Android Apk reverse engineering using Apktool

and Frida

ON APRIL 22, 2019 / BY RUSSELL / IN ANDROID, HOOKING, REVERSE ENGINEERING

Android Apk reverse engineering using Apktool and Frida

Table of Content

Environment Setup (macOS mojave)

All the steps mentioned below are on macOS Mojave

Install Tools on PC

1. Install `frida-tools` with pip3

   Run command `$ pip --version` and `$ pip3 --version` to check which *pip* is from at **Python 3x**. E.g. you should see version information like below:

   ```
   $ pip 19.0.3 from /usr/local/lib/python3.7/site-packages/pip (python 3.7)
   ```

   Then run the right **pip** command to install `frida-tools`, e.g. `pip3`

   ```
   $ pip3 install frida-tools
   ```

   Success outputs:

   ```
    ....
   Successfully built frida-tools frida

   Installing collected packages: colorama, frida, six, wcwidth, prompt-toolkit, pygments, frida-tools
   ```

```
Successfully installed colorama-0.4.1 frida-12.4.7 frida-tools-1.3.2 prompt-toolkit-2.0.9 pygments-2.3.1
six-1.12.0 wcwidth-0.1.7
```

2. Testing your installation

   Copy the `cat` binary to a temporary folder, e.g., `/tmp/cat` then run `cat` from that directory:

   ```
    $ mkdir ~/frida
   $ cp /bin/cat /frida/cat
   $ /frida/cat
   ```

   In another terminal, make a file `example.py` with the following contents:

   ```
    import frida

   def on_message(message, data):
     print("[on_message] message:", message, "data:", data)

   session = frida.attach("cat")

   script = session.create_script("""'use strict';
   ```

```
rpc.exports.enumerateModules = function () {
return Process.enumerateModulesSync();
};
""")
script.on("message", on_message)
script.load()

print([m["name"] for m in script.exports.enumerate_modules()])
```

Then run the `example.py` script with below command

```
$ python3 example.py
```

The output should be something similar to this (depending on your platform and library versions):

```
[u'cat', …, u'ld-2.15.so']
```

3. Android `adb` command

Make sure `adb` can see your device:

```
$ adb devices -l
```

This will also ensure that the adb daemon is running on your desktop, which allows Frida to discover and communicate with your device regardless of whether you’ve got it hooked up through USB or WiFi.

Install `frida-server` on Emulator (or Real Device)

Below steps are based on **Android Emulator Nexus 6P (x86)** api 23 on macOS Majave.

First off, download the latest `frida-server` for Android from frida-server releases page, e.g. for Android `x86` emulator, you should download frida-server-12.4.7-android-x86.xz, and get it run on your emulator:

```
 $ adb root # might be required
$ adb push frida-server /data/local/tmp/
$ adb shell "chmod 755 /data/local/tmp/frida-server"
$ adb shell "/data/local/tmp/frida-server &"
```

For the last step, if you see below error:

```
Unable to load SELinux policy from the kernel: Failed to open file “/sys/fs/selinux/policy”: Permission denied
```

Then you need to run `frida-server` using the root shell, e.g.

```
 $ adb shell
angler:/ $ su
angler:/ # /data/local/tmp/frida-server &
[1] 12089
angler:/ #
```

`[1] 12089` is the process id of `frida-server`.

A quick smoke-test

Now, on your desktop it’s time to make sure the basics are working. Run:

```
 frida-ps -U
```

This should give you a process list along the lines of:

```
  PID  Name
-----  ---------------------------------------------------
```

```
 721 ATFWD-daemon
4450 adbd
 730 android.hardware.biometrics.fingerprint@2.1-service
 407 android.hardware.configstore@1.0-service
 408 android.hardware.graphics.allocator@2.0-service
 409 android.hardware.usb@1.0-service
 410 android.hardware.wifi@1.0-service
 406 android.hidl.allocator@1.0-service
```

Tamper Smali Code

To tamper a `Boolean` value inside Java source code, i.e. the `Boolean bTamperingSucces = false;`, or some other code you have interest in.

Tools required

apktool

`apktool` can be feteched from Apktool website. Just follow the steps inside this page to install apktool.

adb

`adb` is shipped with Android SDK, it can be found from directory `<your-some-path>/Android/sdk/platform-tools/adb`

apksigner

`apksigner` is to sign your apk with a keystore file. This tool can be found at directory `<ANDROID_HOME>/Android/sdk/build-tools/28.0.3/apksigner`, and the usage is documented at command-line apksigner.

Steps

1. Clone the example project from DecompileApk.
2. Find the already compiled apk file `DecompileApk/app/release/app-release.apk`.
3. Decompile it using **apktool**.

```
 $ cd <your-path>/DecompileApk/app/release/
$ apktool d --no-res -f app-release.apk
```

You will see below outputs

```
I: Using Apktool 2.4.0 on app-release.apk
I: Copying raw resources...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

4. Look for `DecompileApk/app/release/app-release/smali/com/arophix/decompileapk/MainActivity.smali` under the smali code directory and find below code

```
const/4 p1, 0x0
```

5. Just change `0x0` (meaning `false`) to `0x1` (meaning `true`) and save the file.
6. Using **apktool** to build the tampered apk.

```
apktool b app-release
```

You should see below outputs

```
I: Using Apktool 2.4.0
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
I: Copying raw resources...
```

```
I: Copying libs... (/lib)

I: Building apk file...

I: Copying unknown files/dir...

I: Built apk...
```

7. Find the newly built apk from `dist` directory `DecompileApk/app/release/app-release/dist/app-release.apk`
8. Sign the apk using **apksigner** and keystore located at `DecompileApk/app/decompileapk.jks` (please modify the paths for keystore and apk per your own case accordingly),

   ```
   $ <ANDROID_HOME>/sdk/build-tools/28.0.3/apksigner sign --ks ../decompileapk.jks app-release.apk
   ```

   You should see below outputs and enter the password `123456`

   ```
   Keystore password for signer #1:
   ```
9. Install the signed apk using adb command.

   ```
   $ adb install <path-to-the-tampered-apk>/app-release.apk
   ```
10. Instead of seeing `"Hello from C++"` from the screen, you should now see `"Hello, Android reverse engineer!"` .

Hooking Android Java Methods

Hooking Android Java source code using Frida.

Find the script from hook_java_methods.py and run it using below command, and then click on the button of your started Android app.

```
 $ python3 hook_java_methods.py


[*] Running CTF
[*] onClick
[*] Called - isPhoneRooted()
[*] onClick
[*] Called - isPhoneRooted()
```

If you see an error like below:

```
 $ frida.ServerNotRunningError: unable to connect to remote frida-server: closed
```

Remember that you have started `frida-server` on your emulator.

Meanwhile, on your emulator screen, you shoud see the toast message changed to `Device not rooted` if success.

Hooking Android C Functions

Hooking Android C source code using Frida.

1. Decompile APK

   Firstly, decompile the apk using apktool to extract the shared library, i.e. `libnative-lib.so` .

   ```
    $ cd DecompileApk/app/release
   $ apktool d --no-res app-release.apk
   ```

2. Find the target JNI method

   Secondly, use below command to find the JNI function to hook.

   ```
    $ nm --demangle --dynamic app-release/lib/x86/libnative-lib.so
   ```

   You should see below outputs:

   ```
    00004d80 T Java_com_arophix_decompileapk_MainActivity_stringFromJNI
   000090b0 T std::bad_typeid::what() const
   00005cf0 T std::bad_exception::what() const
   ```

```
00005e70 T std::bad_array_length::what() const
00005df0 T std::bad_array_new_length::what() const
00008ff0 T std::bad_cast::what() const
...
```

3. Hook C function by name

   Find the script from hooknative-by-function-name.py and run it using below command, and then click on the button of your started Android app.

```
 $ python3 hooknative-by-function-name.py


[*] Running Arophix Hook Test ...
Java_com_arophix_decompileapk_MainActivity_stringFromJNI called with:
ret: 0x200019
```

   If you see an error like below:

```
 $ frida.ServerNotRunningError: unable to connect to remote frida-server: closed
```

   Remember that you have started `frida-server` on your emulator.

   Meanwhile, on your emulator screen, you shoud see the text changed to `Frida is hooking this displayed text from Native layer by function name.` if success.

4. Hook C function by address

   Find the script from hooknative-by-function-address.py and run it using below command, and then click on the button of your started Android app.

   ```
   $ python3 hooknative-by-function-address.py

   [*] Running Arophix Hook Test ...
   [+] membase: 0xb2acd000
   [+] addressOfStringFromJni: 0xb2ad1d80
   [++] addressOfStringFromJni: 0xb2ad1d80
   ret: 0x19
   ```

   If you see an error like below:

   ```
   $ frida.ServerNotRunningError: unable to connect to remote frida-server: closed
   ```

   Remember that you have started `frida-server` on your emulator.

   Meanwhile, on your emulator screen, you shoud see the text changed to `Frida is hooking this displayed text from Native layer by function address.` if success.

References

- https://www.frida.re/docs/android/
- https://www.frida.re/docs/examples/android/
- https://11×256.github.io/Frida-hooking-android-part-1/
- https://github.com/antojoseph/frida-android-hooks/blob/master/hooking-native-code.py
- https://awakened1712.github.io/hacking/hacking-frida/
- https://awakened1712.github.io/hacking/hacking-frida/#c-hook-a-static-function-by-resolving-its-address
- https://stackoverflow.com/questions/51811348/find-manually-registered-obfuscated-native-function-address

Share this:

Twitter Facebook

★ Like

One blogger likes this.

Related

Tamper an Android native shared library (.so) using IDA Pro 7.0
In "Android"

Android Substrate Hooking C Code
In "Android"

A step by step tutorial to test Android backup and restore
In "Android"

ANDROID APKTOOL FRIDA HOOKING SMALI

PREVIOUS

Tamper an Android native shared library (.so) using IDA Pro 7.0

NEXT

A step by step tutorial to test Android backup and restore

Leave a Reply

Enter your comment here...

Follow Arophix 2

FOLLOW BLOG VIA EMAIL

Enter your email address to follow this blog and receive notifications of new posts by email.

Join 4 other followers

Enter your email address

FOLLOW

RECENT POSTS

A step by step tutorial to test Android backup and restore

SEPTEMBER 12, 2019

Android Apk reverse engineering using Apktool and Frida

APRIL 22, 2019

Tamper an Android native shared library (.so) using IDA Pro 7.0

OCTOBER 31, 2018

Android Xposed Hook Example

OCTOBER 16, 2018

sed command usage

OCTOBER 5, 2018

ANDROID JNI

- Android JNI – Overview
- Android JNI – Object Operations
- Android JNI – Class Operations, Exceptions and References
- Android JNI – String Operations
- Android JNI – Array Operations
- Android JNI – NIO and Reflection Support
- Andoid JNI – Summary

HOOKING

- Android Substrate Hooking C Code
- Android Xposed Hook Example

- Tamper an Android native shared library (.so) using IDA Pro 7.0

ARCHIVES

- September 2019 (1)
- April 2019 (1)
- October 2018 (3)
- July 2018 (2)
- April 2018 (3)
- March 2018 (2)
- February 2018 (1)
- January 2018 (1)
- December 2017 (10)
- November 2017 (2)

CATEGORIES

- Android (19)
  - Hooking (4)
  - Native (8)
    - JNI (8)
- Git (2)
- Reverse Engineering (2)
- shell script (1)
- Software Engineering (2)
  - Continuous Integration (2)
- Vim (4)

BLOGS I FOLLOW

- Applications – Android Police – Android news, reviews, apps, games, phones, tablets
- Android « Gadget Hacks
- Android Central - Android Forums, News, Reviews, Help and Android Wallpapers
- Android
- CSDN → 版主推荐-技术区 → RSS
- CSDN博客推荐文章
- Arophix
- Discover
- Longreads
- The Daily Post
- The WordPress.com Blog

RECENT COMMENTS

e on Setup Jenkins with Docker on…

e on Setup Jenkins with Docker on…

Gradle sync issue on… on Android Gradle Plugin 3.0+ syn…

Android Gradle Plugi… on Gradle proxy configuration

Android Gradle Plugi… on Gradle sync issue on Debian OS…

Gradle proxy configu… on Android Gradle Plugin 3.0+ syn…

Gradle proxy configu… on Gradle sync issue on Debian OS…

Gradle sync issue on… on Gradle proxy configuration

Russell on Android Substrate Hooking C…

Russell on Android Substrate Hooking C…

April 2019

M	T	W	T	F	S	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28

M	T	W	T	F	S	S
29	30					

« OCT

SEP »