |
|------------------|--|
| | <ul style="list-style-type: none"> - Obfuscate string tables as well as methods |
| PY-CL-005 | <ul style="list-style-type: none"> • Use strong, industry-standard cipher suites with appropriate key lengths. • Use certificates signed by a trusted CA provider. • Never allow self-signed certificates, and consider certificate pinning for security-conscious applications. • Always require SSL chain verification. • Only establish a secure connection after verifying the identity of the endpoint server using trusted certificates in the key chain. • Alert users through the UI if the mobile app detects an invalid certificate. • If possible, apply a separate layer of encryption to any sensitive data before it is given to the SSL channel. If future vulnerabilities are discovered in the SSL implementation, the encrypted data will provide a secondary defense against confidentiality violation. • References on implementing a fix for this vulnerability <ul style="list-style-type: none"> - https://developer.android.com/training/articles/security-config - https://medium.com/@appmattus/android-security-ssl-pinning-1db8acb6621e |
| PY-CL-006 | <ul style="list-style-type: none"> • Set android:allowBackup to false in the Android Manifest file. • Do not store any sensitive data in the shared storage. |
| PY-CL-007 | <ul style="list-style-type: none"> • The application should check whether the device is rooted or not, and if rooted, the application should warn the user about it or should not run the app. • References on implementing a fix for this vulnerability: <ul style="list-style-type: none"> - https://stackoverflow.com/a/8097801 - https://github.com/scottyab/rootbeer |

Looking at the types and severity of vulnerabilities found, we would recommend:

- The developer should not write sensitive data in the device logs.
- The developer should review the contents of all the files present in the application package for any other sensitive data being hardcoded in the files and should remove them from the application package.
- Strong cryptographic encryption should be used.
- We would also recommend that the client provides the developers with secure coding training to enhance their secure coding practice.

5. Appendix



5.1 Observations

In the observation section, we document any specific issue that we have observed, which, while does not directly lead to any vulnerability, maybe something that needs to be looked at from the perspective of functional bug or configuration issue.

5.1.1 Android Permissions mentioned in AndroidManifest.xml

Description

The purpose of permission is to protect the privacy of an Android user. Each Android app operates in a process sandbox, so apps must explicitly request access to resources and data outside their sandbox. The request is made by declaring the permission they need to use system data and features. Depending on how sensitive or critical the data of the feature is, the Android system will grant automatically or ask the user to approve the request. Make sure that the app needs these permissions and remove unnecessary permissions.

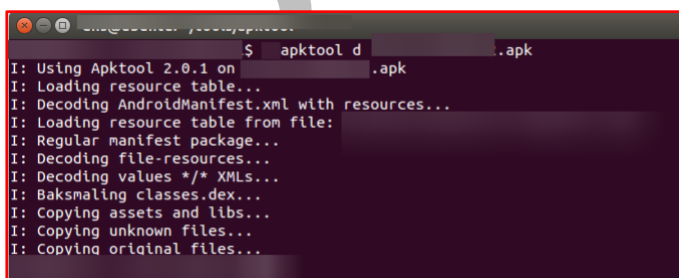
Testing Process

1. Make sure we have .apk file.
2. If the application has been downloaded from the Google Play store, we can get the .apk file by pulling it from the device using the Android inbuilt command line tool- pull – to transfer files from the mobile device to PC. Make sure the device is connected to the PC using the USB cable.

Cmd: adb pull {path-to-apk-file}

3. Open the terminal and using apktool decompile the .apk file.

Cmd: apktool d {Application-apk-name}.apk



```
$ apktool d [redacted].apk
I: Using Apktool 2.0.1 on [redacted].apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file:
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

4. Navigate to the folder created after the de-compilation of the .apk file.
5. Open the decompiled AndroidManifest.xml file. In this file, we can see all the permissions required by the android application.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="275" android:versionName="0.7.0" android:hardwareAccelerated="true" android:compileSdkVersion="28"
  android:compileSdkVersionCodename="9" package="com.payatu" platformBuildVersionCode="275" platformBuildVersionName="0.7.0"
  xmlns:android="http://schemas.android.com/apk/res/android">
  <uses-sdk android:minSdkVersion="21" android:targetSdkVersion="28" />
  <supports-screens android:anyDensity="true" android:smallScreens="true" android:normalScreens="true" android:largeScreens="true"
    android:resizeable="true" android:xlargeScreens="true" />
  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
  <uses-feature android:name="android.hardware.location.gps" />
  <uses-permission android:name="android.permission.RECORD_AUDIO" />
  <uses-permission android:name="android.permission.RECORD_VIDEO" />
  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
  <application android:label="@string/app_name" android:icon="@mipmap/ic_launcher" android:hardwareAccelerated="true" android:supportRtl="true"
    android:networkSecurityConfig="@xml/network_security_config">
    <activity android:theme="@android:style/Theme.DeviceDefault.NoActionBar" android:label="@string/activity_name"
      android:name=".MainActivity" android:launchMode="singleTop" android:screenOrientation="portrait"
      android:configChanges="keyboard|keyboardHidden|locale|orientation|screenLayout|screenSize|smallestScreenSize|uiMode"
      android:windowSoftInputMode="adjustResize">
      <intent-filter android:label="@string/launcher_name">
```



5.2 References

In this section, we have given additional data regarding the vulnerabilities that we have found during the engagement to provide better clarity and more insight into the bugs:

1. **Insecure Logging:** Logging is a method that developers use for tracing the code and watching warnings or errors. Unfortunately, Sometimes developers write sensitive data in logs. In such situations, other applications may gain access and read logs for stealing sensitive data.
Ref: <https://owasp.org/www-project-mobile-top-10/2016-risks/m2-insecure-data-storage>
2. **Reverse Engineering:** Reverse engineering is the process in which the application is decompiled to reveal its source code, libraries, string constants, etc. using any tool.
Ref: <https://owasp.org/www-project-mobile-top-10/2016-risks/m9-reverse-engineering>
3. **Broken Cryptography:** Broken cryptography in Android apps can be introduced due to various reasons. The two main reasons, as mentioned in OWASP Mobile Top 10 Projects, are:
 - Using a weak algorithm for encryption/decryption
 - Using a strong encryption algorithm but implementing it in an insecure way**Ref:** <https://owasp.org/www-project-mobile-top-10/2016-risks/m5-insufficient-cryptography>
4. **Insecure Communication (SSL Pinning not implemented):** Insecure Communication refers to mobile app vulnerability where the attacker can intercept the sensitive data that has been transmitted to the server in the network.
Ref: <https://owasp.org/www-project-mobile-top-10/2016-risks/m3-insecure-communication>



5.3 Failed Test Cases

In the failed test case section, we document some of the application strengths that we have observed during our testing. This serves as a view of the best practices that are being followed by the client.

5.3.1 Attacking Android Intents

Description

The developer might expose some intents without sufficient protection. Exposing intents can lead to various attacks. For example, a malicious application installed in the mobile device can call the exposed activities to invoke the internal pages of the application. Calling internal pages puts the application at risk of phishing by manipulating users to enter details in the phishing app, as well as exposing a user to secret pages, such as admin panels or pages which should have been visible to paid/pro user only.

Testing Process

1. Connect the device to the PC using the USB cable.
2. Start the drozer server on the mobile device.
3. Open the terminal in the device and enter the below command. We are using the Android inbuilt command line tool – forward – used for port forwarding.

Cmd: adb forward tcp:31415 tcp:31415

4. Now start drozer by entering below command. We will get the drozer shell.

Cmd: drozer console connect

5. In the drozer shell. Now we can run drozer command to get the details of activities, broadcast receivers, content providers, services, etc of the application installed in the device.

View the Attack surface of the application

Cmd 1: run app.package.attacksurface <app-package-name>

View the list of activities of the application

Cmd 2: run app.activity.info -a <app-package-name>

View the list of broadcast receivers of the application

Cmd 3: run app.broadcast.info -a <app-package-name>

View the list of service of the application

Cmd 4: run app.service.info -a <app-package-name>

```
dz> run app.package.attacksurface [REDACTED]
Attack Surface:
  3 activities exported
  3 broadcast receivers exported
  0 content providers exported
  6 services exported
dz> run app.activity.info -a [REDACTED]
Package: [REDACTED]
  Permission: null
  [REDACTED] Activity
  Permission: null
  [REDACTED] Activity
  Permission: null

dz> run app.broadcast.info -a [REDACTED]
Package: [REDACTED]
  [REDACTED] Broadcast
  Permission: null
  com.google.android.gms.measurement.AppMeasurementInstallReferrerReceiver
  Permission: android.permission.INSTALL_PACKAGES
  com.google.firebase.iid.FirebaseInstanceIdReceiver
  Permission: com.google.android.c2dm.permission.SEND

dz> run app.service.info -a [REDACTED]
Package: [REDACTED]
  com.hbtf.MADPPushService
  Permission: null
  [REDACTED].plugin.MyFirebaseInstanceIdService
  Permission: null
  [REDACTED].plugin.MyFirebaseMessagingService
  Permission: null
  com.google.android.gms.auth.api.signin.RevocationBoundService
  Permission: com.google.android.gms.auth.api.signin.permission.REVOCATION_NOTIFICATION
  com.google.firebase.messaging.FirebaseMessagingService
  Permission: null
  com.google.firebase.iid.FirebaseInstanceIdService
  Permission: null
```

5.3.2 SQL injection attacks at Client Side

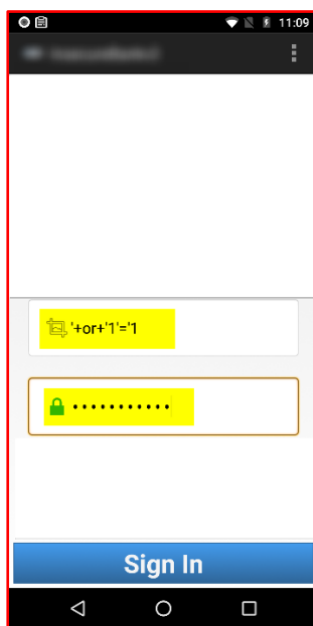
Description

We tested the application for bypassing the login screen of the application by trying the SQL injection attack. But the application seemed to have proper protection against SQL injection attacks.

Testing Process

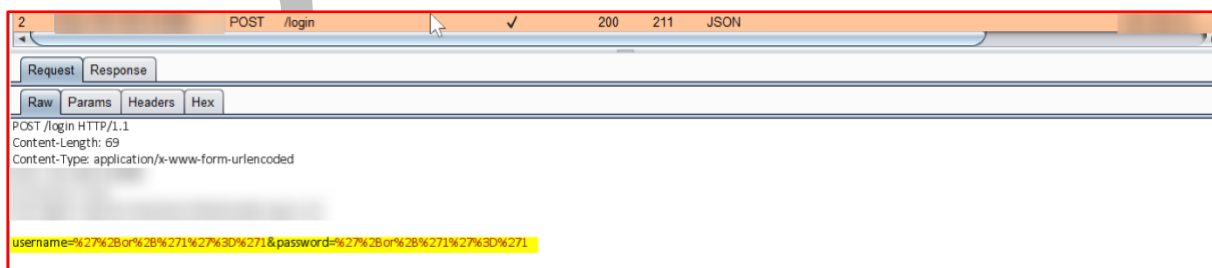
1. Start the application.
2. In the login screen, enter SQL injection payload in the username and the password field.

Payload: '+or+'1'='1

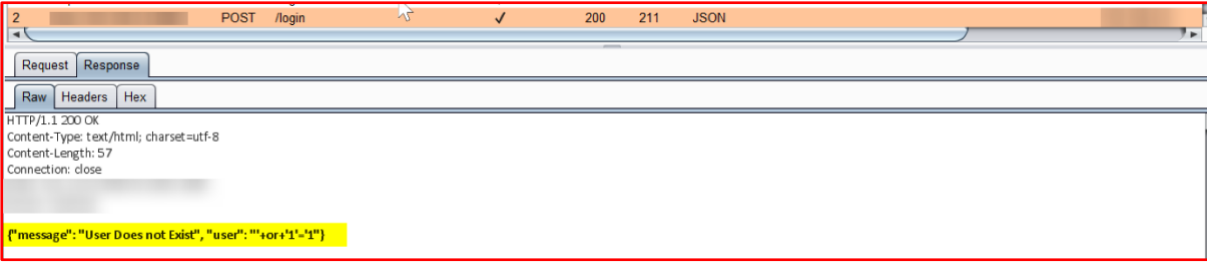


3. Click on the login button.
4. The login screen will not be bypassed and it will display that Incorrect credentials entered.

Request



Response



SAMPLE

5.3.3 Cross-Site Scripting (XSS) attacks at Client Side

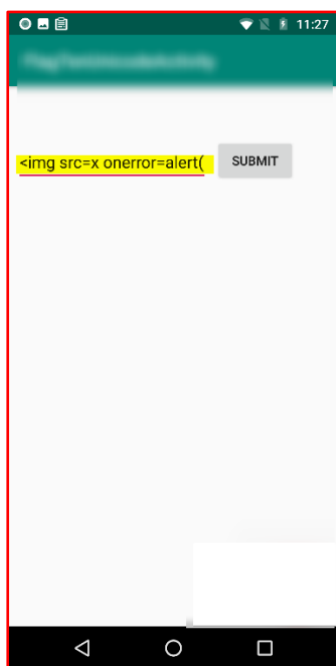
Description

We tested the application for a cross-site scripting attack and tried to exploit the web view in the application, but the application seems to have proper input validations.

Testing Process

1. Start the application.
2. Login into the application.
3. Now open {App-feature-Name}.
4. Enter the below payload in the text box.

Payload: ``



5. The application will not execute the malicious JavaScript payload in the web view.

