



Master of Information Security

MIS4203 – Independent Studies in Information Security

Index Number – 16770217 | Reg. Number – 2016MIS021

Study Number – 03 – Mobile App Vulnerabilities

December 01, 2018

University of Colombo School of Computing

Table of Contents

Study Number – 03 – Mobile App Vulnerabilities	1
Problem.....	3
Approach	3
Conclusion	23

Problem

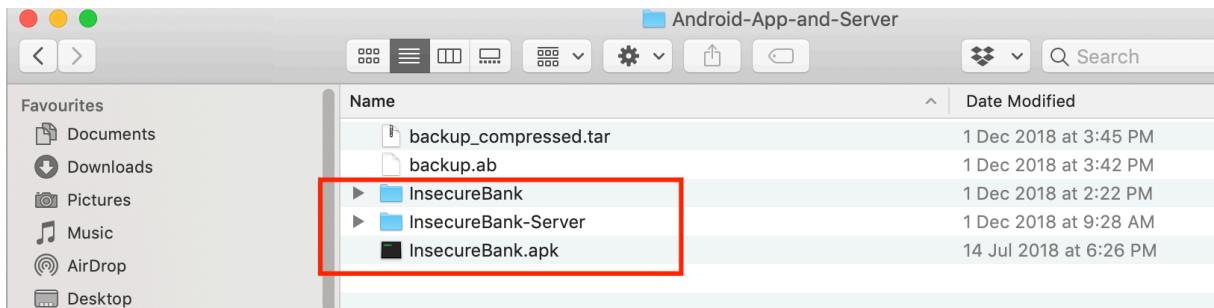
Android mobile application apk file and source code of the same application is given. We need to find the following vulnerabilities.

- Flawed Broadcast Receivers
- Intent Sniffing and Injection
- Weak Authorization mechanism
- Local Encryption issues
- Vulnerable Activity Components
- Root Detection and Bypass
- Insecure Content Provider access
- Insecure WebView implementation
- Weak Cryptography implementation
- Application Patching
- Sensitive Information in Memory
- Insecure Logging mechanism
- Android Pasteboard vulnerability
- Application Debuggable
- Android keyboard cache issues
- Android Backup vulnerability
- Runtime Manipulation
- Insecure SD Card storage
- Insecure HTTP connections
- Parameter Manipulation
- Hardcoded secrets
- Username Enumeration issue
- Developer Backdoors
- Weak change password implementation

Approach

1. Download Android-App-Server file from LMS and then extract.
2. It include InsecureBank and InsecureBank-Server folders and InsecureBank.apk.
3. InsecureBank folder include the android app.
4. InsecureBank-Server include the python backend server for the InsecureBank android app.
5. Setup android app and backend server in your local machine.

Downloaded Files from LMS:

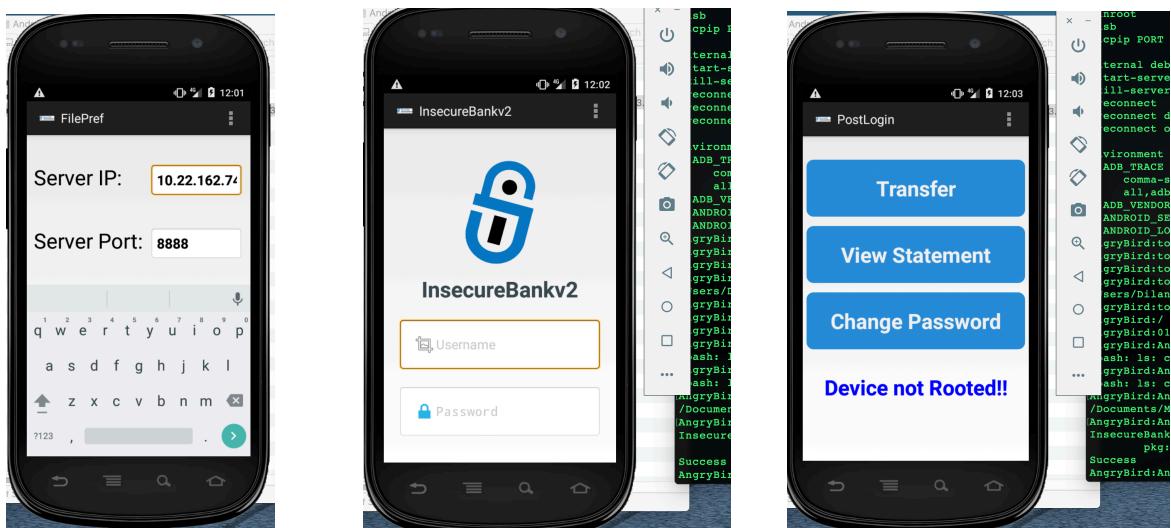


Up and running server in Kali-Linux VM:

```
root@kali:~/Downloads/Android-App-and-Server/InsecureBank-Server# python app.py
The server is hosted on port: 8888
```

Install and running android application' apk file and running on android VDM.

```
AngryBird:Android-App-and-Server Dilanka$ pwd
/Document/MIS/shared_between_vms/mini_projects/01-Dec-2018/Android-App-and-Server
AngryBird:Android-App-and-Server Dilanka$ adb install InsecureBank.apk
InsecureBank.apk: 1 file pushed. 80.6 MB/s (3632378 bytes in 0.043s)
    pkg: /data/local/tmp/InsecureBank.apk
Success
AngryBird:Android-App-and-Server Dilanka$
```



6. Find following vulnerabilities in the InsecureBank bank application.

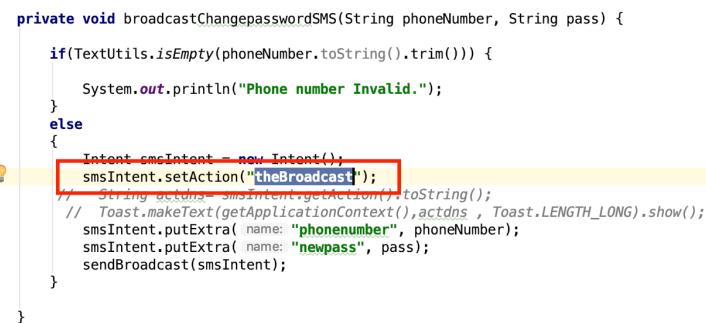
- Flawed Broadcast Receivers



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

This broadcast receiver is found on AndroidManifest.xml as above screenshot. When mobile application exporting contents without restrictions, other applications can access those.

By default, broadcast receivers are exported in Android, as the result any application will be able to send an intent to the Broadcast Receiver of the application. To define which applications can send intents to mobile application's Broadcast Receiver set relevant permissions in the Android Manifest file. Here we cannot see any permission tag. So this is insecure.



```
private void broadcastChangepasswordSMS(String phoneNumber, String pass) {
    if(TextUtils.isEmpty(phoneNumber.toString().trim())) {
        System.out.println("Phone number Invalid.");
    } else {
        Intent smsIntent = new Intent();
        smsIntent.setAction("theBroadcast");
        // String actions= smsIntent.getAction().toString();
        // Toast.makeText(getApplicationContext(),actions , Toast.LENGTH_LONG).show();
        smsIntent.putExtra( name: "phonenumber", phoneNumber);
        smsIntent.putExtra( name: "newpass", pass);
        sendBroadcast(smsIntent);
    }
}
```

- Intent Sniffing and Injection

Intent Sniffing

In android, intents are created implicitly without proper restrictions according to the application requirement. Since this application is banking application, so that intents used in this application must be in secured way to access from only this application. But according to below screenshot of code of application, this application has bad standard code which is not secure. So malicious application can respond to this intend and perform action instead of legitimate action.

```

private void broadcastChangepasswordSMS(String phoneNumber, String pass) {
    if(TextUtils.isEmpty(phoneNumber.toString().trim())) {
        System.out.println("Phone number Invalid.");
    } else {
        Intent smsIntent = new Intent();
        smsIntent.setAction("thebroadcast");
        // String actdns= smsIntent.getAction().toString();
        // Toast.makeText(getApplicationContext(),actdns , Toast.LENGTH_LONG).show();
        smsIntent.putExtra( name: "phonenumber", phoneNumber);
        smsIntent.putExtra( name: "newpass", pass);
        sendBroadcast(smsIntent);
    }
}

```

So this application must be practice proper coding standards when creating intent:

```
Intent myIntent = new Intent (this, MyIntent.class);
```

Injection

Below highlighted code in initialising variable from the context of the view which is not secured as it is exported to other application as well. So anyone who can access the app via adb shell, can inject some other values to those with below given command. So it is insecure.

```
am broadcast -a theBroadcast -n
com.android.insecurebankv2/com.android.insecurebankv2.MyBroadCastRecei
ver --es phonenumber 123456 --es newpass Pass@123!
```

```

public class MyBroadCastReceiver extends BroadcastReceiver {
    String usernameBase64ByteString;
    public static final String MYPREFS = "mySharedPreferences";
    @Override
    public void onReceive(Context context, Intent intent) {
        // TODO Auto-generated method stub
        String phn = intent.getStringExtra( name: "phonenumber");
        String newpass = intent.getStringExtra( name: "newpass");
        if (phn != null) {
            try {
                SharedPreferences settings = context.getSharedPreferences(MYPREFS, Context.MODE_WORLD_READABLE);
                final String username = settings.getString( s: "EncryptedUsername", s1: null);
                byte[] usernameBase64Byte = Base64.decode(username, Base64.DEFAULT);
                usernameBase64ByteString = new String(usernameBase64Byte, charsetName: "UTF-8");
                final String password = settings.getString( s: "superSecurePassword", s1: null);
                CryptoClass crypt = new CryptoClass();
                String decryptedPassword = crypt.aesDecryptedString(password);
                String textPhoneno = phn.toString();
                String textMessage = "Updated Password from: "+decryptedPassword+ " to: "+newpass;
                SmsManager smsManager = SmsManager.getDefault();
                System.out.println("For the changepassword - phonenumber: "+textPhoneno+ " password is: "+textMessage);
                smsManager.sendTextMessage(textPhoneno, scAddress: null, textMessage, sentIntent: null, deliveryIntent: null);
            } catch (Exception e) {
                e.printStackTrace();
            }
        } else {
            System.out.println("Phone number is null");
        }
    }
}

```

- Weak Authorization mechanism

No mechanism is used to user role mapping to this application. Proper mechanism is maintain user/role and permission matrix in database level to identify user permissions and user type.(for ex: which user has admin access, which user has high or low permissions). In this application only use flag which state in xml file as String.)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">InsecureBankv2</string>
    <string name="hello_world">Hello world!</string>
    <string name="loginscreen_username">Username:</string>
    <string name="action_settings">Preferences</string>
    <string name="action_exit">Restart</string>
    <string name="loginscreen_password">Password:</string>
    <string name="title_activity_file_pref">FilePref</string>
    <string name="server_ip">Server IP:</string>
    <string name="server_port">Server Port:</string>
    <string name="pref_submit">Submit:</string>
    <string name="title_activity_setting_preferences">SettingPreferences</string>
    <string name="title_activity_login">LoginActivity</string>
    <string name="title_activity_log_main">LogMainActivity</string>
    <string name="title_activity_do_login">DoLogin</string>
    <string name="title_activity_login_action">LoginAction</string>
    <string name="title_activity_post_login">PostLogin</string>
    <string name="title_activity_wrong_login">WrongLogin</string>
    <string name="title_activity_do_transfer">DoTransfer</string>
    <string name="title_activity_view_statement">ViewStatement</string>
    <string name="is_admin">no</string>
    <string name="title_activity_change_password">ChangePassword</string>
    <string name="title_activity_exit">ExitActivity</string>
    <string name="action_kill">Exit Application</string>

</resources>

<!--- -->
<string name="title_activity_wrong_login">WrongLogin</string>
<string name="title_activity_do_transfer">DoTransfer</string>
<string name="title_activity_view_statement">ViewStatement</string>
<string name="is_admin">yes</string>
<string name="title_activity_change_password">ChangePassword</string>
<string name="title_activity_exit">ExitActivity</string>
<string name="action_kill">Exit Application</string>
```

- Local Encryption issues

- Login username and password passed as plain text, as a proper security practice it should be encrypted or hided.

```
/*
nameValuePairs.add(new BasicNameValuePair( name: "username", uname));
nameValuePairs.add(new BasicNameValuePair( name: "newpassword", newPassword_text.getText().toString()));
httpresponse.setResponseBody;
httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
pattern = Pattern.compile(PASSWORD_PATTERN);
matcher = pattern.matcher(changePassword_text.getText().toString());
```

- User name and passwords are stored in shared preference. Reversible because of known secret key and IV and algorithm.

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="EncryptedUsername">amFjaw==</string>
    <string name="superSecurePassword">lUbe3xborJSKSuxYG3TPUg==</string>
</map>
root@generic_x86:/data/data/com.android.insecurebankv2/shared_prefs #
```

The page that holds the logic for encryption and decryption used in the
@author Dinesh Shetty

```
/*
public class CryptoClass {
```

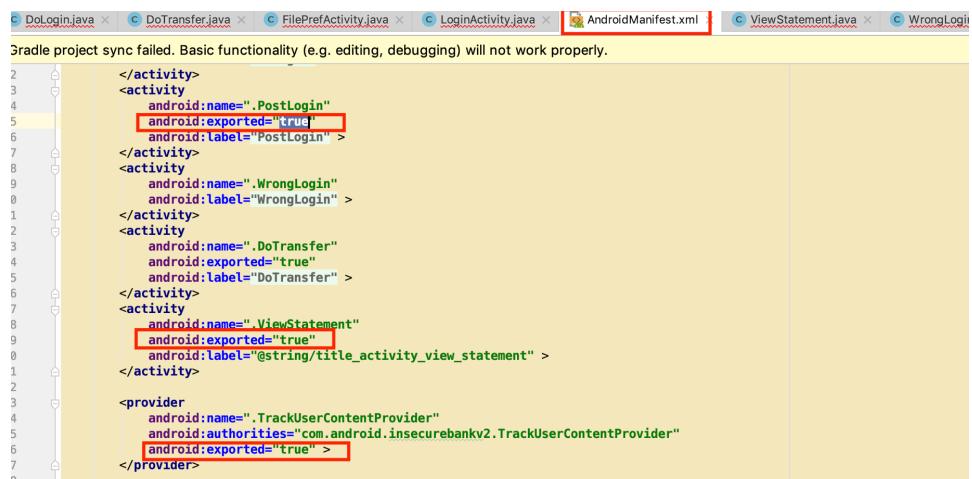
```
// The super secret key used by the encryption function
String key = "This is the super secret key 123";
```

```
public static byte[] aes256encrypt(byte[] ivBytes, byte[] keyBytes, byte[] textBytes)
throws UnsupportedEncodingException,
NoSuchAlgorithmException,
NoSuchPaddingException,
InvalidKeyException,
InvalidAlgorithmParameterException,
IllegalBlockSizeException,
BadPaddingException {

    AlgorithmParameterSpec ivSpec = new IvParameterSpec(ivBytes);
    SecretKeySpec newKey = new SecretKeySpec(keyBytes, algorithm: "AES");
    Cipher cipher = null;
    cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, newKey, ivSpec);
    return cipher.doFinal(textBytes);
}
```

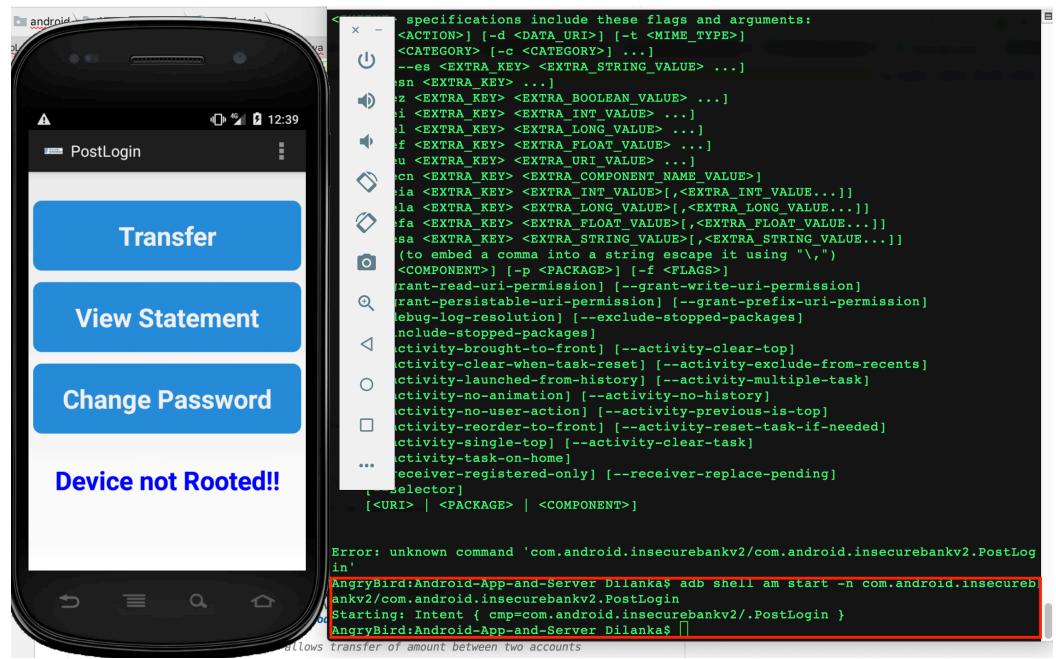
- Vulnerable Activity Components

In android all the activity components are declared in AndroidManifest.xml file in application root. It is the best practice that important activities which should be secured ones make not exportable. But in this application all the activities which should be secured are exported. Which is vulnerable and allow bypass security authentication mechanism.



For example, in above screenshot, post login activity is exportable and it is not secured as attacker can execute that activity directly bypassing the authentication.

```
adb shell am start -n
com.android.insecurebankv2/com.android.insecurebankv2.PostLogin
```



- Root Detection and Bypass

Root detection logic is stated as below screenshot. According to that logic, it is checking whether some apk file exists or not. If it exists application will be automatically rooted. So application and android manager never know this application is rooted.

Note: even below code also has semantic error: instead of `if(isrooted==true)` it should be `if(isrooted)`

```

    void showRootStatus() {
        boolean isrooted = doesSuperuserApkExist( s: "/system/app/Superuser.apk" ) || !doesSUexist();
        if(isrooted==true)
        {
            root_status.setText("Rooted Device!!!");
        }
        else
        {
            root_status.setText("Device not Rooted!!!");
        }
    }

    private boolean doesSUexist() {
        Process process = null;
        try {
            process = Runtime.getRuntime().exec(new String[] { "/system/bin/which", "su" });
            BufferedReader in = new BufferedReader(new InputStreamReader(process.getInputStream()));
            if (in.readLine() != null) return true;
            return false;
        } catch (Throwable t) {
            return false;
        } finally {
            if (process != null) process.destroy();
        }
    }

    private boolean doesSuperuserApkExist(String s) {
        File rootFile = new File( pathname: "/system/app/Superuser.apk");
        Boolean doesexist = rootFile.exists();
        if(doesexist == true)
        {
            return(true);
        }
        else
        {
            return(false);
        }
    }
}

```

- Insecure Content Provider access

Broadcast receiver content provider is declared in AndroidManifest.xml file.

```
<provider
    android:name=".TrackUserContentProvider"
    android:authorities="com.android.insecurebankv2.TrackUserContentProvider"
    android:exported="true" >
</provider>
```

When analysing TrackUserContentProvider.class, Below url can be found and which is users' history information.

Content query –uri

content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers

```
static final String PROVIDER_NAME = "com.android.insecurebankv2.TrackUserContentProvider";
// The Content provider that handles all the tracked user history
static final String URL = "content://" + PROVIDER_NAME + "/trackerusers";
static final Uri CONTENT_URI = Uri.parse(URL);
static final String name = "name";
static final int uriCode = 1;
static final UriMatcher uriMatcher;
private static HashMap<String, String> values;
private SQLiteDatabase db;
static final String DATABASE_NAME = "mydb";
static final String TABLE_NAME = "names";
static final int DATABASE_VERSION = 1;
static final String CREATE_DB_TABLE = " CREATE TABLE " + TABLE_NAME + " (id INTEGER PRIMARY KEY AUTOINCREMENT, " + "
```

- Insecure WebView implementation

App is insecure since it is enabled javascript, by default android app's java script is disabled. So this is vulnerable for malicious javascript executing or XSS attacks.

```
if (fileToCheck.exists()) {
    //Toast.makeText(this, "Statement Exists!!",Toast.LENGTH_LONG).show();

    WebView mWebView = (WebView) findViewById(R.id.webView1);
    // Location where the statements are stored locally on the device sdcard
    mWebView.loadUrl("file://" + Environment.getExternalStorageDirectory() + "/Statements_" + uname +
mWebView.getSettings().setJavaScriptEnabled(true);
    mWebView.getSettings().setSaveFormData(true);
    mWebView.getSettings().setBuiltInZoomControls(true);
    mWebView.setWebViewClient(new MyWebViewClient());
    WebChromeClient cClient = new WebChromeClient();
    mWebView.setWebChromeClient(cClient);
} else
```

No url validation provided

```
/*
The class that manages the WebView functionality used in the application
@author Dinesh Shetty
*/
public class MyWebViewClient extends WebViewClient {
    @Override
    public boolean shouldOverrideUrlLoading(WebView view, String url) {
        view.loadUrl(url);
        return true;
}
```

- Weak Cryptography implementation

We can decompile the apk file or find the hard coded below information from the code.

Key used for encryption:

```
The page that holds the logic for encryption and decryption used in the
@author Dinesh Shetty
*/
public class CryptoClass {

    // The super secret key used by the encryption function
    String key = "This is the super secret key 123";
```

Encryption algorithm information:

```
public static byte[] aes256encrypt(byte[] ivBytes, byte[] keyBytes, byte[] textBytes)
throws UnsupportedEncodingException,
NoSuchAlgorithmException,
NoSuchPaddingException,
InvalidKeyException,
InvalidAlgorithmParameterException,
IllegalBlockSizeException,
BadPaddingException {

    AlgorithmParameterSpec ivSpec = new IvParameterSpec(ivBytes);
    SecretKeySpec newKey = new SecretKeySpec(keyBytes, algorithm: "AES");
    Cipher cipher = null;
    cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, newKey, ivSpec);
    return cipher.doFinal(textBytes);
}
```

Cipher text from the shared preference.xml file from the adb shell:

```
[root@generic_x86:/data/data/com.android.insecurebankv2/shared_prefs # ls
com.android.insecurebankv2_preferences.xml
mySharedPreferences.xml
[root@generic_x86:/data/data/com.android.insecurebankv2/shared_prefs #
[root@generic_x86:/data/data/com.android.insecurebankv2/shared_prefs #
[root@generic_x86:/data/data/com.android.insecurebankv2/shared_prefs # ]]
```

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="EncryptedUsername">amFjaw==
    </string>
    <string name="superSecurePassword">1Ube3xborJSKSuxYG3TPUg==
```

Now we know key, IV and cipher text. So we can use reverse engineering tool to find the password as plain text.

<https://www.devglan.com/online-tools/aes-encryption-decryption>

AES Online Decryption

Enter text to be Decrypted

fUbe3xborJSKSuxYG9TPUg==

Input Text Format: Base64 Hex

Select Mode

ECB

Enter IV Used During Encryption(Optional)

This is the super secret key 12

Key Size in Bits

256

Enter Secret Key

This is the super secret key 123

Decrypt

AES Decrypted Output (Base64):

cGFzc3dvcmQ=

Decode to Plain Text

password

- Application Patching

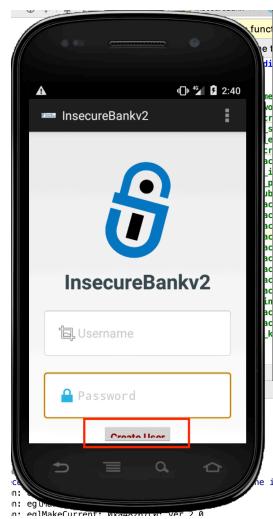
In this code, the login user is admin or not is checking by using String flag. If the attacker can access the relevant file from the adb shell, attacker can gain the admin rights to the application, do malicious things that attacker want and reverse it to the previous state without any trace.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">InsecureBankv2</string>
    <string name="hello_world">Hello world!</string>
    <string name="loginscreen_username">Username:</string>
    <string name="action_settings">Preferences</string>
    <string name="action_exit">Restart</string>
    <string name="loginscreen_password">Password:</string>
    <string name="title_activity_file_pref">FilePref</string>
    <string name="server_ip">Server IP:</string>
    <string name="server_port">Server Port:</string>
    <string name="pref_submit">Submit:</string>
    <string name="title_activity_setting_preferences">SettingPreferences</string>
    <string name="title_activity_login">LoginActivity</string>
    <string name="title_activity_log_main">LogMainActivity</string>
    <string name="title_activity_do_login">DoLogin</string>
    <string name="title_activity_login_action">LoginAction</string>
    <string name="title_activity_post_login">PostLogin</string>
    <string name="title_activity_wrong_login">WrongLogin</string>
    <string name="title_activity_do_transfer">DoTransfer</string>
    <string name="title_activity_view_statement">ViewStatement</string>
    <string name="is_admin">no</string>
    <string name="title_activity_change_password">ChangePassword</string>
    <string name="title_activity_exit">ExitActivity</string>
    <string name="action_kill">Exit Application</string>

</resources>

----- -----
<string name="title_activity_wrong_login">WrongLogin</string>
<string name="title_activity_do_transfer">DoTransfer</string>
<string name="title_activity_view_statement">ViewStatement</string>
<string name="is_admin">yes</string>
<string name="title_activity_change_password">ChangePassword</string>
<string name="title_activity_exit">ExitActivity</string>
<string name="action_kill">Exit Application</string>
```



- Sensitive Information in Memory

Can use android application monitor as below screen.

```
/Users/Dilanka/Library/Android/sdk/tools
AngryBird:tools Dilanka$ monitor
-bash: monitor: command not found
AngryBird:tools Dilanka$ ls
NOTICE.txt           emulator
android              emulator-check
bin                  lib
mksdcard
monitor
package.x
AngryBird:tools Dilanka$ ./monitor
```

Android Device Monitor

Name	Online	2016MIS021 [5.1.1, debug]
emulator-5554	1535	8600
system_process	1956	8649
com.android.systemui	1976	8650
android.process.acore	2093	8651
com.android.inputmethod.latin	2133	8652
com.google.android.googlequicksearchbox:interactor	2157	8653
com.android.phone	2244	8656
com.android.media	2279	8657
com.google.android.googlequicksearchbox:search	2350	8662
com.google.process.gapps	2414	8664
com.google.android.googlequicksearchbox	2497	8665
com.google.android.gms.persistent	2521	8666
com.google.android.gms	2597	8667
com.android.defcontainer	2831	8670
com.google.android.calendar	2861	8671
com.android.providers.calendar	3038	8676
com.android.mms	3355	8677
com.svox.pico	3543	8660
com.google.android.play.games.ui	3627	8672 / 8700
com.android.insecurebankv2	3724	8655
com.android.browser	3752	8663

The screenshot shows the Android Device Monitor interface. The top bar includes 'Quick Access', 'DDMS', 'File Expl...', 'Emulator...', and 'System I...'. The 'Devices' tab is selected, displaying a list of running processes:

Name	PID	TID	Application
com.google.android.googlequicksearchbox	2430	8662	
com.google.android.gmsservice	2414	8664	
com.google.android.gms.persistent	2497	8665	
com.google.android.gms	2521	8666	
com.android.defcontainer	2597	8667	
com.google.android.calendar	2831	8670	
com.google.providers.calendar	2861	8671	
com.android.dns	3058	8676	
com.svox.pico	3386	8677	
com.google.android.play.games.ui	3543	8660	
com.android.insecurebankv2	3627	8672 / 8700	
com.android.browser	3724	8655	
com.android.exchange	3752	8663	

The 'LogCat' tab is also visible, showing log messages from the application. One message is highlighted in red:

```

D 12-01 15:04:00.098 3627 3663 com.android.insecurebankv2 Successful Login: <account->jack;password>
I 12-01 15:04:00.102 3535 1814 system_process ActivityManager
V 12-01 15:04:00.104 1535 1814 system_process WindowManager
V 12-01 15:04:00.137 1535 1555 system_process WindowManager

```

Further analysis can be done via tools such as JProfiler.

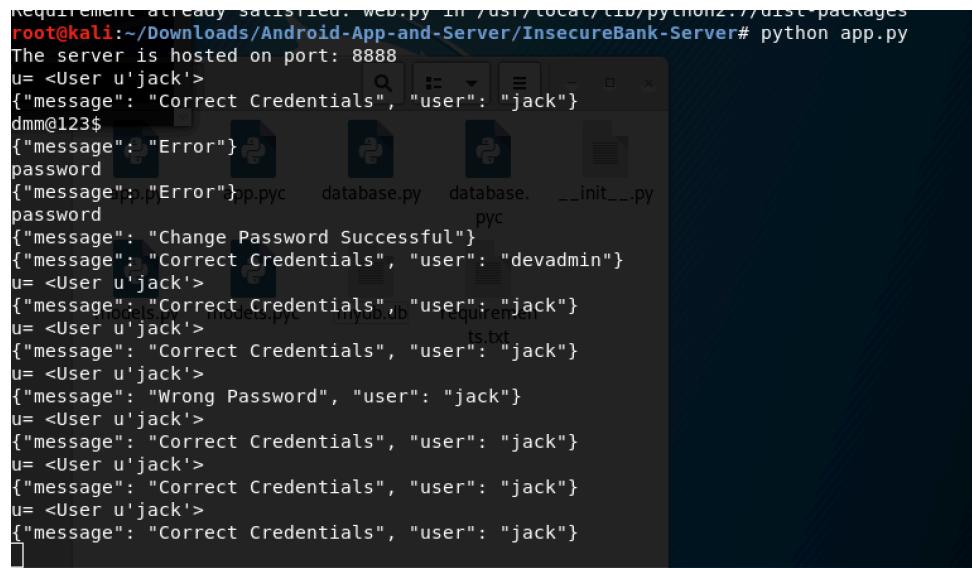
The screenshot shows the JProfiler interface for the file `com.android.insecurebankv2.hprof`. The top menu includes 'Start Center', 'Detach', 'Save Snapshot', 'Session Settings', 'Start Recordings', 'Stop Recordings', 'Start Tracking', 'Run GC', 'Add Bookmark', 'Export', 'View', 'Reveal Tools', 'Help', 'Take Snapshot', 'Mark Heap', 'Back', 'Forward', 'Go To Start', and 'Show Selection'. The main window displays memory analysis results for `java.lang.String[]`:

Object	Retained Size	Shallow Size	Allocation Time (h:m:s)
java.lang.String[]	358 kB	347 kB	n/a
java.lang.String[]	157 kB	157 kB	n/a
java.lang.String[]	153 kB	153 kB	n/a
java.lang.String[]	97,664 bytes	94,664 bytes	n/a
java.lang.String[]	69,984 bytes	69,984 bytes	n/a
java.lang.String[]	58,728 bytes	58,728 bytes	n/a
java.lang.String[]	44,120 bytes	44,120 bytes	n/a
java.lang.String[]	32,976 bytes	32,976 bytes	n/a
java.lang.String[]	14,656 bytes	14,656 bytes	n/a
java.lang.String[]	13,296 bytes	13,296 bytes	n/a
java.lang.String[]	11,496 bytes	11,496 bytes	n/a
java.lang.String[]	7,200 bytes	7,200 bytes	n/a
java.lang.String[]	4,824 bytes	4,824 bytes	n/a
java.lang.String[]	2,344 bytes	2,344 bytes	n/a
java.lang.String[]	2,344 bytes	2,344 bytes	n/a
java.lang.String[]	2,256 bytes	2,256 bytes	n/a
java.lang.String[]	1,520 bytes	1,520 bytes	n/a
java.lang.String[]	1,008 bytes	1,008 bytes	n/a

The bottom status bar indicates 'unlicensed copy for evaluation purposes, 10 days remaining', '0 recordings', 'Dec 1, 2018 3:07:33 PM', 'VM #1', '00:00', and a 'Snapshot' button.

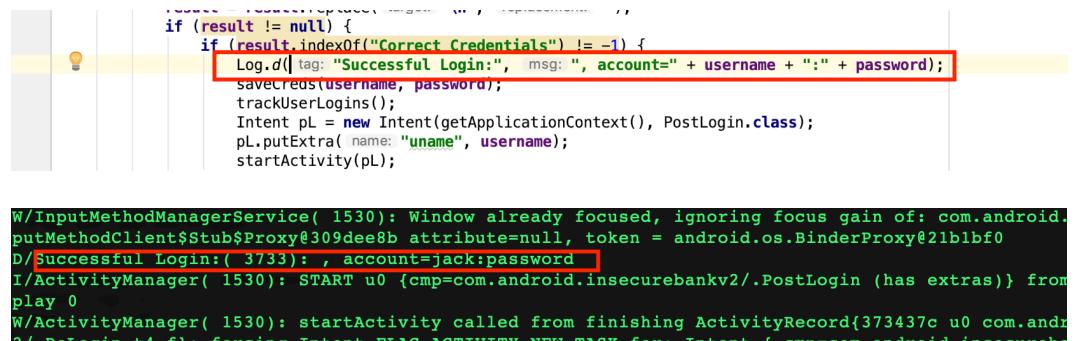
- Insecure Logging mechanism

Login credentials can be seen in server log/console which is insecure for the app.



```
Requirement already satisfied: web.py in /usr/local/lib/python2.7/dist-packages
root@kali:~/Downloads/Android-App-and-Server/InsecureBank-Server# python app.py
The server is hosted on port: 8888
u= <User u'jack'>
{"message": "Correct Credentials", "user": "jack"}
dmm@123$
{"message": "Error"}
password
{"message": "Error"}app.pyc database.py database. __init__.py
password
{"message": "Change Password Successful"}
{"message": "Correct Credentials", "user": "devadmin"}
u= <User u'jack'>
{"message": "Correct Credentials", "user": "jack"}
u= <User u'jack'>
{"message": "Correct Credentials", "user": "jack"}
u= <User u'jack'>
{"message": "Wrong Password", "user": "jack"}
u= <User u'jack'>
{"message": "Correct Credentials", "user": "jack"}
u= <User u'jack'>
{"message": "Correct Credentials", "user": "jack"}
u= <User u'jack'>
{"message": "Correct Credentials", "user": "jack"}
```

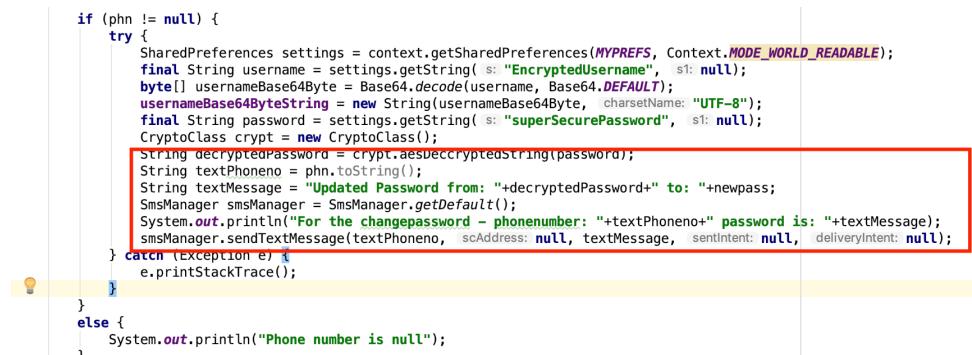
Password (Together with user name) is logged in plain text.



```
if (result != null) {
    if (result.indexOf("Correct Credentials") != -1) {
        Log.d("tag: "Successful Login:", msg: ", account=" + username + ":" + password);
        saveCredentials(username, password);
        trackUserLogins();
        Intent pl = new Intent(getApplicationContext(), PostLogin.class);
        pl.putExtra( name: "uname", username);
        startActivity(pl);
```

W/InputMethodManagerService(1530): Window already focused, ignoring focus gain of: com.android.
putMethodClient\$Stub\$Proxy@309dee8b attribute=null, token = android.os.BinderProxy@21b1bf0
D/Successful Login:(3733): , account=jack:password
I/ActivityManager(1530): START u0 {cmp=com.android.insecurebankv2/.PostLogin (has extras)} from
play 0
W/ActivityManager(1530): startActivity called from finishing ActivityRecord{373437c u0 com.andr
o.insecurebankv2/.PostLogin (1 1) finished T 1530 8145 1530 8145 ACTIVITY NEW_TASK from Intent f
rom com.android.insecurebankv2/.PostLogin (1 1)}

Some information showed with non-standard way with normal Java print statements.

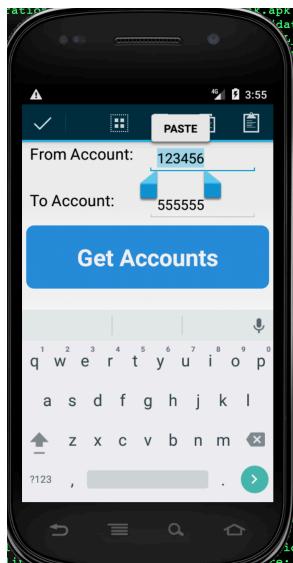


```
if (phn != null) {
    try {
        Sharedpreferences settings = context.getSharedPreferences(MYPREFS, Context.MODE_WORLD_READABLE);
        final String username = settings.getString( s: "EncryptedUsername", s1: null);
        byte[] usernameBase64Byte = Base64.decode(username, Base64.DEFAULT);
        usernameBase64ByteString = new String(usernameBase64Byte, charsetName: "UTF-8");
        final String password = settings.getString( s: "superSecurePassword", s1: null);
        CryptoClass crypt = new CryptoClass();
        String decryptedPassword = crypt.aesDecryptedString(password);
        String textPhoneno = phn.toString();
        String textMessage = "Updated Password from: "+decryptedPassword+ " to: "+newpass;
        SmsManager smsManager = SmsManager.getDefault();
        System.out.println("For the changepassword - phoneno: "+textPhoneno+ " password is: "+textMessage);
        smsManager.sendTextMessage(textPhoneno, scAddress: null, textMessage, sentIntent: null, deliveryIntent: null);
    } catch (Exception e) {
        e.printStackTrace();
    }
} else {
    System.out.println("Phone number is null");
}
```

- Android Pasteboard vulnerability

Information that we copied to the clipboard in this application can be read from the clipboard memory via below command:

```
adb shell su u0_a58 service call clipboard 2 s16 com.android.insecurebankv2
```



```
AngryBird:Android-App-and-Server Dilanka$ adb shell su u0_a58 service call clipboard 2 s16 com.android.insecurebankv2
Result: Parcel{
    0x00000000: 00000001 00000001 ffffff '.....'
    0x00000010: 00000001 00650074 00740078 '.....t.e.x.t.'
    0x00000020: 0070002f 0061006c 006e0069 00000000 '/.p.l.a.i.n..'
    0x00000030: 00000000 00000001 00000000 00000006 '.....'
    0x00000040: 00320031 00340033 00360035 00000000 1.2.3.4.5.6....
    0x00000050: 00000014 00000000 00000000 00000006 '.....'
    0x00000060: 00000021 00000000 ffffffff 00000000 '!.....'
    0x00000070: 00000000 '....')
}
AngryBird:Android-App-and-Server Dilanka$
```

- Application Debuggable

- There is 'android:debuggable="true"' found in file "android/AndroidManifest.xml"
- *Code sample*

```
<application android:allowBackup="true"
    android:debuggable="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@android:style/Theme.Holo.Light.
    DarkActionBar">
```

To avoid this change code to `android:debuggable="false"`

```
<application
    android:allowBackup="true"
    android:debuggable="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@android:style/Theme.Holo.Light.DarkActionBar">
<!--
    android:theme="@style/AppTheme"-->
```

- Android keyboard cache issues
 - Android by default showing previous typed information as suggestions. But best practice is sensitive information should not behave like that. But examining `AndroidManifest.xml` file it is allowed. Developer has to disable it by using below code tag.

```
        android:inputType="textNoSuggestions"
```

The screenshot shows a portion of an Android XML layout file. It includes a `<TextView>` and an `<EditText>`. The `<EditText>` element has the attribute `android:inputType="textNoSuggestions"` highlighted with a yellow background and a lightbulb icon. The rest of the code is in white.

```

<TextView android:id="@+id/textView_FromAccount"
    android:layout_width="wrap_content"
    android:text="From Account:"
    android:textSize="20sp"
    android:textColor="#000000"
    android:layout_height="wrap_content"
    android:layout_marginLeft="10dp"></TextView>
</FrameLayout>
<FrameLayout
    android:layout_height="wrap_content"
    android:layout_width="0dp"
    android:layout_weight="1">
<EditText android:layout_width="fill_parent"
    android:id="@+id/editText_from"
    android:layout_height="wrap_content"
    android:layout_marginLeft="00dp"
    android:layout_marginRight="20dp"
    android:layout_weight="1"
    android:singleLine="true">
</EditText>

```

- Once keyboard cache enabled, we can find it android sqllite database, user_dict database, words table. Then open the file user_dict.db words table using sqlite3 can be identified the what are the dictionary added words . That may lead to leak the sensitive information.
- Android Backup vulnerability
 - When examine `AndroidManifest.xml` file, This app is allowed to backup.

The screenshot shows a portion of an Android XML manifest file. It includes an `<application>` tag. Inside the `<application>` tag, the attribute `android:allowBackup="true"` is highlighted with a red rectangle and a lightbulb icon. The rest of the code is in white.

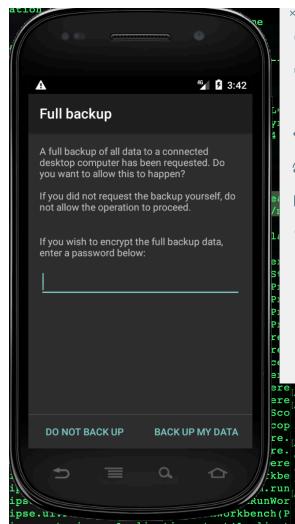
```

<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@android:style/Theme.Holo.Light.DarkActionBar">
    <!--
    android:theme="@style/AppTheme"-->
<activity ...>

```

Anyone can backup the application via below command:

```
AngryBird:Android-App-and-Server Dilanka$ adb backup -apk -shared com.android.insecurebankv2
Now unlock your device and confirm the backup operation...
```



Once you extracted the backup file, it will contain stored credentials, transaction logs and history. Mainly sharedPreferences related information.

- Runtime Manipulation

When examining AndroidManifest.xml file. The sensitive activity is exportable which is doTransfer which can be directly manipulated and accessed since it is exportable. So it is vulnerable.

```
<activity
    android:name=".DoTransfer"
    android:exported="true"
    android:label="DoTransfer" >
</activity>
```

- Insecure SD Card storage

This application can access the external storages in READ/WRITE mode. That means application can save bank related financial data into external storage such as SD card. It will help other apps including malicious apps or external users to access those data from the SD card.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android.insecurebankv2" >

    <uses-sdk
        android:minSdkVersion="11"
        android:targetSdkVersion="24" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.SEND_SMS" />

    <!--
        To retrieve OAuth 2.0 tokens or invalidate tokens to disconnect a user. This disconnect
        option is required to comply with the Google+ Sign-In developer policies
    -->
    <uses-permission android:name="android.permission.USE_CREDENTIALS" /> <!-- To retrieve the account name (email) as
    <uses-permission android:name="android.permission.GET_ACCOUNTS" /> <!-- To auto-complete the email text field in th
    <uses-permission android:name="android.permission.READ_PROFILE" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />

    <android:uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <android:uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
        android:maxSdkVersion="18" />
    <android:uses-permission android:name="android.permission.READ_CALL_LOG" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@android:style/Theme.Holo.Light.DarkActionBar">
        <!--
            android:theme="@style/AppTheme" -->
        <activity
            android:name=".LoginActivity"
            android:label="InsecureBankV2" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
        
```



```

        try {
            jsonObject = new JSONObject(result);
            acc1 = jsonObject.getString( name: "from");
            acc2 = jsonObject.getString( name: "to");
            System.out.println("Message:" + jsonObject.getString( name: "message") + " From:" + from.getText().toString() + " To:" + to.getText().toString());
            final String status = new String( original: "\nMessage:" + "Success" + " From:" + from.getText().toString() + " To:" + to.getText().toString());
            try {
                // Captures the successful transaction status for Transaction history tracking
                String MYFILE = Environment.getExternalStorageDirectory() + "/Statements_" + usernameBase64ByteString + ".html";
                BufferedWriter out2 = new BufferedWriter(new FileWriter(MYFILE, append: true));
                out2.write(status);
                out2.write( str:<hr>);
                out2.close();
            } catch (IOException e) {
                e.toString();
            }
        } catch (JSONException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    
```

- Insecure HTTP connections
 - Less secure protocol is used comparatively and it is hard coded.

```

public class DoLogin extends Activity {
    String responseString = null;
    // Stores the username passed by the calling intent
    String username;
    // Stores the password passed by the calling intent
    String password;
    String result;
    String superSecurePassword;
    String rememberme_username, rememberme_password;
    public static final String MYPREFS = "mySharedPreferences";
    String serverip = "";
    String serverport = "", 
    String protocol = "http://";
    BufferedReader reader,
    SharedPreferences serverDetails;

    // Create a new HttpClient and Post Header
    HttpClient httpclient = new DefaultHttpClient();
    HttpPost httppost = new HttpPost( uri: protocol + serverip + ":" + serverport + "/login");
    HttpPost httppost2 = new HttpPost( uri: protocol + serverip + ":" + serverport + "/devlogin");

    // Add your data

```

- Parameter Manipulation

- Below highlighted code in initialising variable from the context of the view which is not secured as it is exported to other application as well. So anyone who can access the app via adb shell, can inject some other values to those with below given command. So it is insecure.
- am broadcast -a theBroadcast -n com.android.insecurebankv2/com.android.insecurebankv2.MyBroadCa stReceiver --es phonenumer 123456 --es newpass Pass@123!

```

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

    @Override
    public void onReceive(Context context, Intent intent) {
        // TODO Auto-generated method stub

        String phn = intent.getStringExtra( name: "phonenumer");
        String newpass = intent.getStringExtra( name: "newpass");

        if (phn != null) {
            try {
                SharedPreferences settings = context.getSharedPreferences(MYPREFS, Context.MODE_WORLD_READABLE);
                final String username = settings.getString( s: "EncryptedUsername", s1: null);
                byte[] usernameBase64Byte = Base64.decode(username, Base64.DEFAULT);
                usernameBase64ByteString = new String(usernameBase64Byte, charsetName: "UTF-8");
                final String password = settings.getString( s: "superSecurePassword", s1: null);
                CryptoClass crypt = new CryptoClass();
                String decryptedPassword = crypt.aesDecryptedString(password);
                String textPhoneno = phn.toString();
                String textMessage = "Updated Password from: "+decryptedPassword+ " to: "+newpass;
                SmsManager smsManager = SmsManager.getDefault();
                System.out.println("For the changepassword - phonenumer: "+textPhoneno+ " password is: "+textMessage);
                smsManager.sendTextMessage(textPhoneno, scAddress: null, textMessage, sentIntent: null, deliveryIntent: null);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        else {
            System.out.println("Phone number is null");
        }
    }
}

```

- Hardcoded secrets

Some hard coded credentials as comments not removed. Seems not code reviewed properly.

```

// Create a new HttpClient and Post Header
HttpClient httpClient = new DefaultHttpClient();
HttpPost httppost = new HttpPost( uri: protocol + serverip + ":" + serverport + "/login");
HttpPost httppost2 = new HttpPost( uri: protocol + serverip + ":" + serverport + "/devlogin");

// Add your data
List < NameValuePair > nameValuePairs = new ArrayList <> ( initialCapacity: 2);

// Delete below test accounts in production
// nameValuePairs.add(new BasicNameValuePair("username", "jack"));
// nameValuePairs.add(new BasicNameValuePair("password", "jack@123$"));

nameValuePairs.add(new BasicNameValuePair( name: "username", username));
nameValuePairs.add(new BasicNameValuePair( name: "password", password));
HttpResponse responseBody;
if (username.equals("devadmin")) {
    httppost2.setEntity(new UrlEncodedFormEntity(nameValuePairs));
    // Execute HTTP Post Request
    responseBody = httpClient.execute(httppost2);
} else {
    httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
    // Execute HTTP Post Request
    responseBody = httpClient.execute(httppost);
}

```

Cryptographic information also hard coded:

- Username Enumeration issue

Username and password can be brute forced by analysing api calls as below:

```
Requirement already satisfied: web.py in /usr/lib/python2.7/dist-packages
root@kali:~/Downloads/Android-App-and-Server/InsecureBank-Server# python app.py
The server is hosted on port: 8888
u= <User u'jack'>
{"message": "Correct Credentials", "user": "jack"}
dmm@123$ 
{"message": "Error"}   
password  app.pyc database.py database. __init__.py
password  pyc
{"message": "Change Password Successful"}
{"message": "Correct Credentials", "user": "devadmin"}
u= <User u'jack'>
{"message": "Correct Credentials", "user": "jack"}  
u= <User u'jack'>
{"message": "Correct Credentials", "user": "jack"} 
u= <User u'jack'>
{"message": "Wrong Password", "user": "jack"}
u= <User u'jack'>
{"message": "Correct Credentials", "user": "jack"}
u= <User u'jack'>
{"message": "Correct Credentials", "user": "jack"}
u= <User u'jack'>
{"message": "Correct Credentials", "user": "jack"}
```

Code segment:

```
32 ▼ def login():
33     Responsemsg="fail"
34     user = request.form['username']
35     #checks for presence of user in the database #requires models.py
36     u = User.query.filter(User.username == request.form["username"]).first()
37     print "u=",u
38     if u and u.password == request.form["password"]:
39         Responsemsg="Correct Credentials"
40     elif u and u.password != request.form["password"]:
41         Responsemsg="Wrong Password"
42     elif not u:
43         Responsemsg="User Does not Exist"
44     else: Responsemsg="Some Error"
45     data = {"message" : Responsemsg, "user": user}
46     print makejson(data)
47     return makejson(data)
48
```

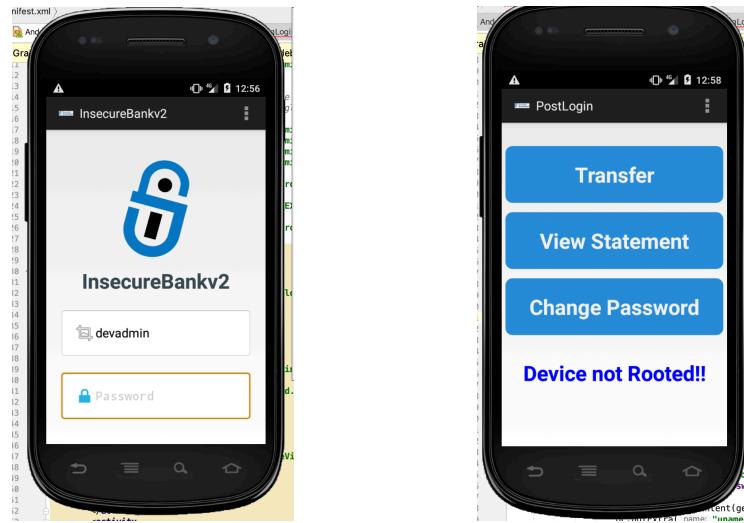
- Developer Backdoors

- Hardcoded user (devadmin) can login with any password authentication:

```

HttpResponse responseBody;
if [username.equals("devadmin")) {
    httpPost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
    // Execute HTTP Post Request
    responseBody = httpClient.execute(httpPost);
} else {
    httpPost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
    // Execute HTTP Post Request
    responseBody = httpClient.execute(httpPost);
}

```



- Weak change password implementation

- Only looking for the new password and change it directly without validation.
- This can be considered as developer backdoor as someone can access the rest url directly and change the password without app. Rest endpoint must be secured, or validate with existing password of the user.

```

/*
The function that makes an HTTP Post to the server endpoint that handles the
change password operation.
*/
public void postData(String valueToSend) throws ClientProtocolException, IOException, JSONException, NoSuchAlgorithmException, NoSuchKeyException {
    HttpClient httpClient = new DefaultHttpClient();
    HttpPost httpPost = new HttpPost(protocol + ":" + serverip + ":" + serverport + "/changepassword");
    List<NameValuePair> nameValuePairs = new ArrayList<> (initialCapacity: 2);
    /*
     * Delete below test accounts once the application goes into production phase.
     * nameValuePairs.add(new BasicNameValuePair("username", "jack"));
     * nameValuePairs.add(new BasicNameValuePair("password", "Jack@123$"));
     */
    nameValuePairs.add(new BasicNameValuePair( name: "username", value: uname));
    nameValuePairs.add(new BasicNameValuePair( name: "newpassword", value: newPassword_text.getText().toString()));
    HttpResponse responsebody;
    httpPost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
    pattern = Pattern.compile(PASSWORD_PATTERN);
    matcher = pattern.matcher(changePassword_text.getText().toString());
    // Check if the password is complex enough
    boolean isStrong= matcher.matches();
    if (isStrong){
        responsebody = httpClient.execute(httpPost);
        InputStream in = responsebody.getEntity().getContent();
        result = convertStreamToString( in );
        result = result.replace( target: "\n", replacement: "" );
    }
}

```

The screenshot shows a POST request to `http://10.22.162.74:8888/changepassword`. The request body is set to `form-data` and contains two fields: `username` (value: `jack`) and `newpassword` (value: `password`). The response body is displayed in Pretty format, showing a JSON object with the key `message` and the value `"Change Password Successful"`.

Conclusion

According to above vulnerability analysis, there are many security issues found that this bank confidential information leak or attackable to attacker. So before release this application to production, it must be fixed above all issues and better to do proper code review.

***** End *****