Android Mobile Pentest 101

Lecture 4 – Reversing The App

Goal: Got some basic reverse skill on android app

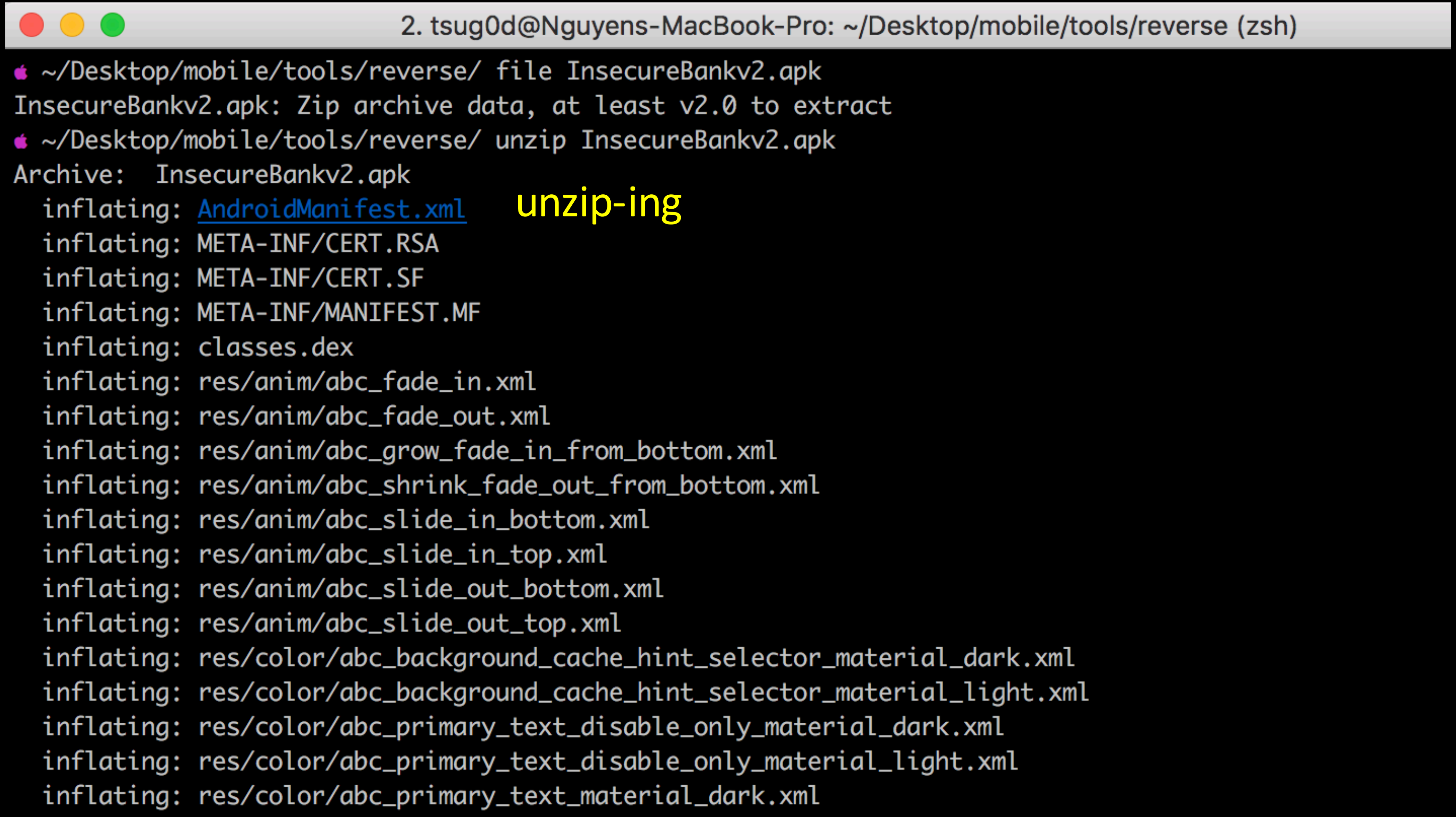
Inside the apk

- APK (Android application package) is an app creator for Android, it contains all the elements that an app needs to install correctly on mobile device
- Like EXE on windows, you can place APK file on mobile device to install. Manually installing apps using APK is called sideloading
- Below is a list describing the most prominent files and folders:

* **META-INF/:** Contains the manifest file, signature, and a list of resources in the archive
* **lib/:** Native libraries that run on specific device architectures (armeabi-v7a, x86, etc.)
* **res/:** Resources, such as images, that were not compiled into **resources.arsc**
* **AndroidManifest.xml**: Describes the name, version, and contents of the APK file
* **classes.dex**: The compiled Java classes to be run on the device (.DEX file)
* **resources.arsc**: The compiled resources, such as strings, used by the app (.ARSC file)

Inside the apk

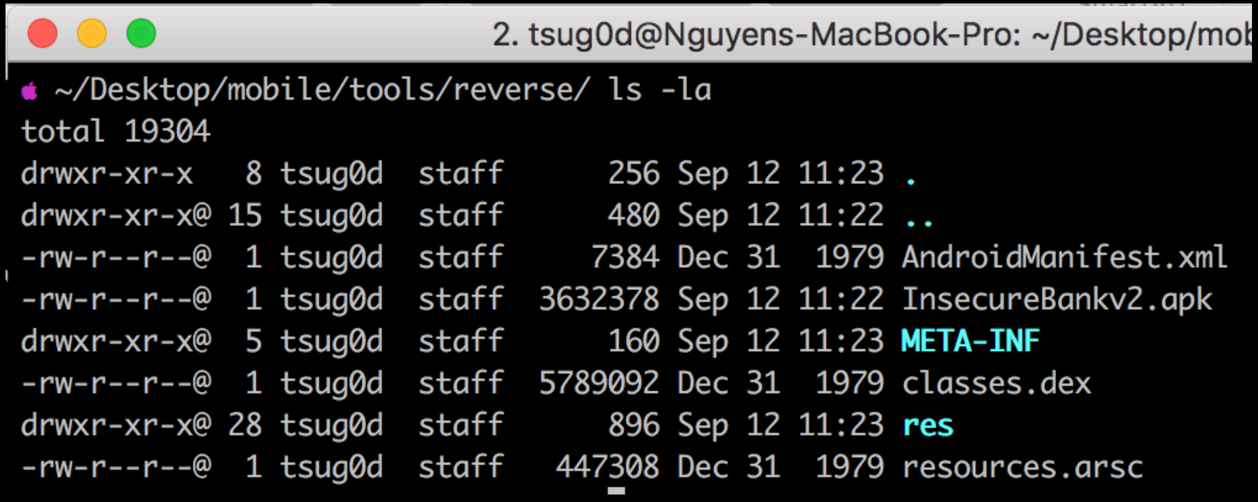
- APK files are a type of archive file, specifically in zip format-type packages, based on the JAR file format, with .apk as the filename extension.
- So we first unzip it:


2. tsug0d@Nguyens-MacBook-Pro: ~/Desktop/mobile/tools/reverse (zsh)
unzip-ing

```
 ~/Desktop/mobile/tools/reverse/ file InsecureBankv2.apk
InsecureBankv2.apk: Zip archive data, at least v2.0 to extract
 ~/Desktop/mobile/tools/reverse/ unzip InsecureBankv2.apk
Archive:  InsecureBankv2.apk
  inflating: AndroidManifest.xml
  inflating: META-INF/CERT.RSA
  inflating: META-INF/CERT.SF
  inflating: META-INF/MANIFEST.MF
  inflating: classes.dex
  inflating: res/anim/abc_fade_in.xml
  inflating: res/anim/abc_fade_out.xml
  inflating: res/anim/abc_grow_fade_in_from_bottom.xml
  inflating: res/anim/abc_shrink_fade_out_from_bottom.xml
  inflating: res/anim/abc_slide_in_bottom.xml
  inflating: res/anim/abc_slide_in_top.xml
  inflating: res/anim/abc_slide_out_bottom.xml
  inflating: res/anim/abc_slide_out_top.xml
  inflating: res/color/abc_background_cache_hint_selector_material_dark.xml
  inflating: res/color/abc_background_cache_hint_selector_material_light.xml
  inflating: res/color/abc_primary_text_disable_only_material_dark.xml
  inflating: res/color/abc_primary_text_disable_only_material_light.xml
  inflating: res/color/abc_primary_text_material_dark.xml
```

Inside the apk

- We can see what we said above



2. tsug0d@Nguyens-MacBook-Pro: ~/Desktop/mob

```
 ~/Desktop/mobile/tools/reverse/ ls -la
total 19304
drwxr-xr-x   8 tsug0d  staff      256 Sep 12 11:23 .
drwxr-xr-x@ 15 tsug0d  staff      480 Sep 12 11:22 ..
-rw-r--r--@  1 tsug0d  staff     7384 Dec 31  1979 AndroidManifest.xml
-rw-r--r--@  1 tsug0d  staff  3632378 Sep 12 11:22 InsecureBankv2.apk
drwxr-xr-x@  5 tsug0d  staff      160 Sep 12 11:23 META-INF
-rw-r--r--@  1 tsug0d  staff  5789092 Dec 31  1979 classes.dex
drwxr-xr-x@ 28 tsug0d  staff      896 Sep 12 11:23 res
-rw-r--r--@  1 tsug0d  staff   447308 Dec 31  1979 resources.arsc
```

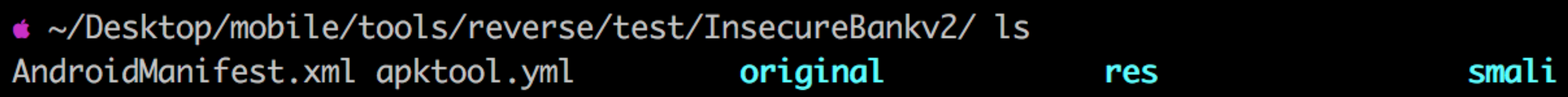
- But it’s not good enough for our reversing phase (because it’s just unzip, and most of the file is in binary format).
- Let Decompile it!

Inside the apk

- APKTool is the very first tool you want to use, it is capable of decompiling the AndroidManifest file to its original XML format, the resources.arsc file and it will also convert the classes.dex (and classes2.dex if present) file to an intermediary language called SMALI, an ASM-like language used to represent the Dalvik VM opcodes as a human readable language.
- Type command:

```
apktool d InsecureBankv2.apk
```

- Once finished, the InsecureBankv2 folder is created and you'll find all the output of apktool in there (included smali code).



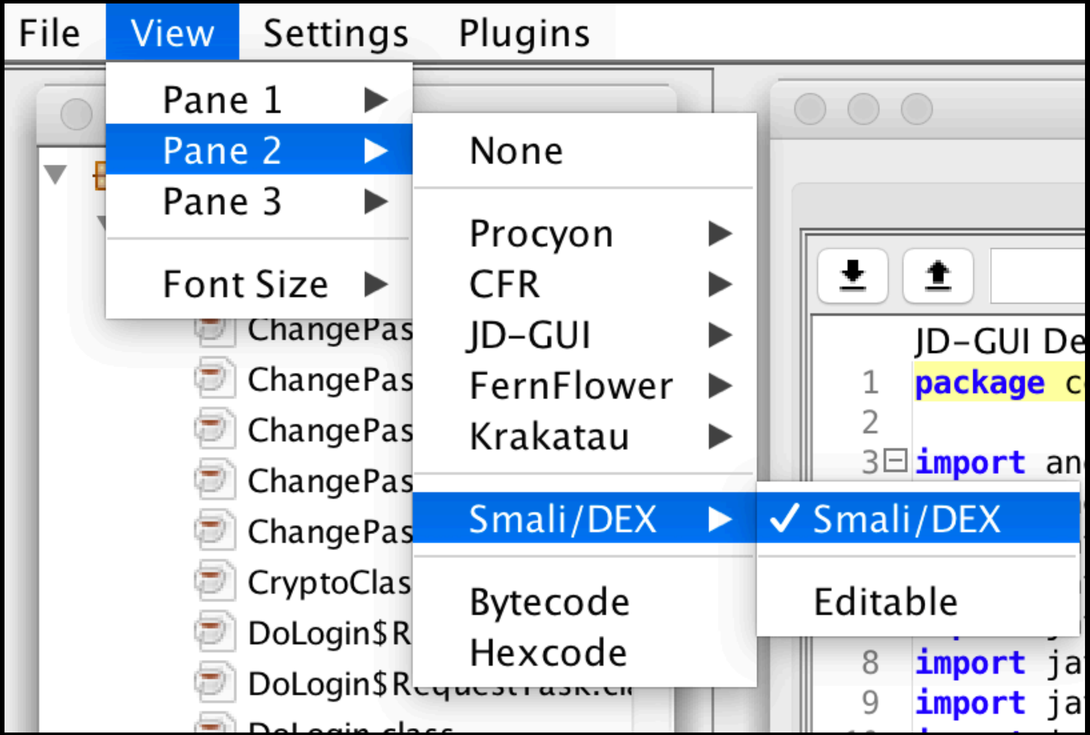
```
~/Desktop/mobile/tools/reverse/test/InsecureBankv2/ ls
AndroidManifest.xml apktool.yml          original             res                  smali
```

- You can also get java source code using dex2jar (output is jar file)
- Read the java source code in jar using jd-gui

Just inform the basic, no need to follow, we can do all the task with ByteCode Viewer (remember? ☺)

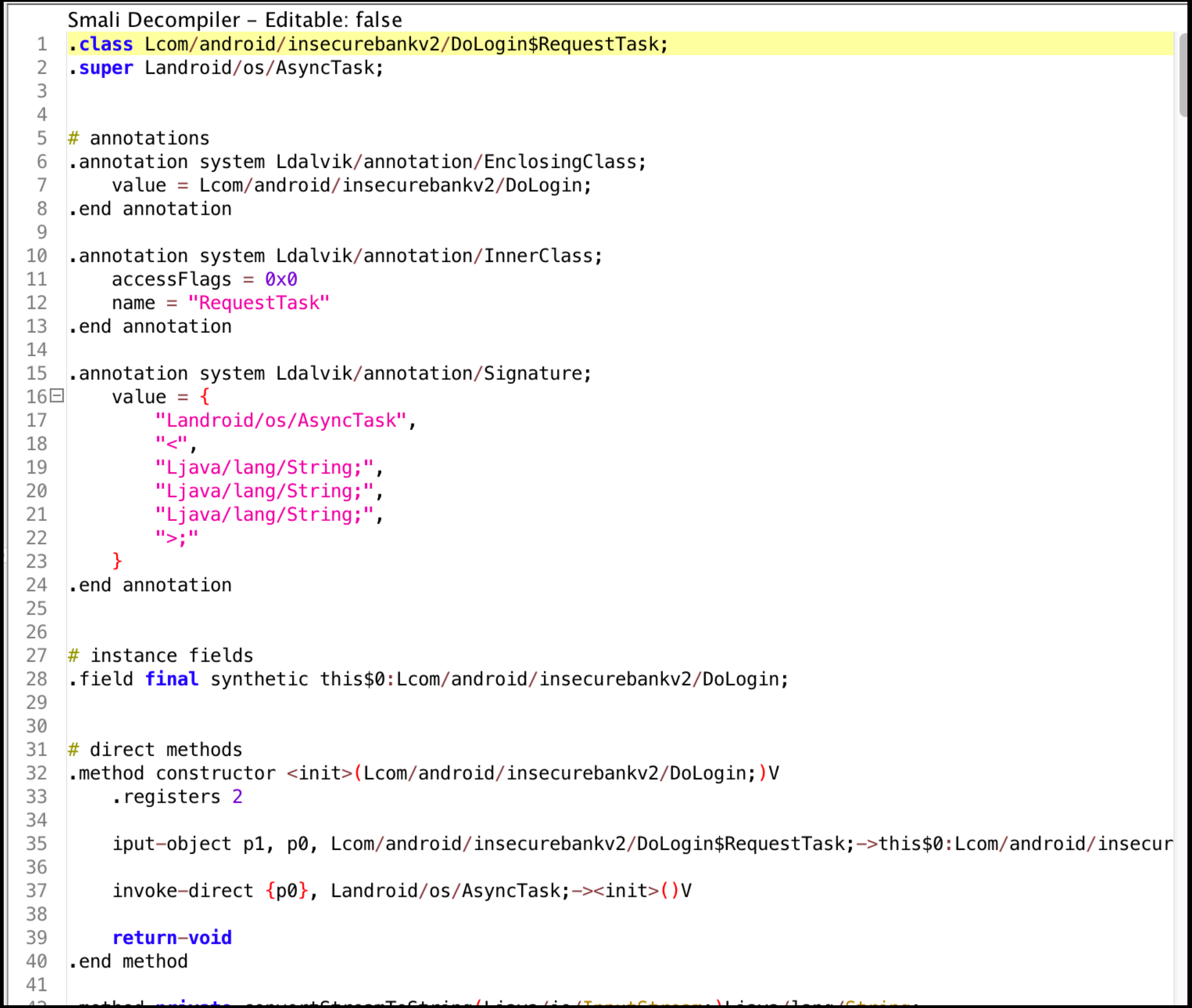
smali

- We will talk about SMALI, let look at some smali code
- Load the apk to ByteCode Viewer
- Choose like this:



smali

- Now open Insecurebankv2.apk/com/android/insecurebankv2/DoLogin$RequestTask.class, we got smali code in panel 2

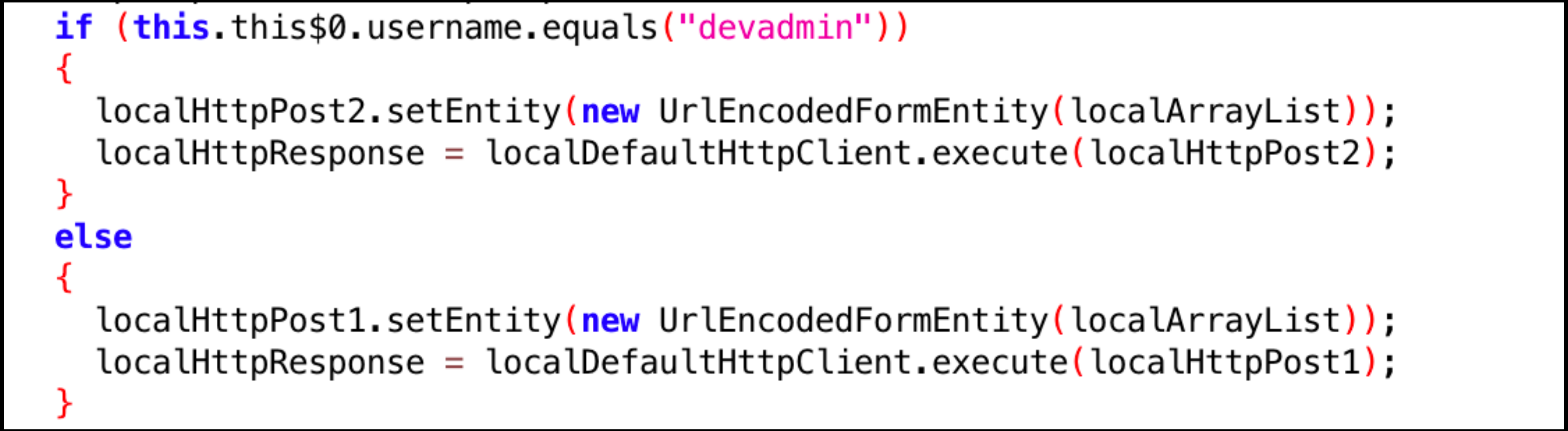


Smali Decompiler – Editable: false

```
.class Lcom/android/insecurebankv2/DoLogin$RequestTask;
.super Landroid/os/AsyncTask;


# annotations
.annotation system Ldalvik/annotation/EnclosingClass;
    value = Lcom/android/insecurebankv2/DoLogin;
.end annotation

.annotation system Ldalvik/annotation/InnerClass;
    accessFlags = 0x0
    name = "RequestTask"
.end annotation

.annotation system Ldalvik/annotation/Signature;
    value = {
        "Landroid/os/AsyncTask",
        "<",
        "Ljava/lang/String;",
        "Ljava/lang/String;",
        "Ljava/lang/String;",
        ">;"
    }
.end annotation


# instance fields
.field final synthetic this$0:Lcom/android/insecurebankv2/DoLogin;


# direct methods
.method constructor <init>(Lcom/android/insecurebankv2/DoLogin;)V
    .registers 2

    iput-object p1, p0, Lcom/android/insecurebankv2/DoLogin$RequestTask;->this$0:Lcom/android/insecur

    invoke-direct {p0}, Landroid/os/AsyncTask;-><init>()V

    return-void
.end method

```

smali

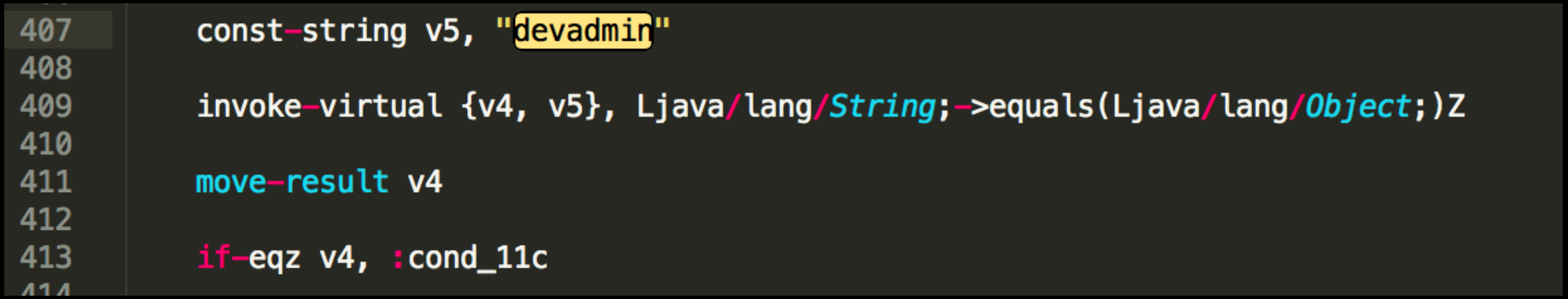
- Remember the “devadmin” bypass login? We will take a look at it in java bytecode (smali) ☺



```
if (this.this$0.username.equals("devadmin"))
{
  localHttpPost2.setEntity(new UrlEncodedFormEntity(localArrayList));
  localHttpResponse = localDefaultHttpClient.execute(localHttpPost2);
}
else
{
  localHttpPost1.setEntity(new UrlEncodedFormEntity(localArrayList));
  localHttpResponse = localDefaultHttpClient.execute(localHttpPost1);
}
```

smali

- So we first find this string in smali code



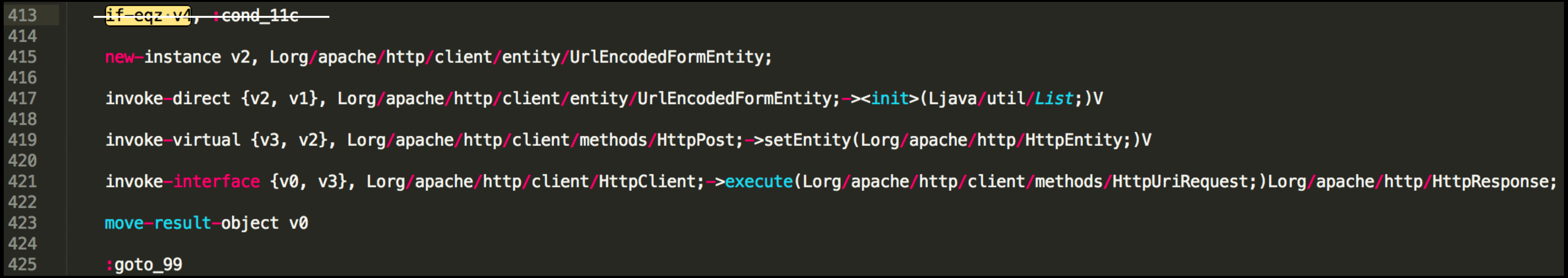
```
    const-string v5, "devadmin"

    invoke-virtual {v4, v5}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

    move-result v4

    if-eqz v4, :cond_11c
```

- What it does is compare our input (v4) with the string "devadmin" (v5) by calling Java.lang.String.equals() method, if equal return 1, else 0
- Then the result is saved in v4
- The program call the condition if-eqz (if equal zero), so if the input is not "devadmin", it goes to :cond_11c

smali

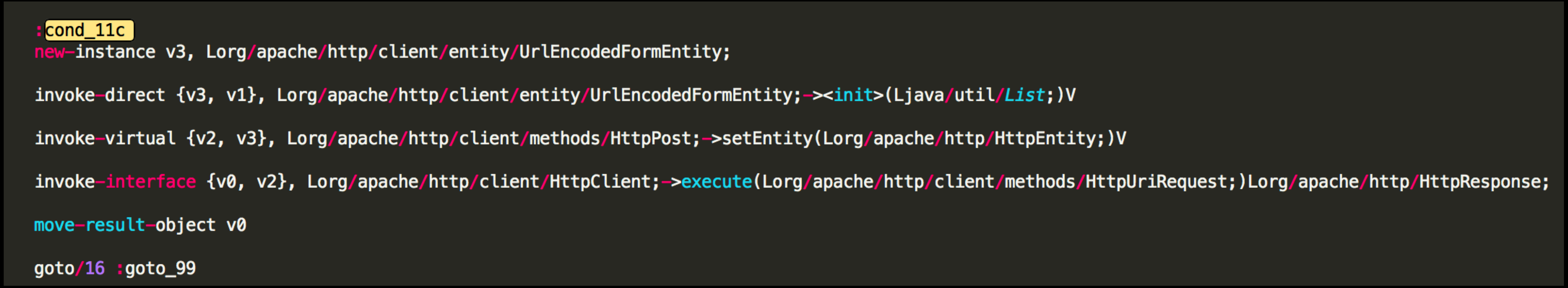
- We look the code after if-eqz fail and the code at :cond_11c

Equals to “devadmin”



```
~~if-eqz v4, :cond_11c~~

new-instance v2, Lorg/apache/http/client/entity/UrlEncodedFormEntity;

invoke-direct {v2, v1}, Lorg/apache/http/client/entity/UrlEncodedFormEntity;-><init>(Ljava/util/List;)V

invoke-virtual {v3, v2}, Lorg/apache/http/client/methods/HttpPost;->setEntity(Lorg/apache/http/HttpEntity;)V

invoke-interface {v0, v3}, Lorg/apache/http/client/HttpClient;->execute(Lorg/apache/http/client/methods/HttpUriRequest;)Lorg/apache/http/HttpResponse;

move-result-object v0

:goto_99
```

Not Equals to “dev-admin”

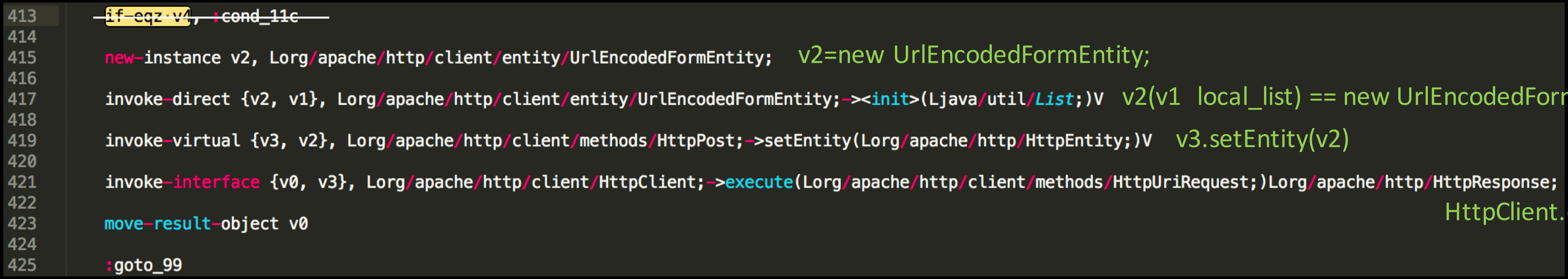


```
:cond_11c
new-instance v3, Lorg/apache/http/client/entity/UrlEncodedFormEntity;

invoke-direct {v3, v1}, Lorg/apache/http/client/entity/UrlEncodedFormEntity;-><init>(Ljava/util/List;)V

invoke-virtual {v2, v3}, Lorg/apache/http/client/methods/HttpPost;->setEntity(Lorg/apache/http/HttpEntity;)V

invoke-interface {v0, v2}, Lorg/apache/http/client/HttpClient;->execute(Lorg/apache/http/client/methods/HttpUriRequest;)Lorg/apache/http/HttpResponse;

move-result-object v0

goto/16 :goto_99
```

Code Looks the same ☺

smali

- Evaluate equal condition:



```
~~if-eqz v4, :cond_11c~~

new-instance v2, Lorg/apache/http/client/entity/UrlEncodedFormEntity;

invoke-direct {v2, v1}, Lorg/apache/http/client/entity/UrlEncodedFormEntity;-><init>(Ljava/util/List;)V

invoke-virtual {v3, v2}, Lorg/apache/http/client/methods/HttpPost;->setEntity(Lorg/apache/http/HttpEntity;)V

invoke-interface {v0, v3}, Lorg/apache/http/client/HttpClient;->execute(Lorg/apache/http/client/methods/HttpUriRequest;)Lorg/apache/http/HttpResponse;

move-result-object v0

:goto_99
```

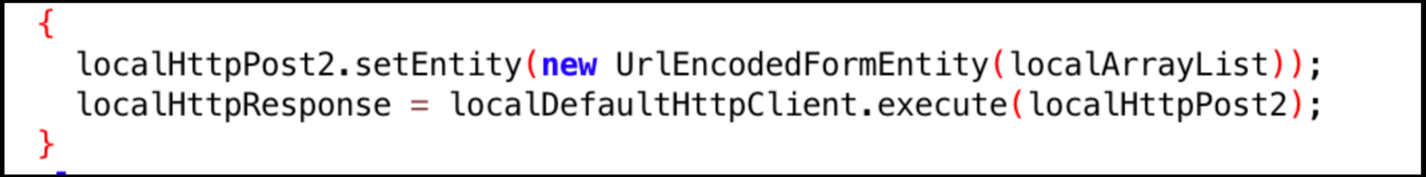
v2=new UrlEncodedFormEntity;

v2(v1 local_list) == new UrlEncodedFormEntity(v1)

v3.setEntity(v2)

HttpClient.execute(v3)

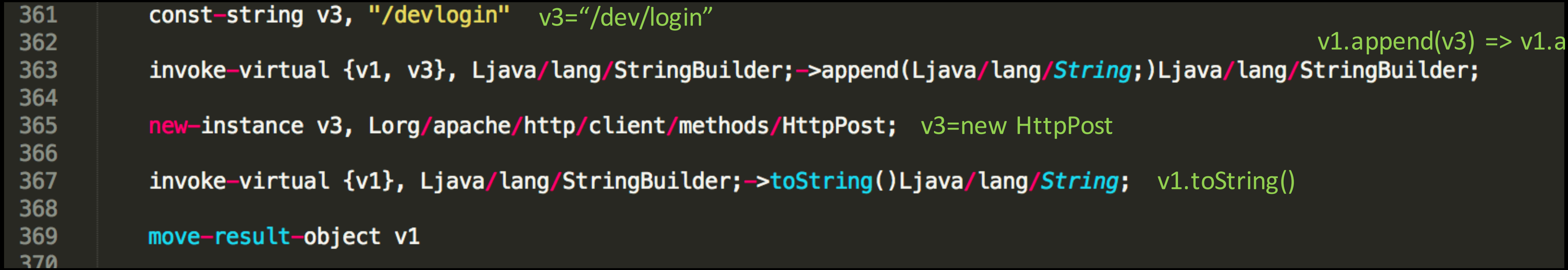
- The above code is equal to in java source



```
{
  localHttpPost2.setEntity(new UrlEncodedFormEntity(localArrayList));
  localHttpResponse = localDefaultHttpClient.execute(localHttpPost2);
}
```

So what v3 is? Next-slide ☺

smali



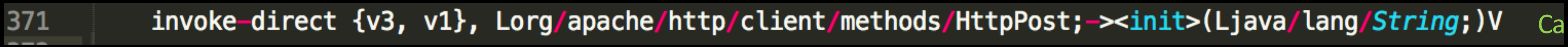
```
const-string v3, "/devlogin"

invoke-virtual {v1, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

new-instance v3, Lorg/apache/http/client/methods/HttpPost;

invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v1
```

v3="/dev/login"

v1.append(v3) => v1.append('/dev/login')

v3=new HttpPost

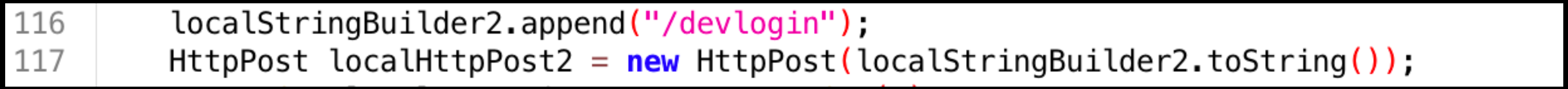
v1.toString()



```
invoke-direct {v3, v1}, Lorg/apache/http/client/methods/HttpPost;-><init>(Ljava/lang/String;)V
```

Call HttpPost(v1)

- The above code is equal to in java source



```
localStringBuilder2.append("/devlogin");
HttpPost localHttpPost2 = new HttpPost(localStringBuilder2.toString());
```

- Basically, if we input "devadmin", it will do POST request to http://mobile-server/devlogin which allow us to free login ☺

smali

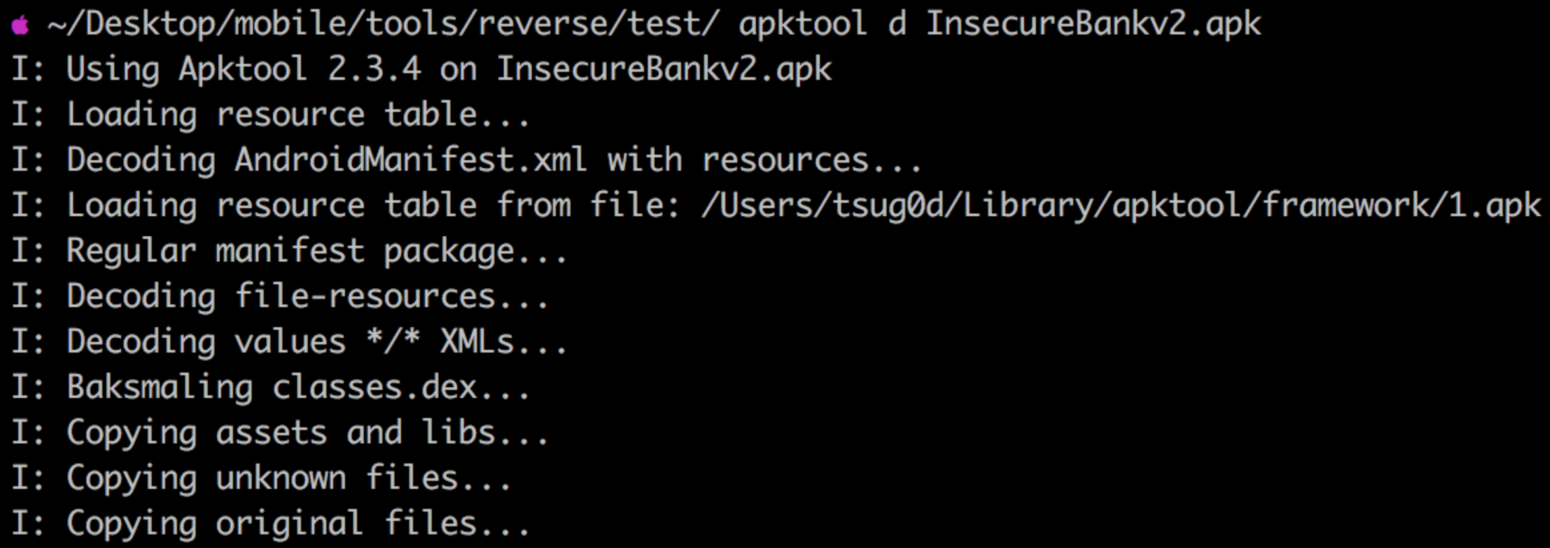
- So you may ask why should we have to read the smali code? Why won’t we read the clear java source code instead?

The answer is, we have to detect it, to patch the program! (talk later)

Patching android app

- Now we will talk about how to patch an android app (no smali in this talk, great ☺)
- A **patch** is a software update comprised code inserted (or **patched**) into the code of an executable program.
- Remember apktool? We use it to decompile the apk first

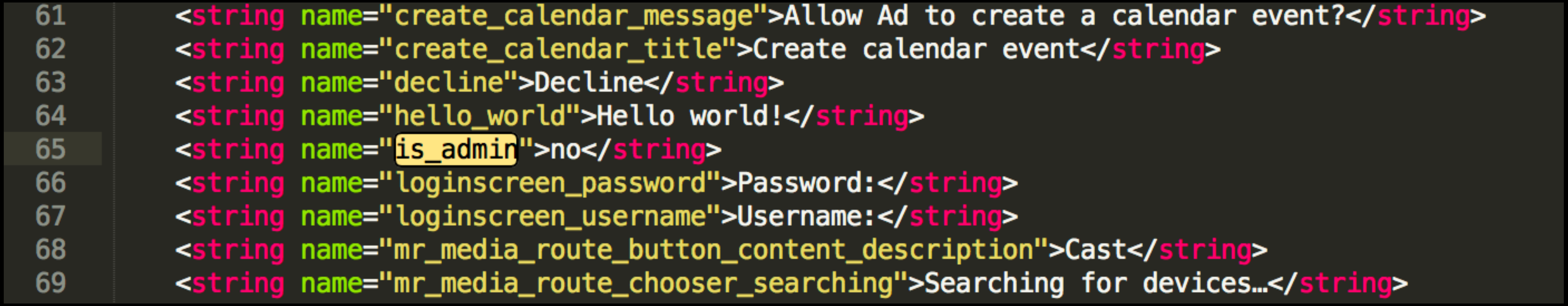
apktool d InsecureBankv2.apk



```
 ~/Desktop/mobile/tools/reverse/test/ apktool d InsecureBankv2.apk
I: Using Apktool 2.3.4 on InsecureBankv2.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /Users/tsug0d/Library/apktool/framework/1.apk
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...
```

Patching android app

- So we go to res folder to find something interesting (why? Because this folder contains resource, and modify it is fun ☺)
- Quickly found this in InsecureBankv2/res/values/strings.xml

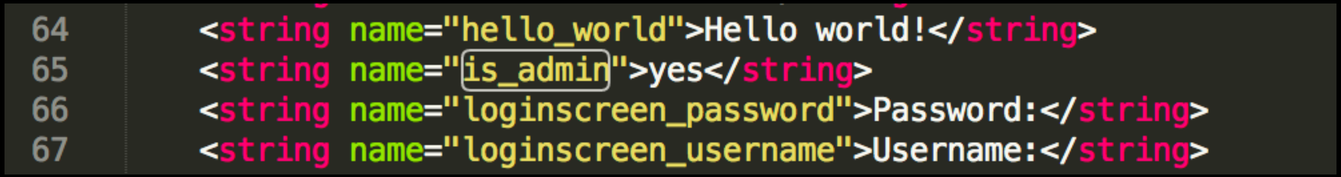


```
    <string name="create_calendar_message">Allow Ad to create a calendar event?</string>
    <string name="create_calendar_title">Create calendar event</string>
    <string name="decline">Decline</string>
    <string name="hello_world">Hello world!</string>
    <string name="is_admin">no</string>
    <string name="loginscreen_password">Password:</string>
    <string name="loginscreen_username">Username:</string>
    <string name="mr_media_route_button_content_description">Cast</string>
    <string name="mr_media_route_chooser_searching">Searching for devices…</string>
```

- If you want to be a good pentester, you must keep in your mind:

"If someone say no, punch him, and say yes instead ☺"

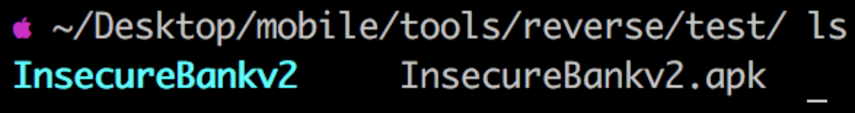
- So modify it to yes



```
    <string name="hello_world">Hello world!</string>
    <string name="is_admin">yes</string>
    <string name="loginscreen_password">Password:</string>
    <string name="loginscreen_username">Username:</string>
```

Patching android app

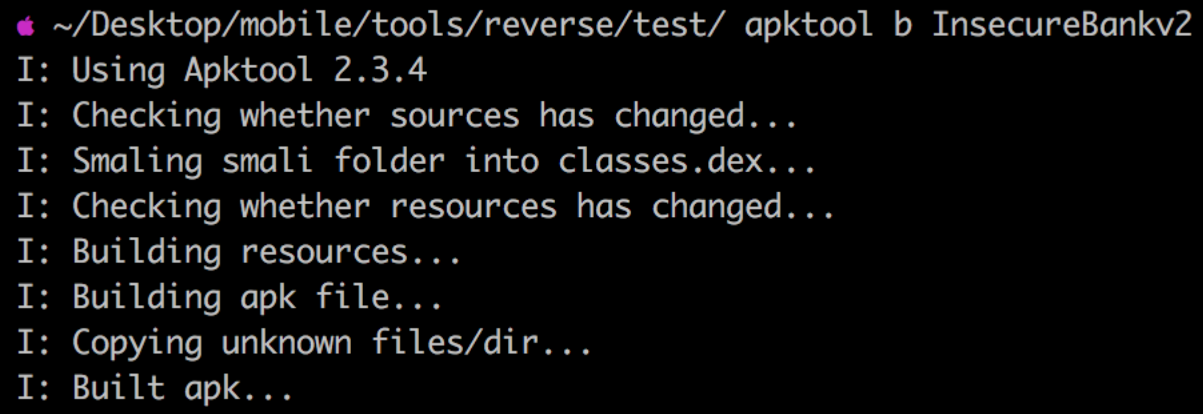
- We go back to base folder



```
~/Desktop/mobile/tools/reverse/test/ ls
InsecureBankv2      InsecureBankv2.apk
```

- And re-compile the application:

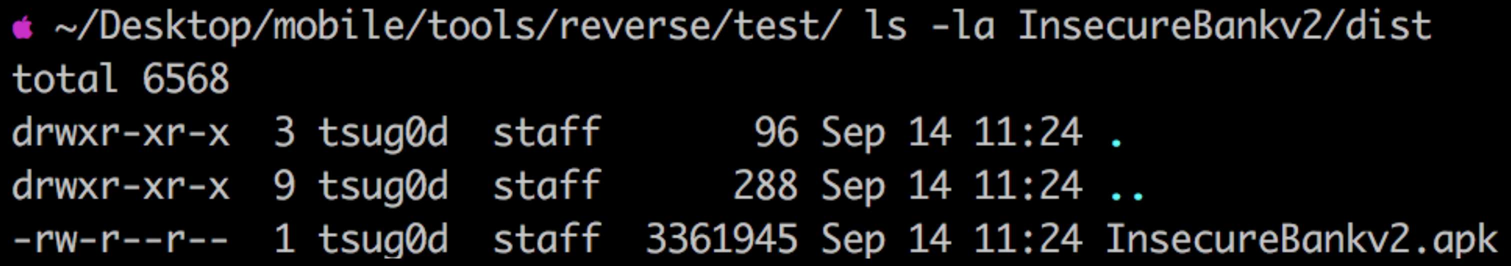
apktool b InsecureBankv2



```
~/Desktop/mobile/tools/reverse/test/ apktool b InsecureBankv2
I: Using Apktool 2.3.4
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
I: Building resources...
I: Building apk file...
I: Copying unknown files/dir...
I: Built apk...
```

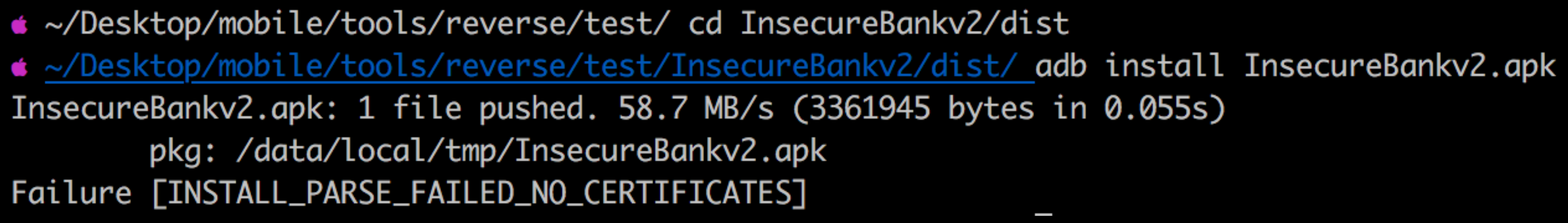
Patching android app

- Note that the re-compile apk is located in “dist” folder, not the old apk in current folder.



```
~/Desktop/mobile/tools/reverse/test/ ls -la InsecureBankv2/dist
total 6568
drwxr-xr-x  3 tsug0d  staff       96 Sep 14 11:24 .
drwxr-xr-x  9 tsug0d  staff      288 Sep 14 11:24 ..
-rw-r--r--  1 tsug0d  staff  3361945 Sep 14 11:24 InsecureBankv2.apk
```

- So install it?



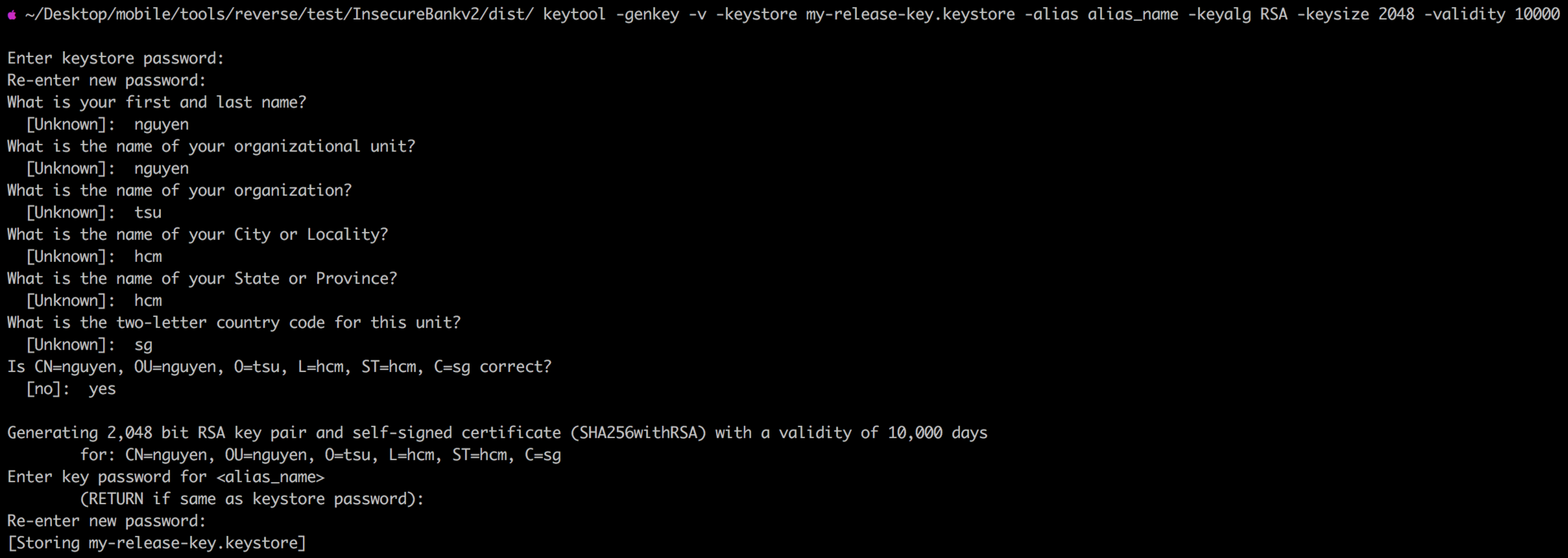
```
~/Desktop/mobile/tools/reverse/test/ cd InsecureBankv2/dist
~/Desktop/mobile/tools/reverse/test/InsecureBankv2/dist/ adb install InsecureBankv2.apk
InsecureBankv2.apk: 1 file pushed. 58.7 MB/s (3361945 bytes in 0.055s)
        pkg: /data/local/tmp/InsecureBankv2.apk
Failure [INSTALL_PARSE_FAILED_NO_CERTIFICATES]
```

- Oops, Failed! Its because Every new compiled Android .apk needs to be signed if it is going to be installed on a phone

Patching android app

- We are going to sign it!
- Create key using:

keytool -genkey -v -keystore my-release-key.keystore -alias alias_name -keyalg RSA -keysize 2048 -validity 10000

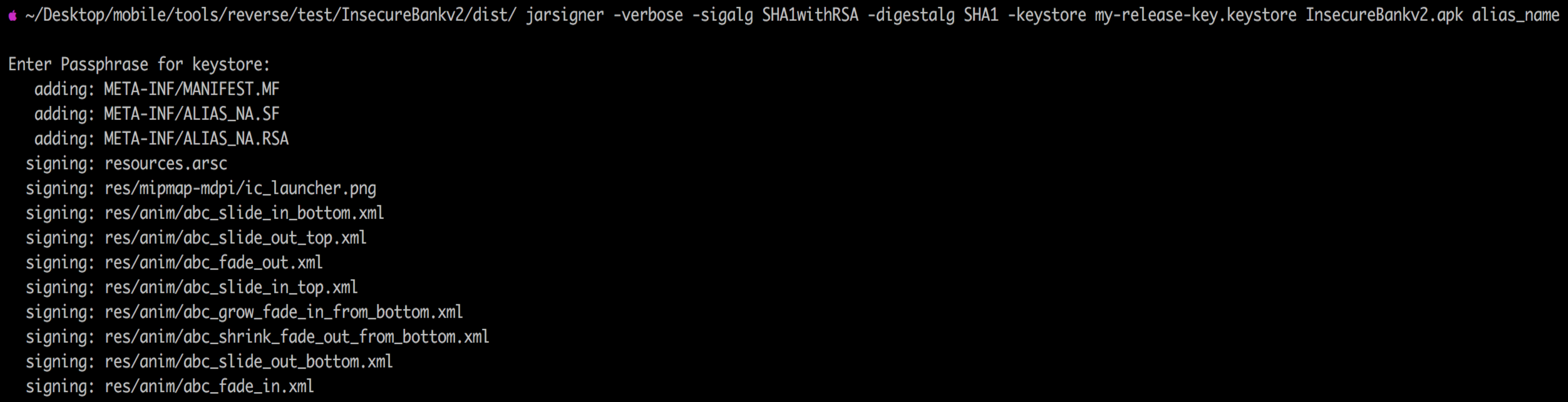


```
~/Desktop/mobile/tools/reverse/test/InsecureBankv2/dist/ keytool -genkey -v -keystore my-release-key.keystore -alias alias_name -keyalg RSA -keysize 2048 -validity 10000

Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  nguyen
What is the name of your organizational unit?
  [Unknown]:  nguyen
What is the name of your organization?
  [Unknown]:  tsu
What is the name of your City or Locality?
  [Unknown]:  hcm
What is the name of your State or Province?
  [Unknown]:  hcm
What is the two-letter country code for this unit?
  [Unknown]:  sg
Is CN=nguyen, OU=nguyen, O=tsu, L=hcm, ST=hcm, C=sg correct?
  [no]:  yes

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 10,000 days
        for: CN=nguyen, OU=nguyen, O=tsu, L=hcm, ST=hcm, C=sg
Enter key password for <alias_name>
        (RETURN if same as keystore password):
Re-enter new password:
[Storing my-release-key.keystore]
```

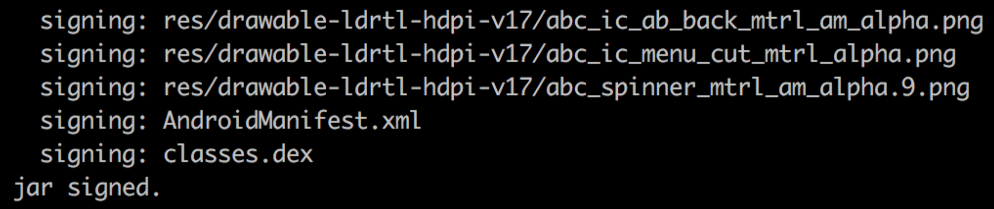
Patching android app

- Sign using:

jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-key.keystore InsecureBankv2.apk alias_name



```
 ~/Desktop/mobile/tools/reverse/test/InsecureBankv2/dist/ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-key.keystore InsecureBankv2.apk alias_name

Enter Passphrase for keystore:
   adding: META-INF/MANIFEST.MF
   adding: META-INF/ALIAS_NA.SF
   adding: META-INF/ALIAS_NA.RSA
  signing: resources.arsc
  signing: res/mipmap-mdpi/ic_launcher.png
  signing: res/anim/abc_slide_in_bottom.xml
  signing: res/anim/abc_slide_out_top.xml
  signing: res/anim/abc_fade_out.xml
  signing: res/anim/abc_slide_in_top.xml
  signing: res/anim/abc_grow_fade_in_from_bottom.xml
  signing: res/anim/abc_shrink_fade_out_from_bottom.xml
  signing: res/anim/abc_slide_out_bottom.xml
  signing: res/anim/abc_fade_in.xml
```



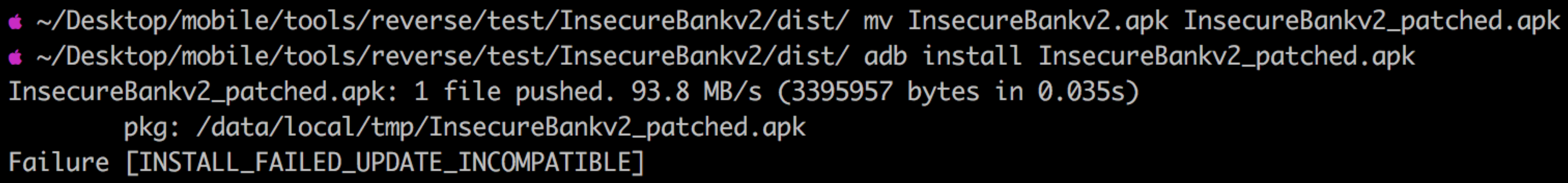
```
  signing: res/drawable-ldrtl-hdpi-v17/abc_ic_ab_back_mtrl_am_alpha.png
  signing: res/drawable-ldrtl-hdpi-v17/abc_ic_menu_cut_mtrl_alpha.png
  signing: res/drawable-ldrtl-hdpi-v17/abc_spinner_mtrl_am_alpha.9.png
  signing: AndroidManifest.xml
  signing: classes.dex
jar signed.
```

Patching android app

- We change its name to make it clearer compare to original apk:

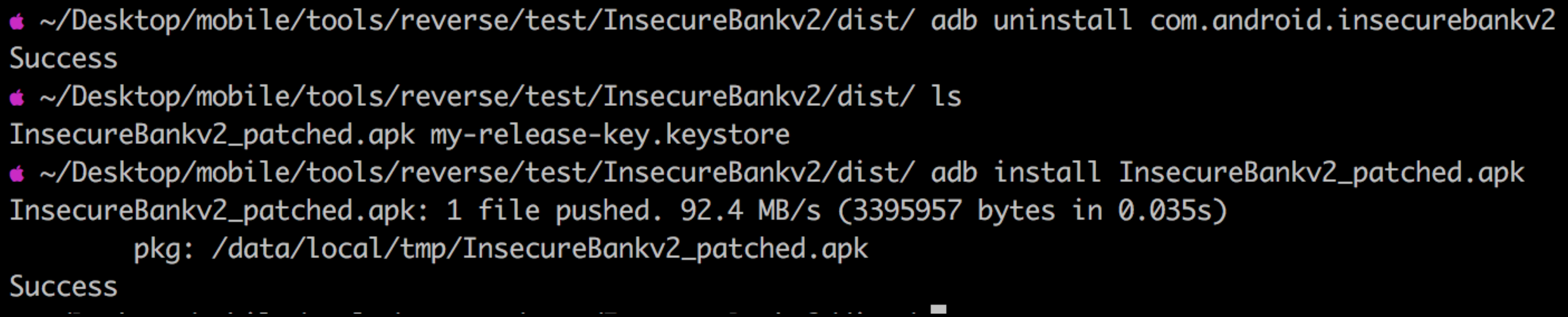
mv InsecureBankv2.apk InsecureBankv2_patched.apk

- Then install



```
~/Desktop/mobile/tools/reverse/test/InsecureBankv2/dist/ mv InsecureBankv2.apk InsecureBankv2_patched.apk
~/Desktop/mobile/tools/reverse/test/InsecureBankv2/dist/ adb install InsecureBankv2_patched.apk
InsecureBankv2_patched.apk: 1 file pushed. 93.8 MB/s (3395957 bytes in 0.035s)
        pkg: /data/local/tmp/InsecureBankv2_patched.apk
Failure [INSTALL_FAILED_UPDATE_INCOMPATIBLE]
```

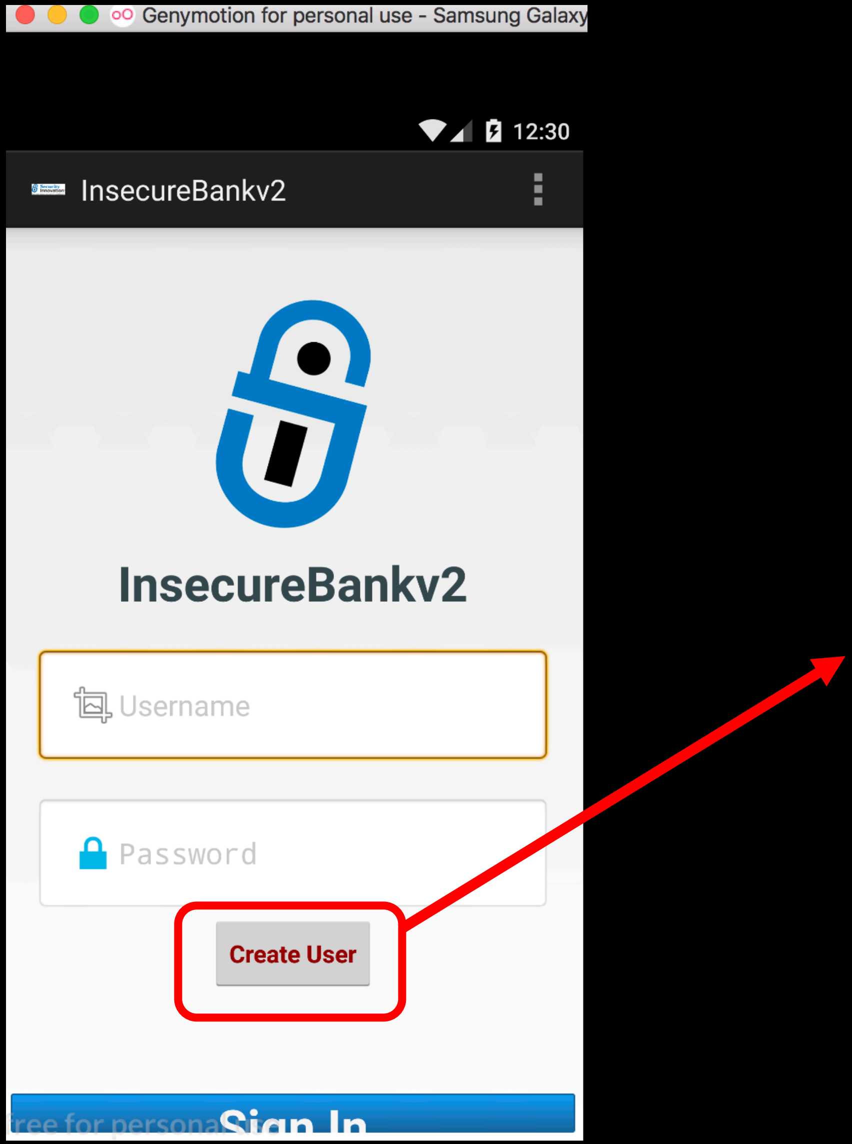
- Still fail? It means the application which you want to install is already installed. just remove the old one and try again.



```
~/Desktop/mobile/tools/reverse/test/InsecureBankv2/dist/ adb uninstall com.android.insecurebankv2
Success
~/Desktop/mobile/tools/reverse/test/InsecureBankv2/dist/ ls
InsecureBankv2_patched.apk my-release-key.keystore
~/Desktop/mobile/tools/reverse/test/InsecureBankv2/dist/ adb install InsecureBankv2_patched.apk
InsecureBankv2_patched.apk: 1 file pushed. 92.4 MB/s (3395957 bytes in 0.035s)
        pkg: /data/local/tmp/InsecureBankv2_patched.apk
Success
```

Patching android app

- Open it again:



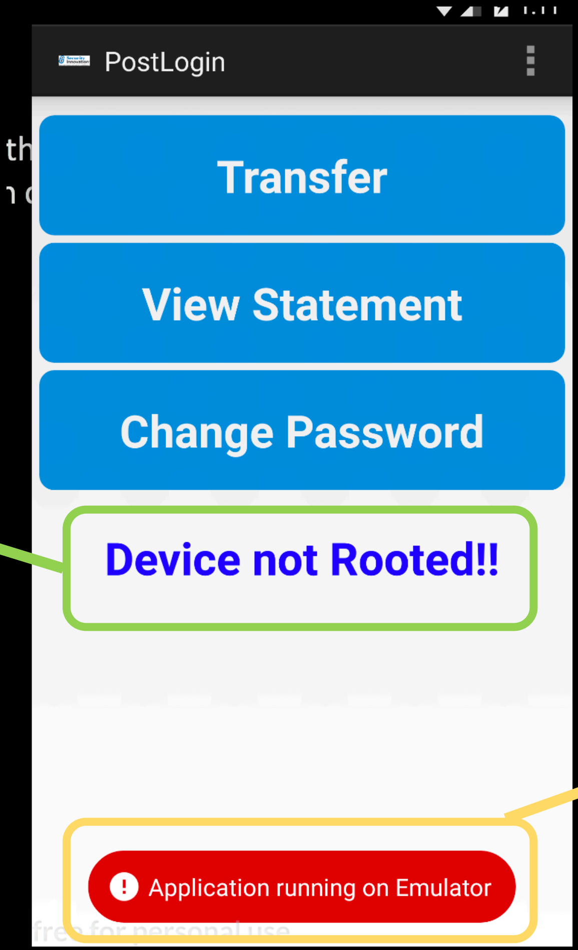
Good one, we got Create User button! That's admin module ☺

Detection

- "Life is not easy", it's true, at least in this lecture ☺
- The application can contain detection code, to prevent attacker analysis it, usually are:

1. Root Detection (or anti-root)
2. Emulator Detection (or anti-vm)

- Back to the app, we see this:

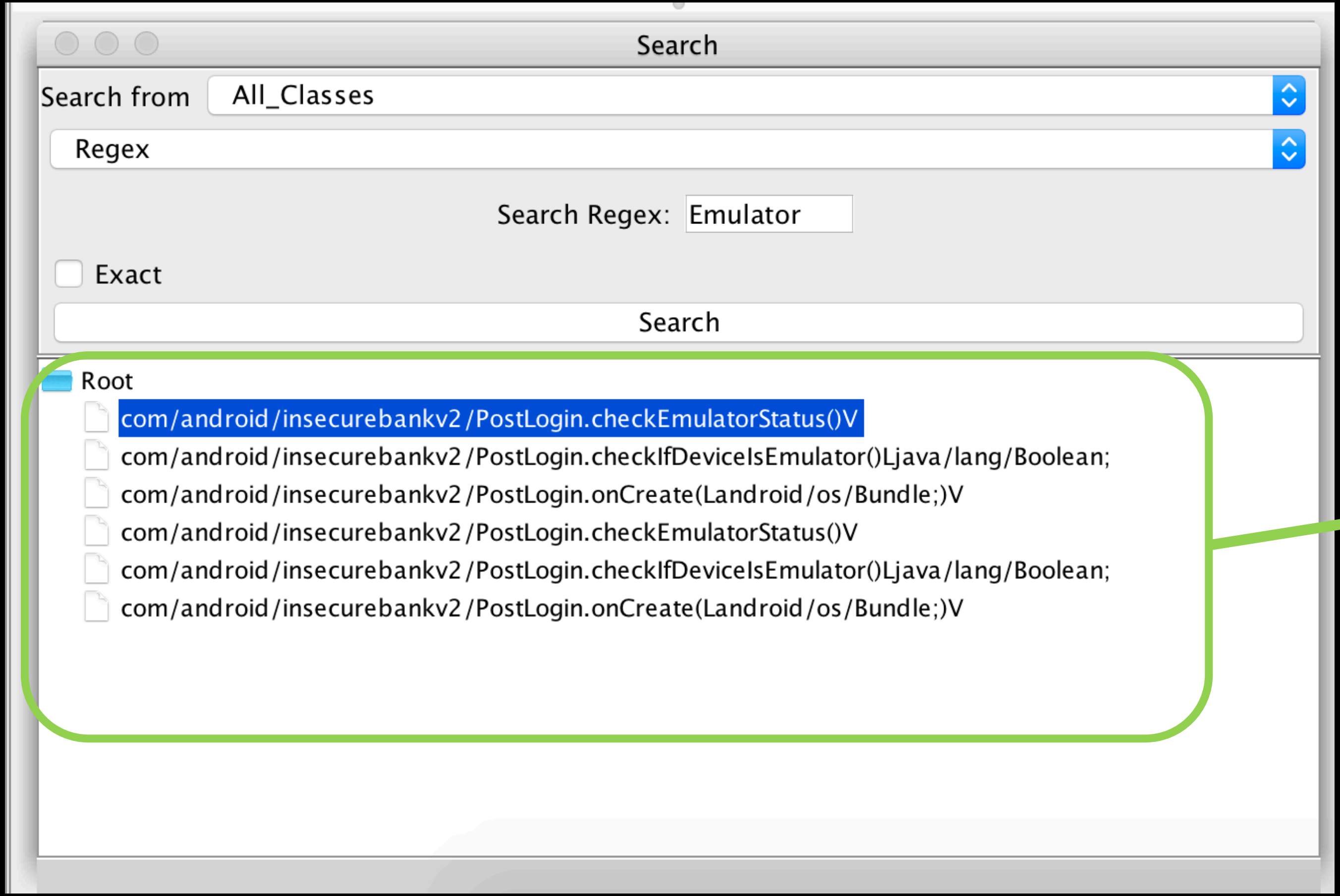


It got root detection code, but the code fail, so we are safe. I'll talk later ☺

Emulator detected, we are running this app on galaxy s6 based on genymotion right?

Detection -> Anti-emu

- So we come to Anti-emu bypass. We saw this alert on PostLogin section, so we check its code
- Or we can search it => Go to Bytecode Viewer, load the apk and search for "Emulator"



Search

Search from: All_Classes

Regex

Search Regex: Emulator

Exact

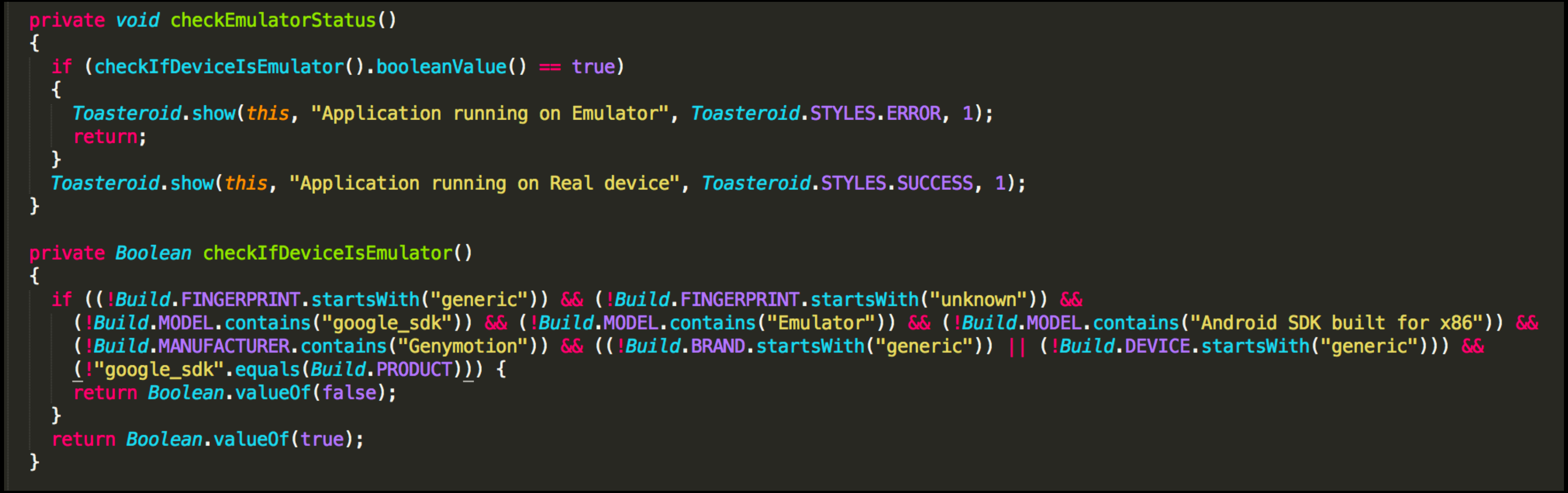
Search

```
Root
  com/android/insecurebankv2/PostLogin.checkEmulatorStatus()V
  com/android/insecurebankv2/PostLogin.checkIfDeviceIsEmulator()Ljava/lang/Boolean;
  com/android/insecurebankv2/PostLogin.onCreate(Landroid/os/Bundle;)V
  com/android/insecurebankv2/PostLogin.checkEmulatorStatus()V
  com/android/insecurebankv2/PostLogin.checkIfDeviceIsEmulator()Ljava/lang/Boolean;
  com/android/insecurebankv2/PostLogin.onCreate(Landroid/os/Bundle;)V
```

Search result!

Detection -> Anti-emu

- We found the code in com/android/insecurebankv2/PostLogin.class

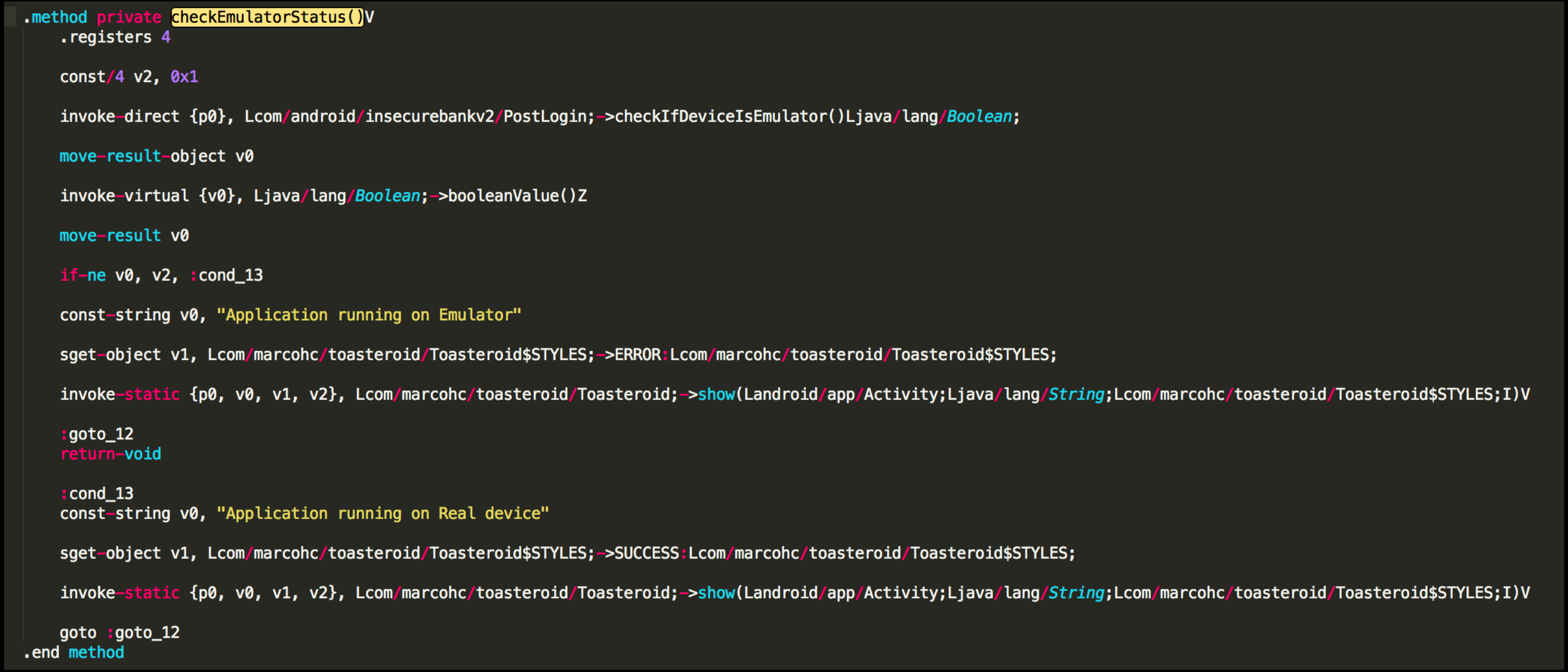


```
private void checkEmulatorStatus()
{
  if (checkIfDeviceIsEmulator().booleanValue() == true)
  {
    Toasteroid.show(this, "Application running on Emulator", Toasteroid.STYLES.ERROR, 1);
    return;
  }
  Toasteroid.show(this, "Application running on Real device", Toasteroid.STYLES.SUCCESS, 1);
}

private Boolean checkIfDeviceIsEmulator()
{
  if ((!Build.FINGERPRINT.startsWith("generic")) && (!Build.FINGERPRINT.startsWith("unknown")) &&
    (!Build.MODEL.contains("google_sdk")) && (!Build.MODEL.contains("Emulator")) && (!Build.MODEL.contains("Android SDK built for x86")) &&
    (!Build.MANUFACTURER.contains("Genymotion")) && ((!Build.BRAND.startsWith("generic")) || (!Build.DEVICE.startsWith("generic"))) &&
    (!"google_sdk".equals(Build.PRODUCT))) {
    return Boolean.valueOf(false);
  }
  return Boolean.valueOf(true);
}
```

- We can see the checkEmulatorStatus() call checkIfDeviceIsEmulator(), this function find the string defined in FINGERPRINT, MODEL, MANUFACTURE, BRAND, DEVICE, etc… to detect, in our case: “Genymotion”
- This time not easy patch like before, because this is code check, if we want to patch, we must modify SMALI (yes, smali again ☺).

Detection -> Anti-emu

- So step one is to find the smali code match with anti-emu check in java code
- We already known the check is the function checkEmulatorStatus(), search for it ☺



```
.method private checkEmulatorStatus()V
    .registers 4

    const/4 v2, 0x1

    invoke-direct {p0}, Lcom/android/insecurebankv2/PostLogin;->checkIfDeviceIsEmulator()Ljava/lang/Boolean;

    move-result-object v0

    invoke-virtual {v0}, Ljava/lang/Boolean;->booleanValue()Z

    move-result v0

    if-ne v0, v2, :cond_13

    const-string v0, "Application running on Emulator"

    sget-object v1, Lcom/marcohc/toasteroid/Toasteroid$STYLES;->ERROR:Lcom/marcohc/toasteroid/Toasteroid$STYLES;

    invoke-static {p0, v0, v1, v2}, Lcom/marcohc/toasteroid/Toasteroid;->show(Landroid/app/Activity;Ljava/lang/String;Lcom/marcohc/toasteroid/Toasteroid$STYLES;I)V

    :goto_12
    return-void

    :cond_13
    const-string v0, "Application running on Real device"

    sget-object v1, Lcom/marcohc/toasteroid/Toasteroid$STYLES;->SUCCESS:Lcom/marcohc/toasteroid/Toasteroid$STYLES;

    invoke-static {p0, v0, v1, v2}, Lcom/marcohc/toasteroid/Toasteroid;->show(Landroid/app/Activity;Ljava/lang/String;Lcom/marcohc/toasteroid/Toasteroid$STYLES;I)V

    goto :goto_12
.end method
```

- Note that variable name, condition name, etc... is different based on decompile tool.

Detection -> Anti-emu

- If you reverse the app, you don't really need to understand all the work, you just find what you need, and hack it ☺
- Some points:

```
    if-ne v0, v2, :cond_13

    const-string v0, "Application running on Emulator"
```

```
    :cond_13
    const-string v0, "Application running on Real device"
```

- if-ne vx, vy, target == "Jumps to target if vx!=vy. vx and vy are integer values."
- So the code above can be explained:

If v0 != v2, then go to :cond_13 (:cond_13 means the app is running on real device, not emulator), else continue to "Emulator detect" part

Detection -> Anti-emu

- So we need to find out what v0, v2 is, and do blah, blah, blah, super power, zeronight, 0day xyz to bypass it?.... Um no, there is a easier way
- Look at this:

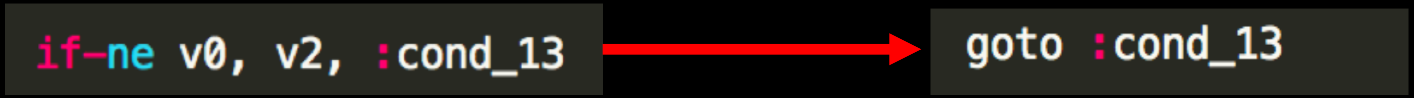
```
if-ne v0, v2, :cond_13
```

If ... not equal ... jump to cond_13 (right part)

- So what happens if we remove the "not equal" and force it always go to :cond_13?

=> The flow will always jump to :cond_13 ☺

- Let try it:

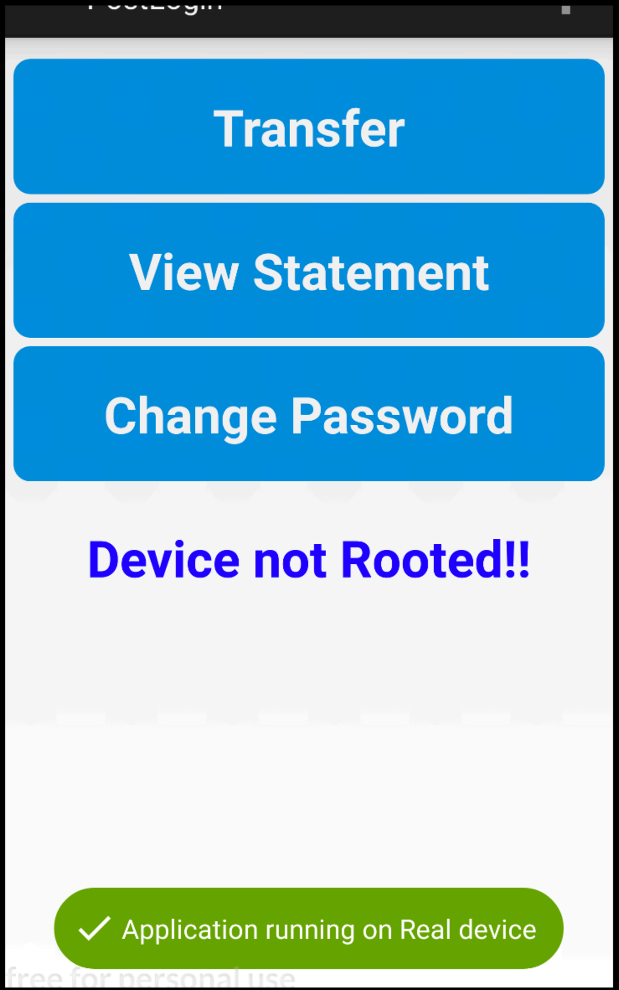


```
if-ne v0, v2, :cond_13
```

```
goto :cond_13
```

Detection -> Anti-emu

- With this idea, let change it in InsecureBankv2/smali/com/android/insecurebankv2/PostLogin.smali
- Then re-compile to get a new patched apk, sign it.
- Open & see the result ☺:



Transfer

View Statement

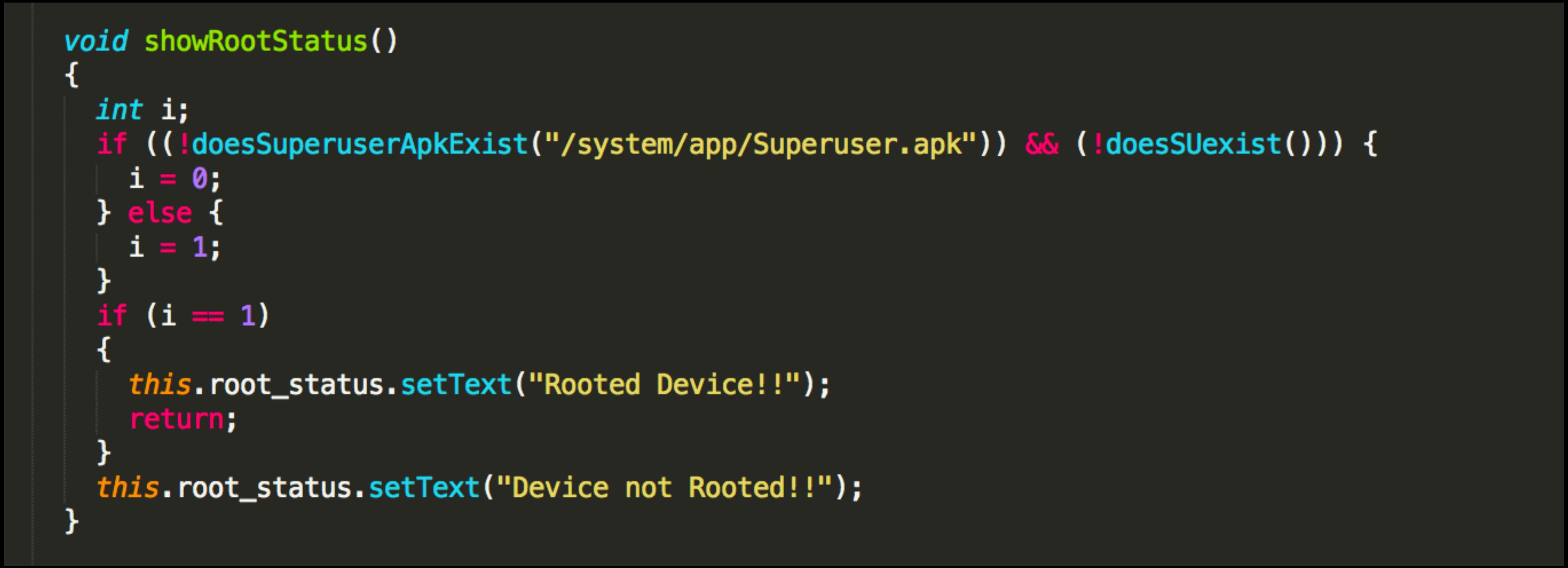
Change Password

Device not Rooted!!

✓ Application running on Real device

Detection -> Anti-root

- So why the anti-root code failed to detect us? We can see the code is here:



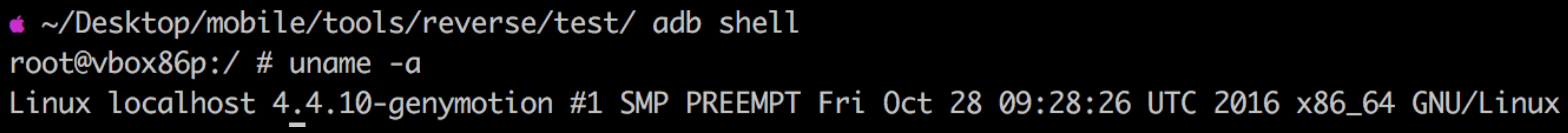
```
void showRootStatus()
{
  int i;
  if ((!doesSuperuserApkExist("/system/app/Superuser.apk")) && (!doesSUexist())) {
    i = 0;
  } else {
    i = 1;
  }
  if (i == 1)
  {
    this.root_status.setText("Rooted Device!!");
    return;
  }
  this.root_status.setText("Device not Rooted!!");
}
```

- It check if /system/app/Superuser.apk existed?

Yes => detected, else no.

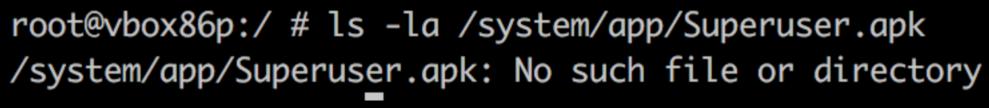
Detection -> Anti-root

- To find out what is happened, we access to our device



```
~/Desktop/mobile/tools/reverse/test/ adb shell
root@vbox86p:/ # uname -a
Linux localhost 4.4.10-genymotion #1 SMP PREEMPT Fri Oct 28 09:28:26 UTC 2016 x86_64 GNU/Linux
```

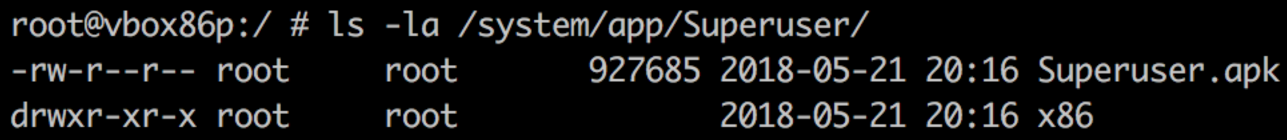
- Then find it



```
root@vbox86p:/ # ls -la /system/app/Superuser.apk
/system/app/Superuser.apk: No such file or directory
```

That’s why we pass the check by default

- Basically, its located here ☺:



```
root@vbox86p:/ # ls -la /system/app/Superuser/
-rw-r--r-- root     root       927685 2018-05-21 20:16 Superuser.apk
drwxr-xr-x root     root              2018-05-21 20:16 x86
```