Mobile Phone Security

Dr. Digvijaysinh Rathod
Associate Professor & Associate Dean
School of Cyber Security and Digital Forensics
National Forensic Sciences University with status of Institution of National Importance

digvijay.rathod@gfsu.edu.in

Frida

Frida tools,

Basic commands

https://frida.re/docs

https://github.com/rortega/Frukah

Frida Tools

- Frida is particularly useful for dynamic analysis on Android/iOS/Windows applications.
- It allows us to set up hooks on the target functions so that we can inspect/modify the parameters and return value.
- We can also alter the entire logic of the hooked function.
- This article shows the most useful code snippets for copy & paste to save time reading the lengthy documentation page.

https://frida.re/docs/frida-trace/

Frida Tools

- Frida Cli : REPL interface, a tool aimed at rapid prototyping and easy debugging, for more Use Frida –h

```
# Connect Frida to an Android over USB and list running processes
$ frida-ps -U

# List running applications
$ frida-ps -Ua

# List installed applications
$ frida-ps -Uai

# Connect Frida to the specific device
$ frida-ps -D 0216027d1d6d3a05
```

Frida Tools

- frida-trace : frida-trace is a tool for dynamically Monitoring/tracing Method calls. It is useful for debugging method calls on every event in mobile application.

```
Syntaxt:

frida -j "class!method" -U <<application_package_name>>

Demo Usages :
frida -j "*com.example.demotest*!*" -U com.example.demotest

*com.example.demotest* : Matching fQCN(fully qualified class name) that match  com.exmple.demotest and all
methods in com.example.demotest Android Application.
```

https://frida.re/docs/frida-trace/

Frida Tools

- frida-ps : This is a command-line tool for listing processes, which is very useful when interacting with a remote system.
- frida-discover: frida-discover is a tool for discovering internal functions in a program, which can then be traced by using frida-trace.
- frida-ls-devices: This is a command-line tool for listing attached devices, which is very useful when interacting with multiple devices.
- frida-kill: This is a command-line tool for killing processes.

https://frida.re/docs/frida-trace/

```
import frida, sys
 ss = """
Java.perform(function () {
    // declare classes that are going to be used
    const System = Java.use('java.lang.System');
    const Log = Java.use("android.util.Log");
    const Exception = Java.use("java.lang.Exception");
     System.exit.implementation = function() {
        //
console.log(Log.getStackTraceString(Exception.$new()));
    };
});
```

Hooking different methods in java

- Now, a class might have multiple methods and each of these methods have a specific purpose.
- For example, the onCreate() method defines the implementation of activity as soon as the activity is created (or launched).
- So, what is, we can hook this function and change the behaviour of the activity when it is created.
- For the demonstration purpose, I'll just print some custom text in my console as soon as the activity is called but the possibilities are limitless.
- Typically you won't have access to the source code, hence, what we'll do is extract the apk first and then decompile it to view source code.
- To pull the apk we'll first know it's the path and then pull it.

Hooking different methods in java

- adb shell pm path jakhar.aseem.diva
- adb pull /data/app/jakhar.aseem.diva-dxAm4hRxYY4VgIq2X5zU6w==/base.apk
- we'll decompile it using apktool and then use dex2jar to convert it in jar format, and finally use jd-gui to view the decompiled source code like below.
- Here is the MainActivity class decompiled.

Hooking different methods in java

MainActivity.class

```
package jakhar.aseem.diva;

import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;

public class MainActivity extends AppCompatActivity {
  protected void onCreate(Bundle paramBundle) {
    super.onCreate(paramBundle);
    setContentView(2130968616);
    setSupportActionBar((Toolbar)findViewById(2131493015));
  }
```

Hooking different methods in java

- Here we see the following things:
- We can see that onCreate has a Bundle parameter
- It’s creating a view of the main page
- Now, below is an example of how to hook onCreate() method.

Hooking different methods in java

```
console.log("Script loaded!");

Java.perform(function(){

var mainapp = Java.use("jakhar.aseem.diva.MainActivity");

mainapp.onCreate.implementation = function(){

        console.log("My script called!");

    var ret =

    this.onCreate.overload("android.os.Bundle").call(this);

    };

       send("Hooks installed");

);
```

Explanation:

1. Any implementation of the hook is put inside perform(function(){ //<code>

2. The activity we want to hook (main activity) is put inside use("jakhar.aseem.diva.MainActivity"), and assign a variable to it. Here, mainapp

3. Now, onCreate.implementation sets a definition of the function.

4. Here, we can insert any code we cant to run in the onCreate method. I just inserted log function to output “My script called!” every time onCreate is called.

5. New variable ret calls this newly formed implementation function. overload method is used to add this code to the existing piece of code. Here, “os.Bundle” is input as a parameter since in the original function a bundle object is used.

6. Finally, the call method is used to call the current method using “this” pointer.

7. send() function outputs the text in double-quotes on the current frida command line.

To launch this script we type in the following command:

```
frida -U -l mainactivityhook.js -f jakhar.aseem.diva
```

As you can see now, the hook is successfully installed, activity launches and our custom output is now displayed and the hook is successfully installed

Hooking a defined method

- Unlike the onCreate method that is present in the native libraries, some methods are custom created.
- For example, if you inspect the code of diva, you'll see a function startChallenge() that is launching challenges in the application.
- I'm not putting the code in here but you can refer to the decompiled code in the above step.
- Now, we'll observe that startChallenge is launching activities present in the project.

Hooking a defined method

- And since it is launching an activity, it has an "android.view.VIEW" argument passed in its code.
- Now in the code below, every time a user hits a button to start any challenge, we'll just force him to call our hook and our defined output would be displayed (that is MainActivity.startChallenge() is now started).
- Needless to say, we can change this by any implementation we want.

Hooking a defined method

```
console.log("Hooked startChallenge() function");
Java.perform(function(){
var newstart = Java.use("jakhar.aseem.diva.MainActivity");

newstart.startChallenge.overload("android.view.View").impleme
ntation = function(v){
        //enter any implementation of startChallenge you want
        //for demo I'm just sending an alert on frida console
         send("MainActivity.startChallenge() is now started");
         var ret =
this.startChallenge.overload("android.view.View").call(this);
          };
});
```

Hooking a defined method

- To call this script, without having to input %resume this time, we can type in the command with –no-pause filter:

  frida -U -l main_startchallenge.js -f jakhar.aseem.diva

- And sure enough, every time a button is pressed, our custom input is displayed.

Hooking exit() method:

- We can also tamper the exit method in android just like we tampered onCreate method.
- Here, I’m using a demonstration application that I custom coded.
- It has a button that is performing an exit function. You can see a sample screenshot below:

Hooking exit() method:

- Now, here we see the exit button. As the name states, on pressing it, application exits.
- We create a hook down below that will stop the exit. Here, “java.lang.System” is the package that has exit function and so we’ll overload it using “sysexit.exit.overload().implementation.”
- Now, whenever a user clicks on exit, our send method will be called and exit will be stopped.

Hooking exit() method:

```
console.log("Hooking on exit function");

Java.perform(function(){

var sysexit = Java.use("java.lang.System");

sysexit.exit.overload("int").implementation = function(var_0)

{

        send("I've stopped java.lang.System.exit() now!");

            };

});
```

Hooking exit() method:

- Let’s fire this script up and sure enough, we can see that the process is not terminated when the exit button is clicked.
- If it had been terminated frida must have thrown a process terminated error and closed the console.

frida -U -l avoidexit.js -f com.example.harshitrajpal --no-pause

Hooking return value

- We have hooked methods till now, but a return variable can also be hooked and its output be tampered with.
- In the application that I custom coded which is mentioned above, there is a simple program to display output of 10 and 50.
- We'll hook this return value and output 100. The code to do this is pretty straightforward:

Hooking return value

```
console.log("Hook for implementation of method");

Java.perform(function myFunc() {

        var myClass =

Java.use("com.example.harshitrajpal.MainActivity");

myClass.returnValue.implementation = function()
{
                //we will manipulate the return value here
                var ret = 100;
                return ret;
        }
});
```

Hooking return value

- Let’s first run the program without loading our hook. We can see that the program outputs 60 which is the correct answer.
- Now, we’ll fire up our script and see what changes happen in the application now.

frida -U -l retValueHook.js -f com.example.harshitrajpal --no-pause

Java.available:

This api is used to check Frida running on android or not. It is to check if you are actually running on Android. For example, you could create 1 SSL bypassing script that first checks if you're on Android or iOS, and act accordingly .It specify whether the current process has the a Java VM loaded, i.e. Dalvik or ART return boolean (true or false)

https://frida.re/docs/frida-trace/

Java.available:

This api is used to check Frida running on android or not. It is to check if you are actually running on Android. For example, you could create 1 SSL bypassing script that first checks if you're on Android or iOS, and act accordingly .It specify whether the current process has the a Java VM loaded, i.e. Dalvik or ART return boolean (true or false)

https://frida.re/docs/frida-trace/

Java.androidVersion: This Api return the android version of device that we are using.

Java.enumerateLoadedClasses(callback): This API enumerates classes where object specifying onMatch(name, handle): called for each loaded class with a name that may be passed to use() to get a JavaScript wrapper. onComplete(): called when all classes have been enumerated

https://frida.re/docs/frida-trace/

Mobile Phone Security

Dr. Digvijaysinh Rathod
Associate Professor & Associate Dean
School of Cyber Security and Digital Forensics
National Forensic Sciences University with status of Institution of National Importance

digvijay.rathod@gfsu.edu.in