Root Detection Bypass

Root detection in Android applications is a common security measure used by developers to help preserve the integrity of their apps. It's purpose is to detect whether a device has been rooted—a state in which users gain privileged access to the operating system. While rooting allows for deeper customization, it also introduces significant security risks. Rooted devices are more vulnerable to malicious attacks, as root access can bypass many of Android's built-in security protections. This poses a particular threat to applications that handle sensitive information, such as banking apps.

There are many methods developers can use to implement root detection checks in an application. Simple checks might include searching for common rooting apps, looking for the presence of a `su` binary, checking for common open-source libraries, and others. When an application detects a rooted device, it might restrict access, limit certain features, or display a warning message to the user. However, malicious users—and sometimes even regular users—may attempt to bypass these checks using evasion techniques. There are several reasons someone might want to circumvent root detection: root access allows users to modify the device beyond manufacturer limitations, such as uninstalling system apps or running apps that require elevated privileges. If an application blocks usage on rooted devices, a user may wish to bypass this restriction in order to retain both functionality and the benefits of rooting. In this section, we'll explore how to bypass root detection by patching the application.

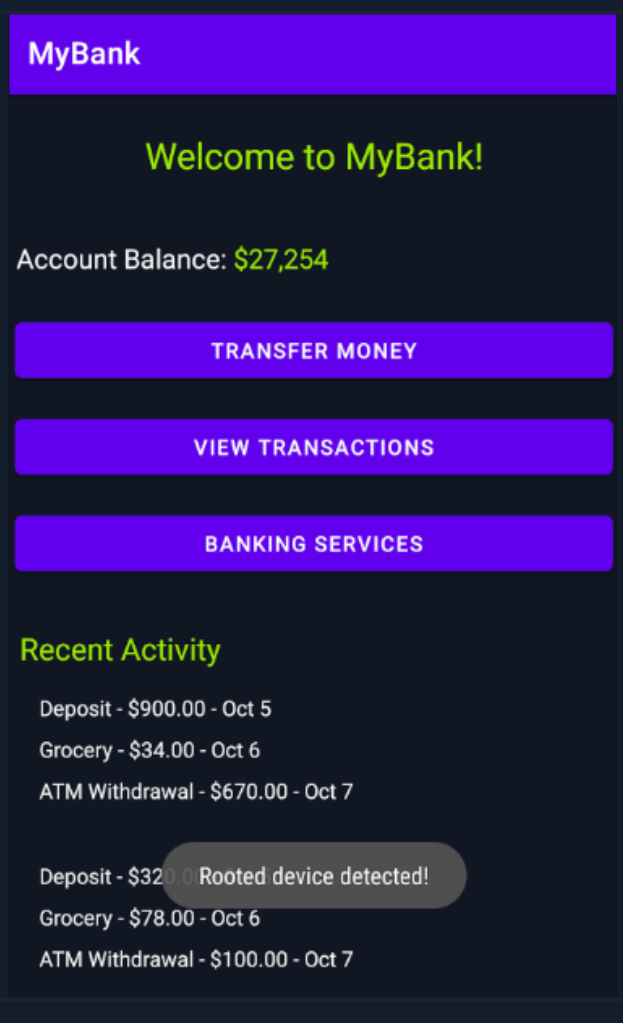
For this hands-on exercise, we suggest you use an AVD emulator, although another emulator or physical device will work as well. Once your emulator is running, connect to the device via `ADB` and install the application:

Root Detection Bypass

```
rl1k@htb[/htb]$ adb connect
rl1k@htb[/htb]$ adb install myapp.apk

Performing Streamed Install
Success
```

The following scenario features a bank application with root detection, used to prevent the app from running on rooted devices.

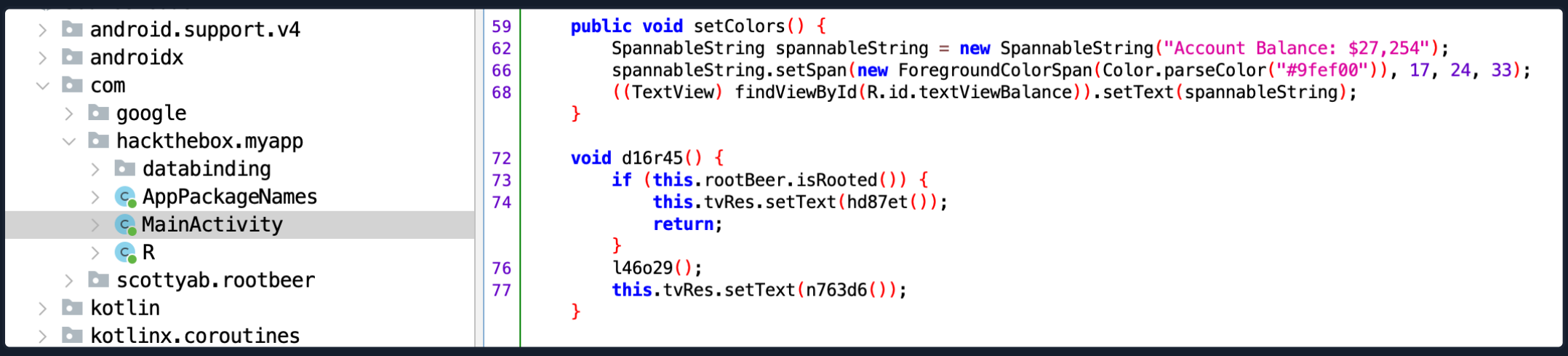


Third-Party Libraries

As we can see in the above image, the bank application detected a rooted device, and the message `Rooted device detected` is displayed. Let's open the APK file using JADX and read its source code.

Root Detection Bypass

```
rl1k@htb[/htb]$ jadx-gui myapp.apk
```



The above snippet reveals the method `d16r45()`, which includes the line `if (this.rootBeer.isRooted())`. If the `isRooted()` method is true, the string return from the method `hd87et()` will be printed. The imported package `com.scottyab.rootbeer.RootBeer` tells us the app is using a library called rootbeer to check if the device is rooted. Using APKTool, we can edit the Smali code and bypass the root detection check by reversing the `if` condition, as demonstrated in the previous section.

First, we'll disassemble the APK file and open its smali source code with a text editor.

Root Detection Bypass

```
rl1k@htb[/htb]$ apktool d myapp.apk
rl1k@htb[/htb]$ vim myapp/smali/com/hackthebox/myapp/MainActivity.smali
```

```
85      invoke-virtual {v0}, Lcom/scottyab/rootbeer/RootBeer;->isRooted()Z
86
87      move-result v0
88
89      if-eqz v0, :cond_0
90
91      .line 74
92      iget-object v0, p0, Lcom/hackthebox/myapp/MainActivity;->tvRes:Landroid/widget/TextView;
93
94      invoke-direct {p0}, Lcom/hackthebox/myapp/MainActivity;->hd87et()Ljava/lang/String;
95
96      move-result-object v1
```

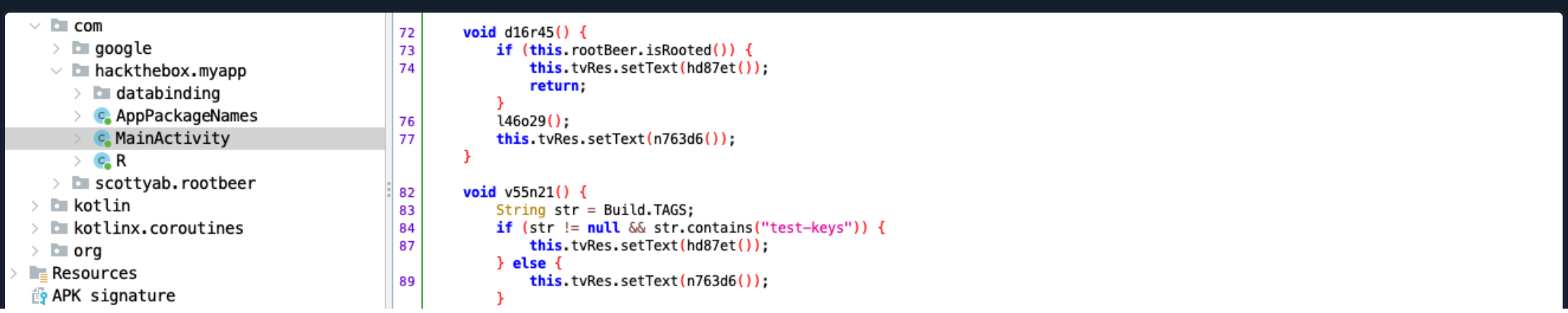
Changing the line `if-eqz v0, :cons_0` to `if-nez v0, :cons_0` will reverse the condition and bypass the root detection check. Before recompiling and signing the app, let's review the source code for any additional root detection mechanisms.

Build Tags

In this example, root detection is implemented by checking the build tags of the device's operating system. Build tags are part of the system's build properties and indicate how the OS was compiled. On non-rooted devices, the build tag is typically set to something like `release-keys`, whereas rooted devices (or those running custom ROMs) often use tags like `test-keys` or `dev-keys`. Let's open the APK with JADX and see how this check is implemented.

Root Detection Bypass

```
rl1k@htb[/htb]$ jadx-gui myapp.apk
```



The code snipped reveals another method called `v55n21()`. The line `String str = Build.TAGS;` stores the string returned from the field `Build.TAGS` in the `str` variable. If the substring `test-keys` is found within this string, the app considers the device as rooted. To bypass this check, we can modify the Smali code using APKTool by changing the condition if `(str != null && str.contains("test-keys"))`.

Root Detection Bypass

```
rl1k@htb[/htb]$ vim myapp/smali/com/hackthebox/myapp/MainActivity.smali
```

```
322     const-string v1, "test-keys"
323
324     .line 84
325     invoke-virtual {v0, v1}, Ljava/lang/String;->contains(Ljava/lang/CharSequence;)Z
326
327     move-result v0
328
329     if-eqz v0, :cond_0
330
331     const/4 v0, 0x1
332
333     goto :goto_0
334
```

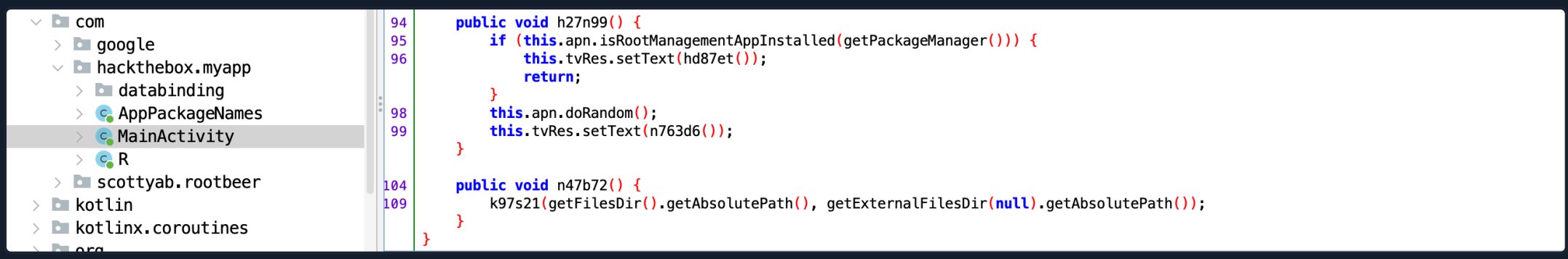
Changing the line `if-eqz v0, :cons_0` to `if-nez v0, :cons_0` will reverse the condition and the root detection check. Again, let's hold off on recompiling and signing the app for now, and move on to another example.

Root Management Apps

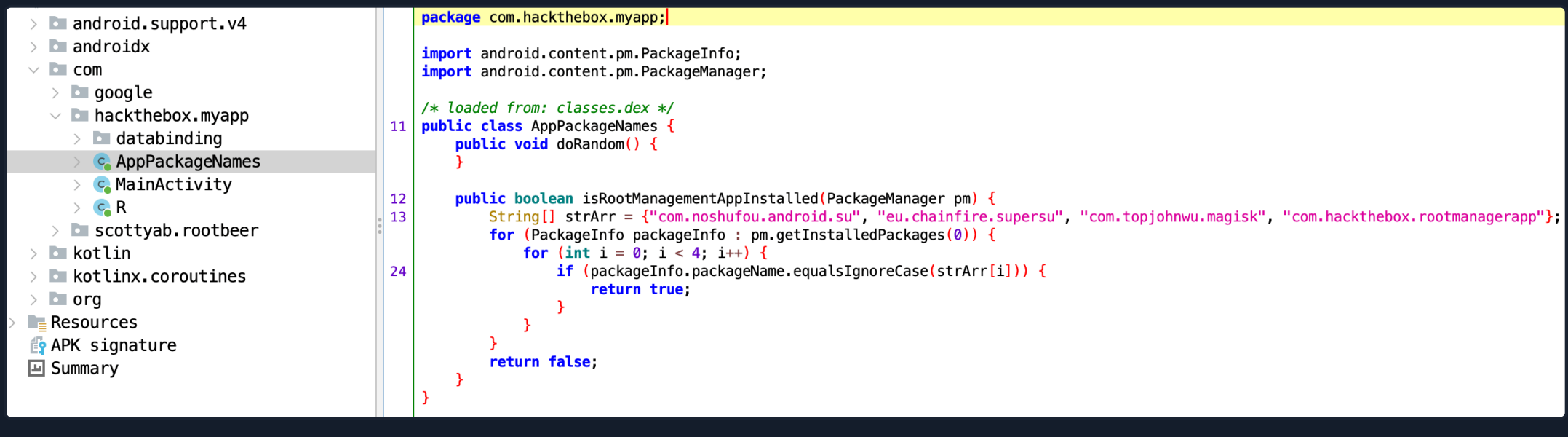
In this scenario, root management applications are installed on the device. These allow users to manage root permissions on rooted Android systems, controlling which apps are granted superuser access.

Root Detection Bypass

```
rl1k@htb[/htb]$ jadx-gui myapp.apk
```



The method `h27n99()` checks whether the condition `if (this.apn.isRootManagementAppInstalled(getPackageManager()))` returns true. To understand how this check is performed, double-click on `isRootManagementAppInstalled()` to view its contents.



This reveals the class `AppPackageNames` and the method `isRootManagementAppInstalled()`. The method lists the package names of installed applications by way of the `getInstalledPackages()` method, which checks the `strArr` array for the existence of `"com.noshufou.android.su"`, `"eu.chainfire.supersu"`, `"com.topjohnwu.magisk"`, and `"com.hackthebox.rootmanagerapp"`. These are the package names of several root management applications. Subsequently, we can use APKTool to remove them from the array and see if the bypass is successful.

Let's decompile the APK and edit the Smali code accordingly.

Root Detection Bypass

```
rl1k@htb[/htb]$ apktool d myapp.apk
rl1k@htb[/htb]$ grep -Rnw './myapp/smali' -e 'com.hackthebox.rootmanagerapp'
rl1k@htb[/htb]$ sed -i -e 's/com.noshufou.android.su//g' -e 's/eu.chainfire.supersu//g' -e 's/com.topjohnwu.magisk//g' -e 's/
```

Listing the contents of the file `./myapp/smali/com/hackthebox/myapp/AppPackageNames.smali` shows that the package names have been successfully removed.

Root Detection Bypass

```
rl1k@htb[/htb]$ cat ./myapp/smali/com/hackthebox/myapp/AppPackageNames.smali

<SNIP>
  const-string v0, ""

    const-string v1, ""

    const-string v2, ""

    const-string v3, ""

    .line 13
    filled-new-array {v2, v3, v0, v1}, [Ljava/lang/String;

    move-result-object v0
    const/4 v1, 0x0
<SNIP>
```

At this point, we are ready to recompile, sign the app, and test.

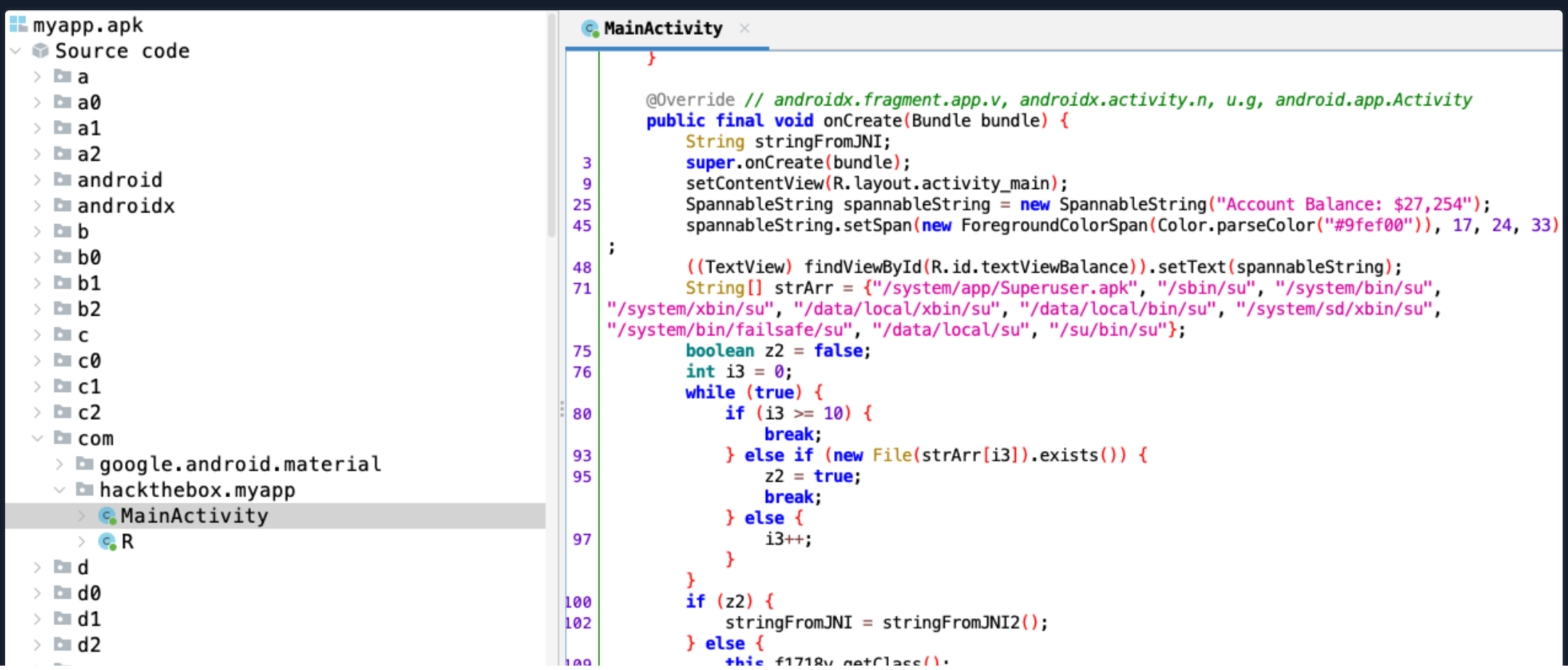
SU Binary

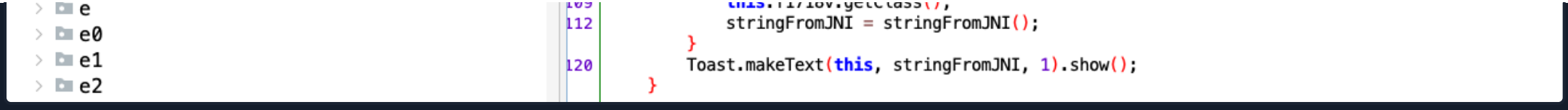
In Java Code

In this example, the source code is obfuscated, and reading the method names or the hardcoded messages is not possible. Let's try to follow the application flow to see if we can find any root detection methods. Checking the source code of the app using JADX reveals the following snippet.

Root Detection Bypass

```
rl1k@htb[/htb]$ jadx-gui myapp.apk
```





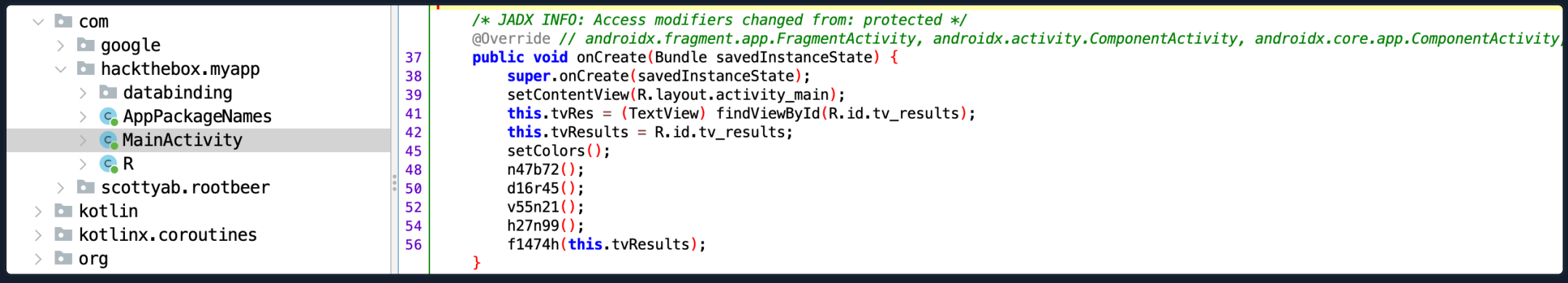
The application checks whether the `su` binary exists in specific directories—common locations where rooting tools typically install it. Detecting this file is a widely used method for determining if a device is rooted. As shown in the code snippet above, the methods `f1718v.getClass()` and `stringFromJNI()` are only called if the variable `z2` is false. For `z2` to be false, the `strArr` array must not contain any of the known paths where the su binary might reside. Since a rooted device could include this file in one of those locations, we can bypass the check by patching the application with APKTool. This involves either reversing the `if` condition or replacing the paths with empty strings, as demonstrated in previous examples.

Within Native Code

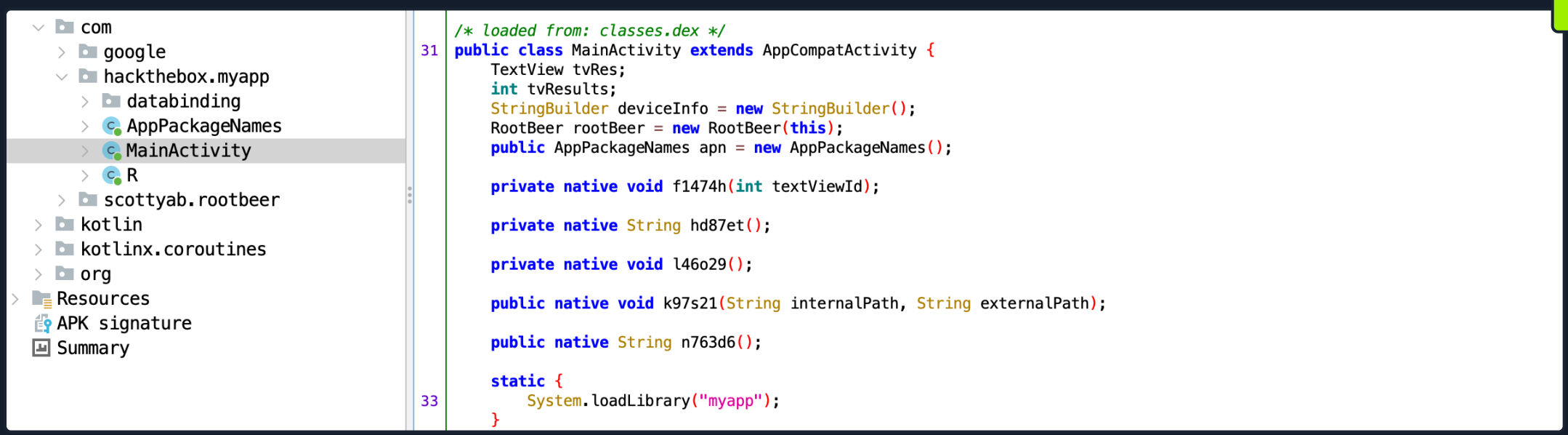
Implementing root detection in native code significantly increases the difficulty of bypassing it, as native libraries are harder to reverse engineer than Java code. The following example features an app that performs root checks within native code. We'll start by analyzing the source using JADX:

Root Detection Bypass

```
rl1k@htb[/htb]$ jadx-gui myapp.apk
```



Reading the content of the method `onCreate()` reveals the method `f1474h()`. Further analysis of the `MainActivity` confirms that it is a native method inside the library `myapp`.



Let's use APKTool to extract the library from the APK.

Root Detection Bypass

```
rl1k@htb[/htb]$ apktool d myapp.apk
rl1k@htb[/htb]$ ls -l myapp/lib/x86_64

total 576
-rw-r--r--  1 bertolis  bertolis  294320 Nov 17 16:31 libmyapp.so
```

Then, can open the extracted library using Ghidra.

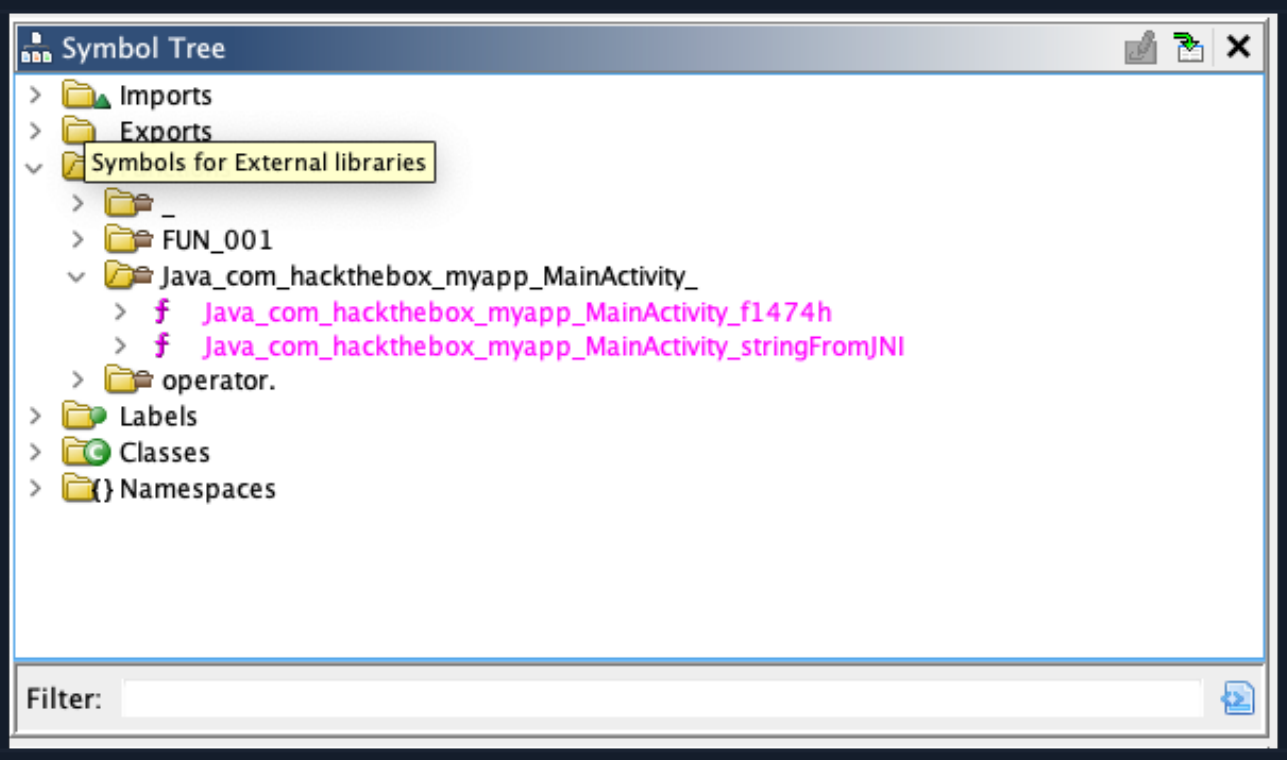
Root Detection Bypass

```
rl1k@htb[/htb]$ ghidra
```

When Gihdra starts, click on `File` -> `New Project` -> `Non-Shared Project` -> `Next`, enter a Project Name, and click `Finish`. Once the project is created,

click on `File` -> `Import File`, navigate to the `libmyapp.so` file, click `Select File To Import`, and click `OK` at eacg if the following pop-up windows.

Next, double-click on the imported file, click `Yes` and `Analyze` on the next pop-up window. An inspection of the functions included in this library (under `Symbol Tree` at the left of the window) reveals the function `Java_com_hackthebox_myapp_MainActivity_f1474h`.



Let's click on it and view the library's source code in the `Decompile` pane at the left of the window.

Decompile: Java_com_hackthebox_myapp_MainActivity_f1474h - (libmyapp.so)

```
1
2 void Java_com_hackthebox_myapp_MainActivity_f1474h(long *param_1,_jmethodID *param_2)
3
4 {
5   int iVar1;
6   long lVar2;
7   ulong uVar3;
8
9   iVar1 = access("/system/app/Superuser.apk",0);
10  uVar3 = 1;
11  if (iVar1 == -1) {
12    iVar1 = access("/sbin/su",0);
13    if (iVar1 == -1) {
14      iVar1 = access("/system/bin/su",0);
15      if (iVar1 == -1) {
16        iVar1 = access("/system/xbin/su",0);
17        if (iVar1 == -1) {
18          iVar1 = access("/data/local/xbin/su",0);
19          if (iVar1 == -1) {
20            iVar1 = access("/data/local/bin/su",0);
21            if (iVar1 == -1) {
22              iVar1 = access("/system/sd/xbin/su",0);
23              if (iVar1 == -1) {
24                iVar1 = access("/system/bin/failsafe/su",0);
25                if (iVar1 == -1) {
26                  iVar1 = access("/data/local/su",0);
27                  if (iVar1 == -1) {
28                    iVar1 = access("/su/bin/su",0);
29                    uVar3 = (ulong)(iVar1 != -1);
30                  }
31                }
32              }
33            }
34          }
35        }
36      }
37    }
38  }
39  lVar2 = (**(code **)(*param_1 + 0xf8))(param_1,param_2);
40  if (lVar2 != 0) {
41    lVar2 = (**(code **)(*param_1 + 0x108))(param_1,lVar2,"showToast",&DAT_00113485);
42    if (lVar2 != 0) {
43      _JNIEnv::CallVoidMethod((_jobject *)param_1,param_2,lVar2,uVar3);
44      return;
45    }
46  }
47  return;
48 }
```

This function performs several checks for rooted status, such as verifying the presence of files like `/system/bin/su`. It uses the same detection technique mentioned earlier but executes it within native code. Since this function solely handles root detection, we can bypass it by removing the call to `f1474h()` from the `onCreate()` method in the Smali code.

Let's open the file `myapp/smali/com/hackthebox/myapp/MainActivity.smali` using a text editor and search for the method `f1474h()` within the `onCreate()` method.

Root Detection Bypass

```
rl1k@htb[/htb]$ vim myapp/smali/com/hackthebox/myapp/MainActivity.smali
```

Code: smali

```
<SNIP>
   .line 52
    invoke-virtual {p0}, Lcom/hackthebox/myapp/MainActivity;->v55n21()V

    .line 54
    invoke-virtual {p0}, Lcom/hackthebox/myapp/MainActivity;->h27n99()V

    .line 56
    iget p1, p0, Lcom/hackthebox/myapp/MainActivity;->tvResults:I

    invoke-direct {p0, p1}, Lcom/hackthebox/myapp/MainActivity;->f1474h(I)V

    return-void
.end method
<SNIP>
```

Next, remove the following lines.

Code: smali

```
<SNIP>
   invoke-direct {p0, p1}, Lcom/hackthebox/myapp/MainActivity;->f1474h(I)V
<SNIP>
```

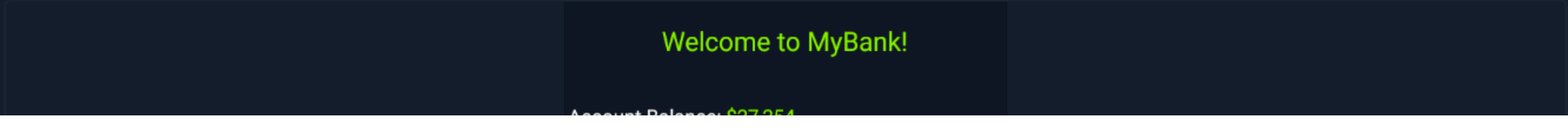
Then, we recompile, sign, and install the modified application.

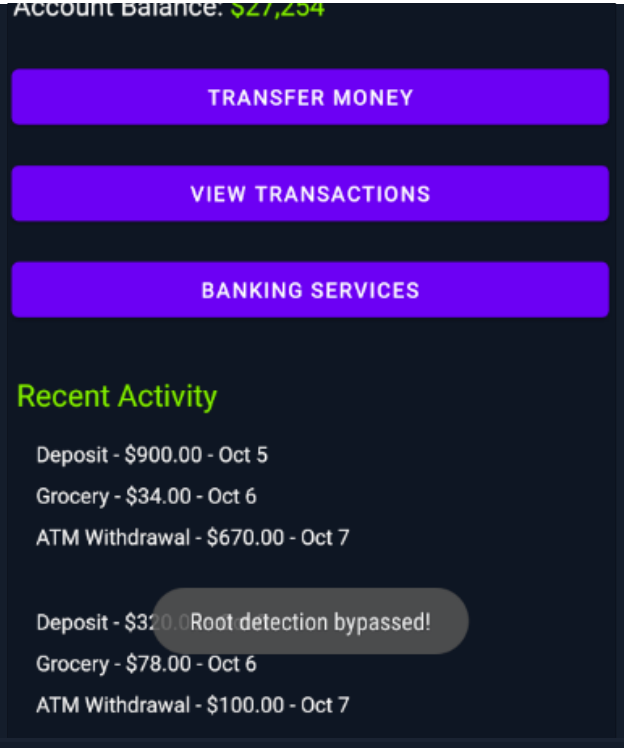
Root Detection Bypass

```
rl1k@htb[/htb]$ apktool b myapp
rl1k@htb[/htb]$ echo -e "password\npassword\njohn doe\ntest\ntest\ntest\ntest\ntest\nyes" > params.txt
rl1k@htb[/htb]$ cat params.txt | keytool -genkey -keystore key.keystore -validity 1000 -keyalg RSA -alias john
rl1k@htb[/htb]$ zipalign -p -f -v 4 myapp/dist/myapp.apk myapp_aligned.apk
rl1k@htb[/htb]$ echo password | apksigner sign --ks key.keystore myapp_aligned.apk
rl1k@htb[/htb]$ adb uninstall com.hackthebox.myapp
rl1k@htb[/htb]$ adb install myapp_aligned.apk

Performing Incremental Install
Serving...
All files should be loaded. Notifying the device.
Success
Install command complete in 381 ms
```

Finally, running the app on the device displays the message `Root detection bypassed`.





It is not recommended to delete a function or remove the entire library without first understanding its role. Other parts of the application may rely on functions within the library, and doing so could cause the app to crash. In many cases, root detection relies on a combination of multiple checks—both in Java and native code—to increase robustness.

Connect to Pwnbox

Your own web-based Parrot Linux instance to play our labs.

Pwnbox Location

UK 31ms

Terminate Pwnbox to switch location

Start Instance

∞ / 1 spawns left

Waiting to start...

Enable step-by-step solutions for all questions

Questions

Answer the question(s) below to complete this Section and earn cubes!

Cheat Sheet

+ 6 Download and install both APK files into the emulator. After successfully bypassing the root detection mechanism, what message is displayed on the screen? (Note: The "SU Binary In Java Code" step is excluded from this task.)

Submit your answer here...

+10 Streak pts | Submit | myapp_detection_mechanisms.zip

← Previous | Next →

Cheat Sheet

? Go to Questions

Table of Contents

Extracting and Enumerating APK Files

- Introduction
- Disassembling the APK

Understanding Smali

Analyzing Application's Source Code

- Reading Hardcoded Strings
- Bad Cryptography Implementation
- Reversing Hybrid Apps
- Reading Obfuscated Code
- Deobfuscating Code

Analyzing Native Libraries

- Reversing Shared Objects
- Reversing DLL Files

Application Patching

- Authentication Bypass
- Modifying Game Apps
- License Verification Bypass
- Root Detection Bypass

Skills Assessment

- Skills Assessment

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left