iGNITE Technologies

ANDROID PENETRATION TESTING

FRIDA

WWW.HACKINGARTICLES.IN

Contents

Abstract

In this publication, we'll explain the basics of Frida, how to create your own Frida script, hook it into processes and perform various functions. Needless to say, there is no end to what a program can do, therefore, there is no limit on frida's applications, hence, this publication is only restricted to basics. If you want an advanced look into Frida and reverse engineering,

Frida

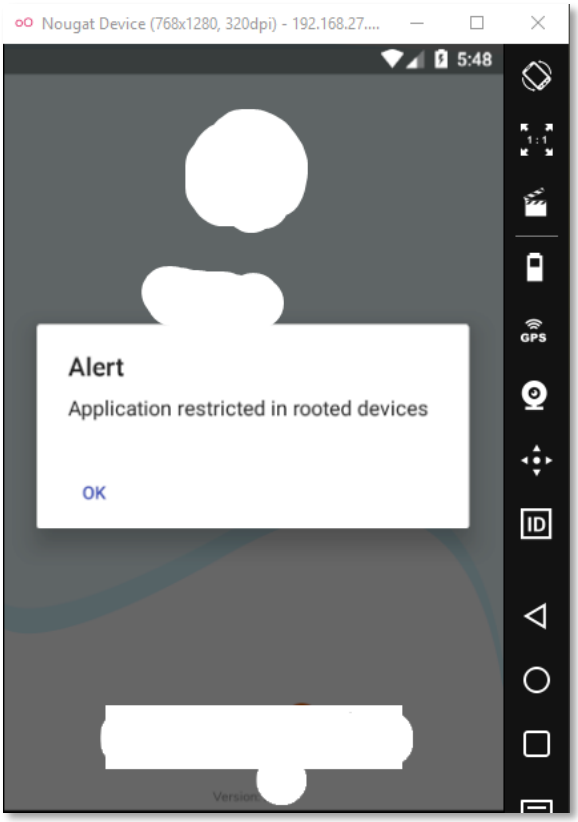
Introduction

Frida is a dynamic instrumentation toolkit that is used by researchers to perform android hooking (intercepting IPC and modifying it to make a function perform the desired function). Frida uses javascript to perform hooking since Android's native code and javascript both run on JIT compilation techniques, it can intercept its inter-process communication, add the code specified in a script and completely change the function's implementation. Some of its use cases in real life are:

- Spy on Crypto APIs
- Modify function's output
- Bypass AES encryption
- Bypass SSLPinning and Root detection
- Trace private application code
- Bypass various software sided locks (like applock)

Root Detection Bypass

Application developers sometimes hard code a detection logic per which an application successfully detects the presence of various SU binaries and stops execution of the application. One such example is demonstrated below. As you can see the app gives a popup of restriction and exits as soon user hits ok.



Now, we'll try and remove this restriction using Frida. First, it is recommended you install a Frida server in the device (Follow steps **here**). Next, we'll launch the server onto the device.

```
adb connect 192.168.27.105
adb shell "/tmp/frida-server &"
```

```
root@hex-VirtualBox:/home/hex# adb connect 192.168.27.105
connected to 192.168.27.105:5555
root@hex-VirtualBox:/home/hex# adb shell "/tmp/frida-server &"
```

Now, we'll first install frida with the command:

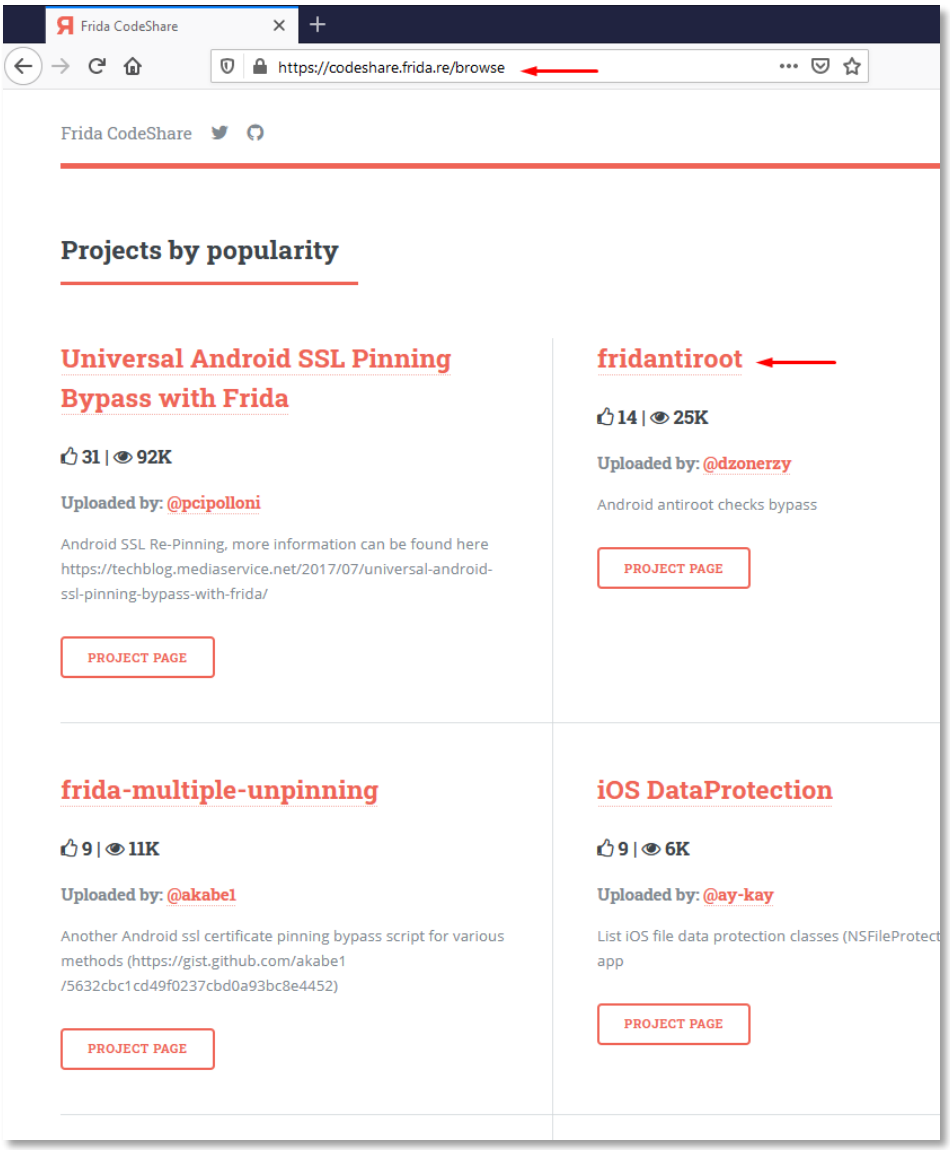
```
pip install frida
pip install frida-tools
```

After a successful install, we can see all the running process in the device on which frida server is running by the command:

```
frida-ps -U
```

```
root@hex-VirtualBox:/home/hex/frida-scripts# frida-ps -U
 PID  Name
----  ------------------------------
 120  adbd
1038  android.ext.services
1254  android.process.media
 268  audioserver
 259  batteryd
 269  cameraserver
1334  com.android.calendar
1054  com.android.deskclock
1371  com.android.email
 644  com.android.inputmethod.latin
1141  com.android.launcher3
 785  com.android.phone
1090  com.android.printspooler
1387  com.android.providers.calendar
1279  com.android.quicksearchbox
 654  com.android.systemui
1104  com.genymotion.genyd
1481  com.genymotion.superuser
1096  com.genymotion.systempatcher
  98  debuggerd
 108  debuggerd:signaller
 260  diskiod
 270  drmserver
2410  frida-server
 118  gatekeeperd
 283  genybaseband
 287  healthd
1995  in
   1  init
 272  installd
 273  keystore
 262  lmkd
 257  local_camera
```

As you can see that our app is running here. We have to bypass root detection here. We can either try and reverse engineer the jar files, create our own javascript code and bypass root detection or we can rely on code already created by a large community of developers on codeshare frida repo.
Weblink to the site is: **https://codeshare.frida.re/browse**



Here, we can see an antiroot script by dzonerzy. We'll run it with the following command:

```
frida -U --codeshare dzonerzy/fridantiroot -f
in.<package company>.<package name>
```

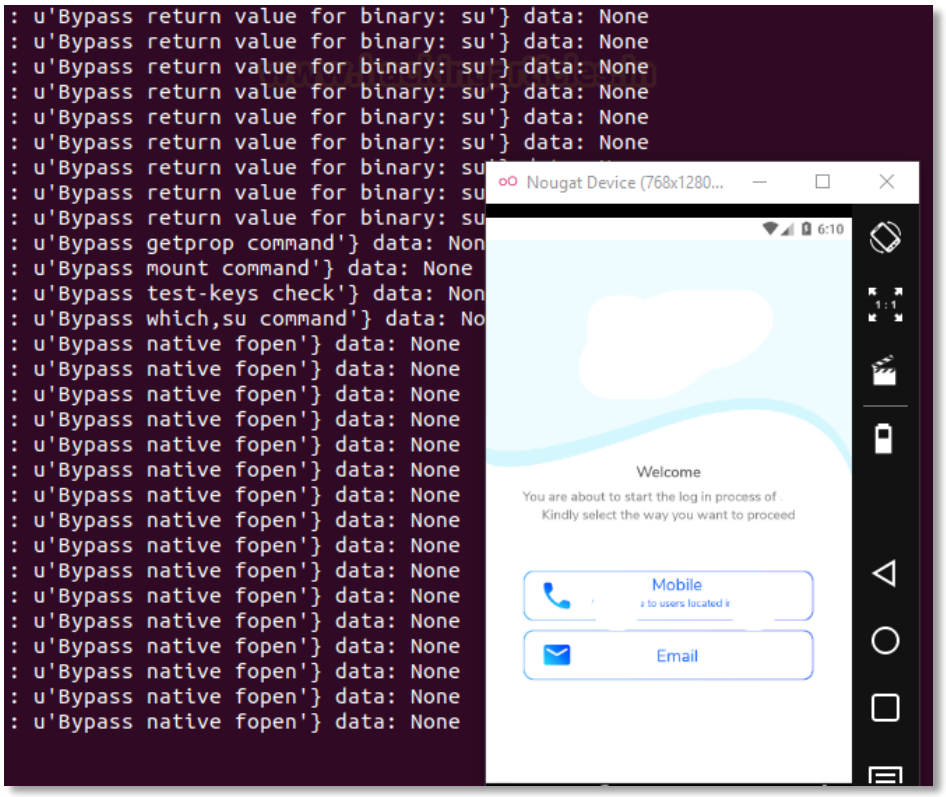
Now, press y to trust the project.

```
root@hex-VirtualBox:/home/hex/frida-scripts# frida -U --codeshare dzonerzy/fridantiroot
-f in.[redacted]

    ____
   / _  |   Frida 14.2.2 - A world-class dynamic instrumentation toolkit
  | (_| |
   > _  |   Commands:
  /_/ |_|       help      -> Displays the help system
  . . . .       object?   -> Display information about 'object'
  . . . .       exit/quit -> Exit
  . . . .
  . . . .   More info at https://www.frida.re/docs/home/
Spawning `in.[redacted]`...
Hello! This is the first time you're running this particular snippet, or the snippet's s
ource code has changed.

Project Name: fridantiroot
Author: @dzonerzy
Slug: dzonerzy/fridantiroot
Fingerprint: 0c8865d37a19c009bb4902742f4590301917aad93d072311d9e29eb8cd968586
URL: https://codeshare.frida.re/@dzonerzy/fridantiroot

Are you sure you'd like to trust this project? [y/N] y
```

Now, all that's left to do is press **"%resume"** to resume the execution with our hooked code!

```
Are you sure you'd like to trust this project? [y/N] y
Adding fingerprint 0c8865d37a19c009bb4902742f4590301917aad93d072311d9e29eb8cd968586 to t
he trust store! You won't be prompted again unless the code changes.
Spawned `in.nic.gimkerala`. Use %resume to let the main thread start executing!
[Nougat Device::in.[redacted]]-> %resume
```

And just like that, we can see that root detection has been successfully bypassed!



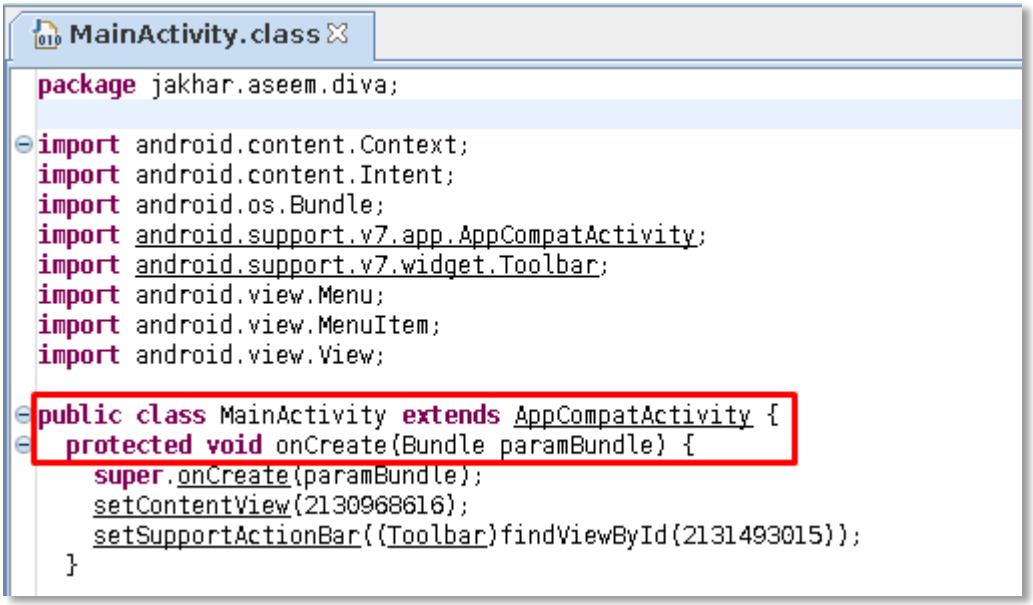
Hooking different methods in java

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 behavior 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.

```
adb shell pm path jakhar.aseem.diva
adb pull /data/app/jakhar.aseem.diva-dxAm4hRxYY4VgIq2X5zU6w==/base.apk
```

```
root@hex-VirtualBox:/home/hex# adb shell pm path jakhar.aseem.diva
package:/data/app/jakhar.aseem.diva-dxAm4hRxYY4VgIq2X5zU6w==/base.apk
root@hex-VirtualBox:/home/hex#
root@hex-VirtualBox:/home/hex# adb pull /data/app/jakhar.aseem.diva-dxAm4hRxYY4VgIq2X5zU6w==/base.apk
/data/app/jakhar.aseem.diva-dxAm4hRxYY4VgIq2X5zU6w==/base.apk: 1 file pulled. 33.3 MB/s (1502294 bytes i
n 0.043s)
root@hex-VirtualBox:/home/hex# ls base.apk
base.apk
```

Now, as explained in part 1 of this series (refer para 3 of the article **here**), 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.



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.

```
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);

	};
```

Explanation:

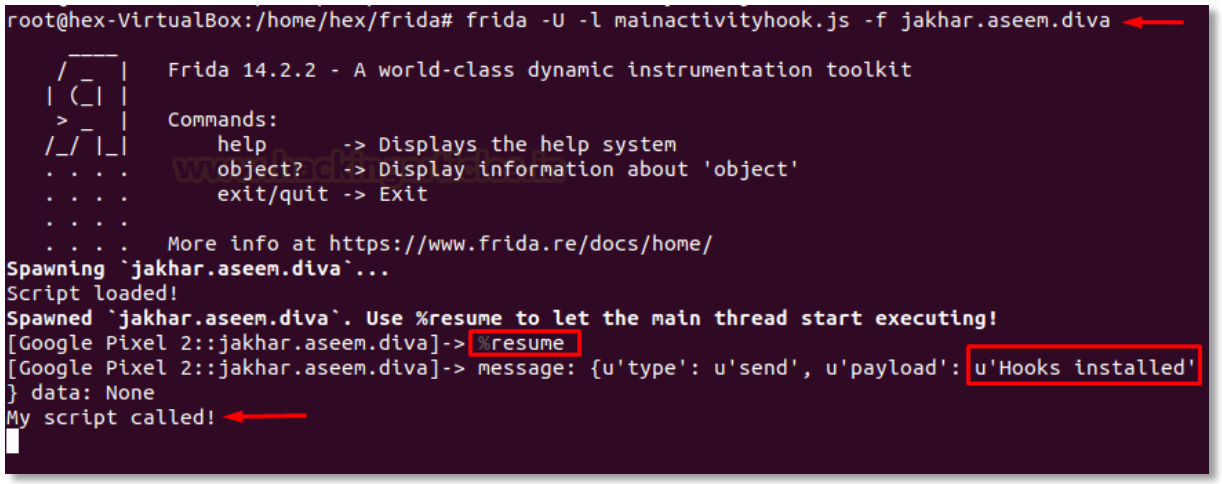
1. Any implementation of the hook is put inside **perform(function(){ //<code>**
2. The activity we want to hook (main activity) is put inside **use("jakhar.aseem.diva.MainActivity"),** and assign a variable to it. Here, **mainapp**
3. Now, **onCreate.implementation** sets a definition of the function.
4. Here, we can insert any code we can't 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.

```
hex@hex-VirtualBox:~/frida$ cat mainactivityhook.js
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");
});
hex@hex-VirtualBox:~/frida$
```

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. 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.

```
console.log("Hooked startChallenge() function");

Java.perform(function(){

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

       newstart.startChallenge.overload("android.view.View").implementation
= 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);

       };
```

```
root@hex-VirtualBox:/home/hex/frida# cat main_startchallenge.js
console.log("Hooked startChallenge() function");

Java.perform(function(){

var newstart = Java.use("jakhar.aseem.diva.MainActivity");
        newstart.startChallenge.overload("android.view.View").implementatio
n = 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);
        };
});
```

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.

```
root@hex-VirtualBox:/home/hex/frida# frida -U -l main_startchallenge.js -f
jakhar.aseem.diva --no-pause
     ____
    / _  |   Frida 14.2.2 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://www.frida.re/docs/home/
Spawning `jakhar.aseem.diva`...
Hooked startChallenge() function
Spawned `jakhar.aseem.diva`. Resuming main thread!
[Google Pixel 2::jakhar.aseem.diva]-> message: {u'type': u'send', u'payload
': u'MainActivity.startChallenge() is now started'} data: None
message: {u'type': u'send', u'payload': u'MainActivity.startChallenge() is
now started'} data: None
message: {u'type': u'send', u'payload': u'MainActivity.startChallenge() is
now started'} data: None
message: {u'type': u'send', u'payload': u'MainActivity.startChallenge() is
now started'} data: None
```

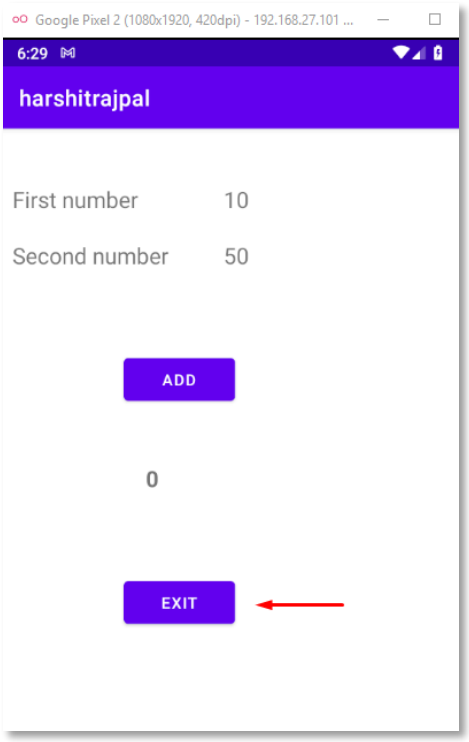
Hooking exit() method

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 (link **here**). It has a button that is performing an exit function. You can see a sample screenshot below:

activity_main.xml | AndroidManifest.xml | MainActivity.java

```
1  package com.example.harshitrajpal;
2
3  import ...
10
11 public class MainActivity extends AppCompatActivity {
12
13
14     Button add_button;
15     Button exit_button;
16     TextView a,b,sum;
17     double add=0;
18     @Override
19     protected void onCreate(Bundle savedInstanceState) {
20         super.onCreate(savedInstanceState);
21         setContentView(R.layout.activity_main);
22         a=(TextView)findViewById(R.id.num1);
23         b=(TextView)findViewById(R.id.num2);
24         add_button=(Button) findViewById(R.id.add_button);
25         sum=(TextView)findViewById(R.id.sum);
26         exit_button = (Button) findViewById(R.id.exit_button);
27
28         add_button.setOnClickListener(new View.OnClickListener(){
29             public void onClick(View v){
30                 sum.setText(Double.toString(returnValue()));
31             }
32         });
33
34         exit_button.setOnClickListener(new View.OnClickListener(){
35             public void onClick(View v){
36                 System.exit( status: 2);
37             }
38         });
39     }
40     double returnValue(){
41         double a1 = 10;
42         double b1 = 50;
43         add= a1 + b1;
44         return add;
45     }
46 }
```

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.

```
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!");
     };
});
```

```
root@hex-VirtualBox:/home/hex/frida# cat avoidexit.js
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!");
        };
});
root@hex-VirtualBox:/home/hex/frida#
```

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
```

```
root@hex-VirtualBox:/home/hex/frida# frida -U -l avoidexit.js -f com.example.harshitraj
pal --no-pause
     ____
    / _  |   Frida 14.2.2 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://www.frida.re/docs/home/
Spawning `com.example.harshitrajpal`...
Hooking on exit function
Spawned `com.example.harshitrajpal`. Resuming main thread!
[Google Pixel 2::com.example.harshitrajpal]-> message: {u'type': u'send', u'payload': u
"I've stopped java.lang.System.exit() now!"} data: None
message: {u'type': u'send', u'payload': u"I've stopped java.lang.System.exit() now!"} d
ata: None
```

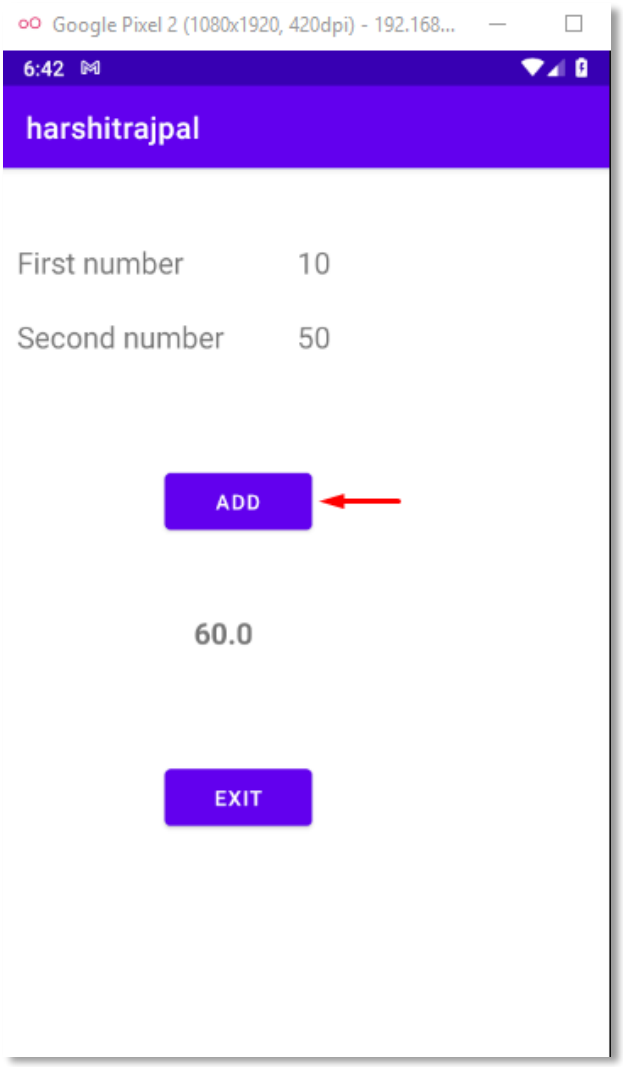
Hooking return value

We have hooked methods till now, but a return variable can also be hooked and its output be tampered with. In article 3 of this series, I had already demonstrated this using Objection tool but today we'll do this using Frida and our manual code. 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:

```
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;
            }
});
```

```
root@hex-VirtualBox:/home/hex/frida# cat retValueHook.js
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;
        }
});
root@hex-VirtualBox:/home/hex/frida#
```

Let's first run the program without loading our hook. We can see that the program outputs 60 which is the correct answer.

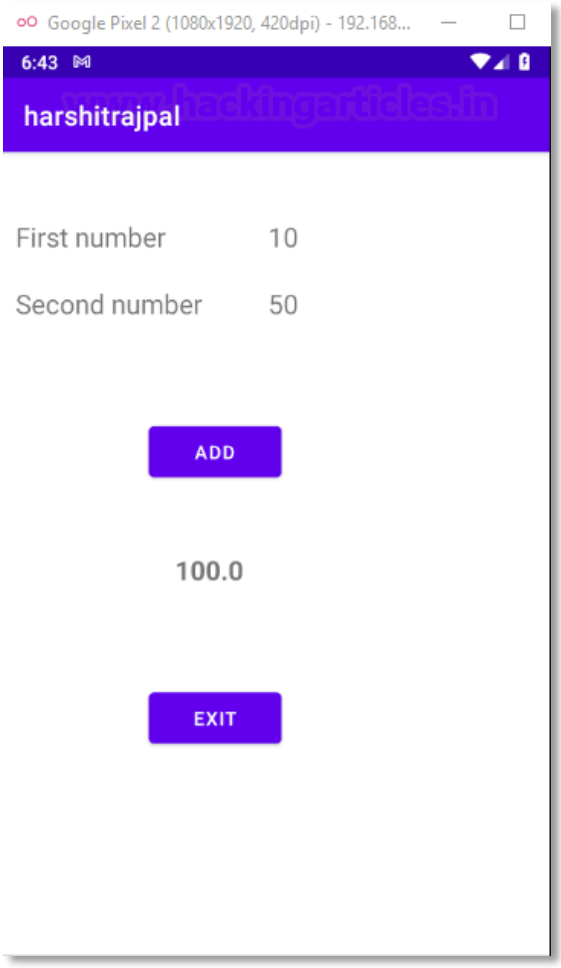

Google Pixel 2 (1080x1920, 420dpi) - 192.168...
6:42
harshitrajpal
First number
10
Second number
50
ADD
60.0
EXIT

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
```

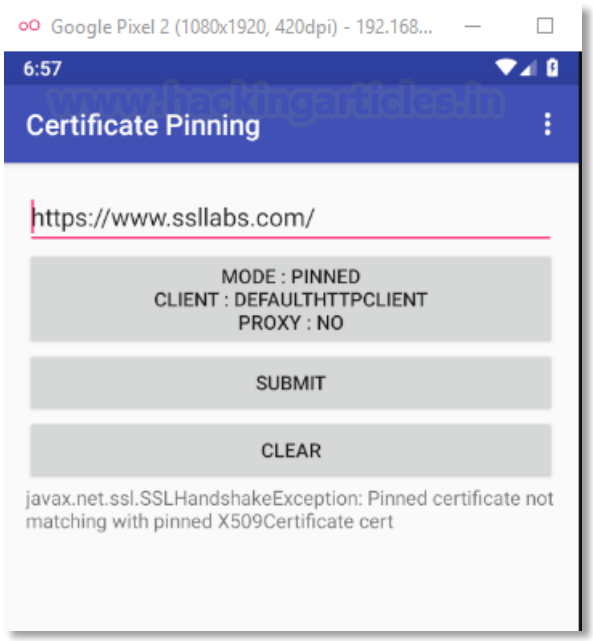
```
root@hex-VirtualBox:/home/hex/frida# frida -U -l retValueHook.js -f com.example.harshit
rajpal --no-pause
     ____
    / _  |   Frida 14.2.2 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://www.frida.re/docs/home/
Spawning `com.example.harshitrajpal`...
Hook for implementation of method
Spawned `com.example.harshitrajpal`. Resuming main thread!
[Google Pixel 2::com.example.harshitrajpal]->
```

And sure enough, the output gets tampered and 100 is returned now!

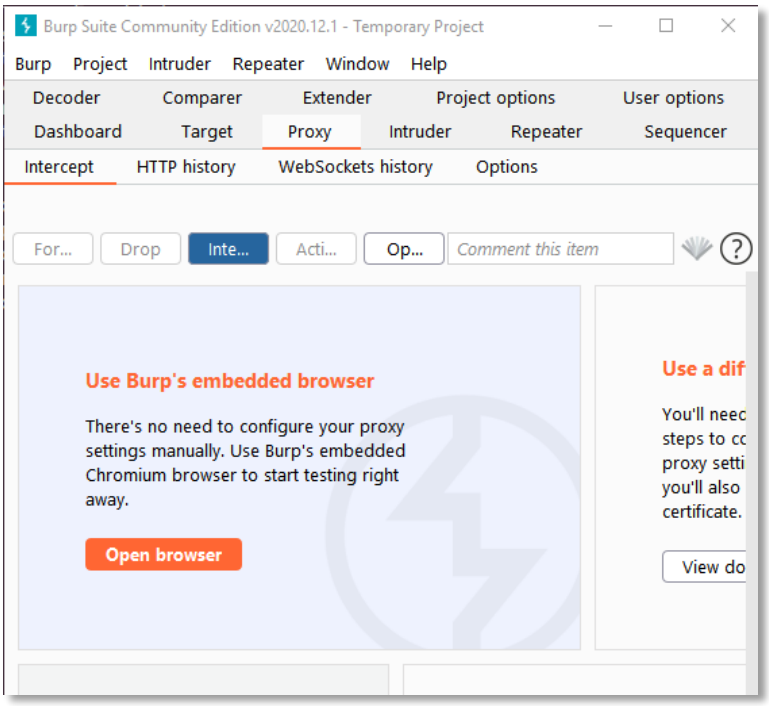


SSLPinning Bypass

Frida is most commonly used to bypass SSLPinning in android so that researchers and pen testers can intercept its network calls and conduct a traffic analysis. For the demo of this attack, I downloaded an application named “Certificate Pinning Demo”. For the demonstration of this attack, you must have your burp suite configured with your device (follow point 3 of the article **here**). Now, when I pin the client and send an HTTPS request, it throws an SSL error.



And sure enough, no communication is intercepted in burp suite as well.



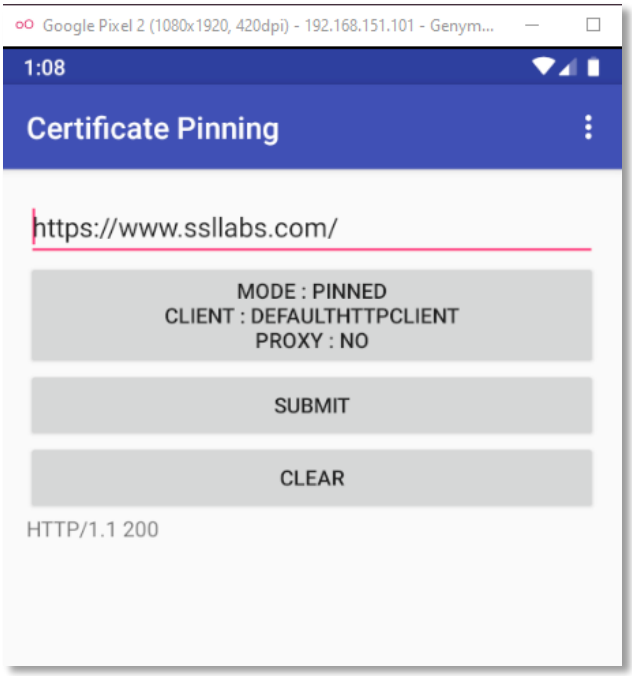
Now, on the codeshare repository **here**, akabe1 has put a great script to perform SSLPinning bypass. We'll use this script to perform the attack. Note that applications might have different code of pinning, so these codes need to be modified as and when required.

```
frida -U --codeshare akabe1/frida-multiple-unpinning -f
com.osfg.certificatepinning
```

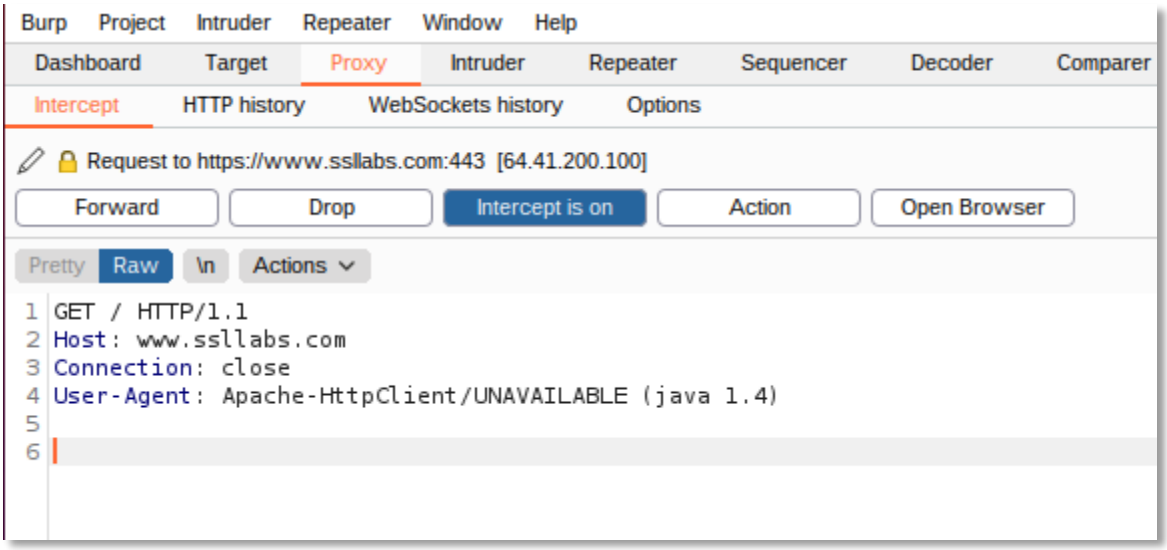
Type **%resume** once the script gets loaded.

```
root@hex-VirtualBox:/home/hex/frida# frida -U --codeshare akabe1/frida-multiple-unpinn
ing -f com.osfg.certificatepinning
     ____
    / _  |   Frida 14.2.2 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://www.frida.re/docs/home/
Spawning `com.osfg.certificatepinning`...
Hello! This is the first time you're running this particular snippet, or the snippet's
 source code has changed.

Project Name: frida-multiple-unpinning
Author: @akabe1
Slug: akabe1/frida-multiple-unpinning
Fingerprint: 0da412f7f7173208c78c18b97894b16000b9056de24fc39f19f69cc91ee26550
URL: https://codeshare.frida.re/@akabe1/frida-multiple-unpinning

Are you sure you'd like to trust this project? [y/N] y
Adding fingerprint 0da412f7f7173208c78c18b97894b16000b9056de24fc39f19f69cc91ee26550 to
 the trust store! You won't be prompted again unless the code changes.
Spawned `com.osfg.certificatepinning`. Use %resume to let the main thread start execut
ing!
[Google::com.osfg.certificatepinning]-> %resume
[Google::com.osfg.certificatepinning]->
======
[#] Android Bypass for various Certificate Pinning methods [#]
======
[-] OkHTTPv3 {1} pinner not found
[-] OkHTTPv3 {2} pinner not found
[-] OkHTTPv3 {3} pinner not found
```

And finally, when we now send a request to sslabs.com in pinned mode, we are able to get an HTTP 200 response code!



Surely, we are now able to intercept communication in burp suite as well.



Hooking in Python

Python coders can customize a whole fridascript to run in python environment using the python's frida package and API. This would make performing multiple processes in hooks easier. Here, I'll create a hook on startChallenge function as above.

```
jscode = """
console.log("Hooked startChallenge() function");
Java.perform(function(){
var newstart = Java.use("jakhar.aseem.diva.MainActivity");
            newstart.startChallenge.overload("android.view.View").implementation
= function(v){
                        //enter any implementation of startChallenge you want
                        //for demo I'm just sending an alert on console
                        send("MainActivity.startChallenge() is now started");
                        console.log("You clicked...but in vain!");
                        var ret =
this.startChallenge.overload("android.view.View").call(this);
            };
});
"""
import frida,sys
process = frida.get_usb_device().attach("jakhar.aseem.diva")
script = process.create_script(jscode)
print("*** Running Hook on startChallenge() now!")
script.load()
sys.stdin.read()
```

Now, every time user clicks on any button to start the challenge, the execution stops and our custom output is printed instead

.

```
root@hex-VirtualBox:/home/hex/frida# cat startChallenge.py

jscode = """
console.log("Hooked startChallenge() function");

Java.perform(function(){

var newstart = Java.use("jakhar.aseem.diva.MainActivity");
        newstart.startChallenge.overload("android.view.View").implementation = function(v){
                //enter any implementation of startChallenge you want
                //for demo I'm just sending an alert on console
                send("MainActivity.startChallenge() is now started");
                console.log("You clicked...but in vain!");
                var ret = this.startChallenge.overload("android.view.View").call(this);
        };
});

"""
import frida,sys

process = frida.get_usb_device().attach("jakhar.aseem.diva")
script = process.create_script(jscode)
print("*** Running Hook on startChallenge() now!")
script.load()
sys.stdin.read()
```

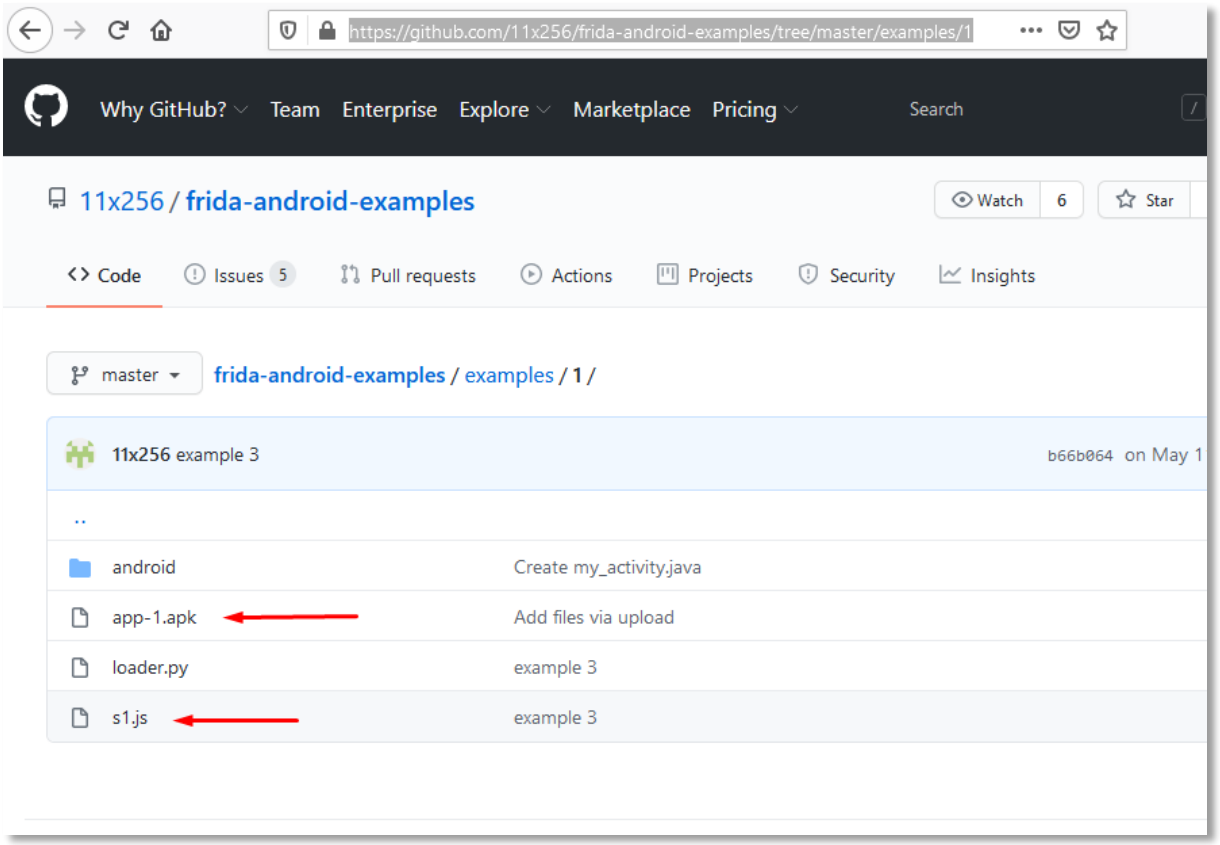
We, run this script using the command below:

```
python3 startChallenge.py
```

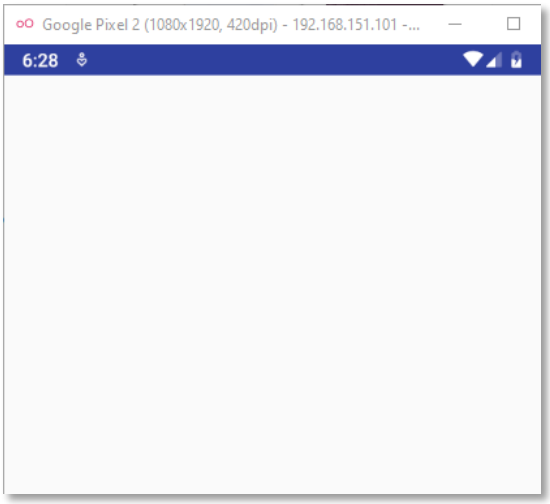
```
root@hex-VirtualBox:/home/hex/frida# python3 startChallenge.py
*** Running Hook on startChallenge() now!
Hooked startChallenge() function
You clicked...but in vain!
You clicked...but in vain!
You clicked...but in vain!
You clicked...but in vain!
You clicked...but in vain!
```

Let's Play a Game!

All the examples demonstrated till now are very basic. There are advanced hooking techniques to perform various different functions whose references I'll mention at the end. One such challenge I found was on 11×256's blog. In example #1, we have to intercept the APK, see what's happening behind the white screen, change its implementation and modify its behaviour. finally, we'll check logcat to see if our hook worked and the sum of our custom defined integers is thrown or not.
Follow the link **here** and download the sample apk.


https://github.com/11x256/frida-android-examples/tree/master/examples/1
Why GitHub?
Team
Enterprise
Explore
Marketplace
Pricing
Search
11x256 / frida-android-examples
Watch
6
Star
Code
Issues
5
Pull requests
Actions
Projects
Security
Insights
master
frida-android-examples / examples / 1 /
11x256 example 3
b66b064 on May 1
..
android
Create my_activity.java
app-1.apk
Add files via upload
loader.py
example 3
s1.js
example 3

First, after running the application in the emulator we saw just a plain white screen. That means something must probably be happening in the background.



We'll use drozer to see activities here:

```
run app.activity.info -a com.example.a11x256.frida_test
```

As you can see, **my_activity** is present. This means this is the activity responsible for the full white front screen.

```
dz> run app.activity.info -a com.example.a11x256.frida_test
Package: com.example.a11x256.frida_test
  com.example.a11x256.frida_test.my_activity
    Permission: null
```

Now, we'll use objection to watch what this class is actually doing. (Full objection tutorial **here**.)

```
objection --gadget com.example.a11x256.frida_test explore

android hooking watch class com.example.a11x256.frida_test --dump-args
--dump-return --dump-backtrace
```

Here, observe that **fun()** is being called. This has two int parameters, so, presumably, these two integers are getting performed a mathematical operation on.
Now, we write a code in javascript:

```
hex@hex-VirtualBox:~$ objection --gadget com.example.a11x256.frida_test explore
Using USB device `Google Pixel 2`
Agent injected and responds ok!

     _   _         _   _
 ___| |_|_|___ ___| |_|_|___ ___
| . | . | | -_|  _|  _| | . |   |
|___|___| |___|___|_| |_|___|_|_|
      |___|(object)inject(ion) v1.9.6

     Runtime Mobile Exploration
        by: @leonjza from @sensepost

[tab] for command suggestions
com.example.a11x256.frida_test on (Android: 9) [usb] # android hooking watch class com.e
xample.a11x256.frida_test.my_activity --dump
-args --dump-return --dump-backtrace
(agent) Hooking com.example.a11x256.frida_test.my_activity.fun(int, int)
(agent) Hooking com.example.a11x256.frida_test.my_activity.onCreate(android.os.Bundle)
(agent) Registering job 5203865403581. Type: watch-class for: com.example.a11x256.frida_
test.my_activity
com.example.a11x256.frida_test on (Android: 9) [usb] # (agent) [5203865403581] Called co
m.example.a11x256.frida_test.my_activity.fun(int, int)
(agent) [5203865403581] Called com.example.a11x256.frida_test.my_activity.fun(int, int)
(agent) [5203865403581] Called com.example.a11x256.frida_test.my_activity.fun(int, int)
(agent) [5203865403581] Called com.example.a11x256.frida_test.my_activity.fun(int, int)
(agent) [5203865403581] Called com.example.a11x256.frida_test.my_activity.fun(int, int)
(agent) [5203865403581] Called com.example.a11x256.frida_test.my_activity.fun(int, int)
(agent) [5203865403581] Called com.example.a11x256.frida_test.my_activity.fun(int, int)
(agent) [5203865403581] Called com.example.a11x256.frida_test.my_activity.fun(int, int)
(agent) [5203865403581] Called com.example.a11x256.frida_test.my_activity.fun(int, int)
com.example.a11x256.frida_test on (Android: 9) [usb] # exit
Exiting...
Asking jobs to stop...
Unloading objection agent...
```

```
console.log("Script loaded successfully ");
Java.perform(function x() { //Silently fails without the sleep from the python
code
    console.log("Inside java perform function");
    //get a wrapper for our class
    var my_class = Java.use("com.example.a11x256.frida_test.my_activity");
    //replace the original implmenetation of the function `fun` with our custom
function
    my_class.fun.implementation = function (x, y) {
        //print the original arguments
        console.log("original call: fun(" + x + ", " + y + ")");
        //call the original implementation of `fun` with args (2,5)
        var ret_value = this.fun(2, 5);
        return ret_value;
    }
});
```

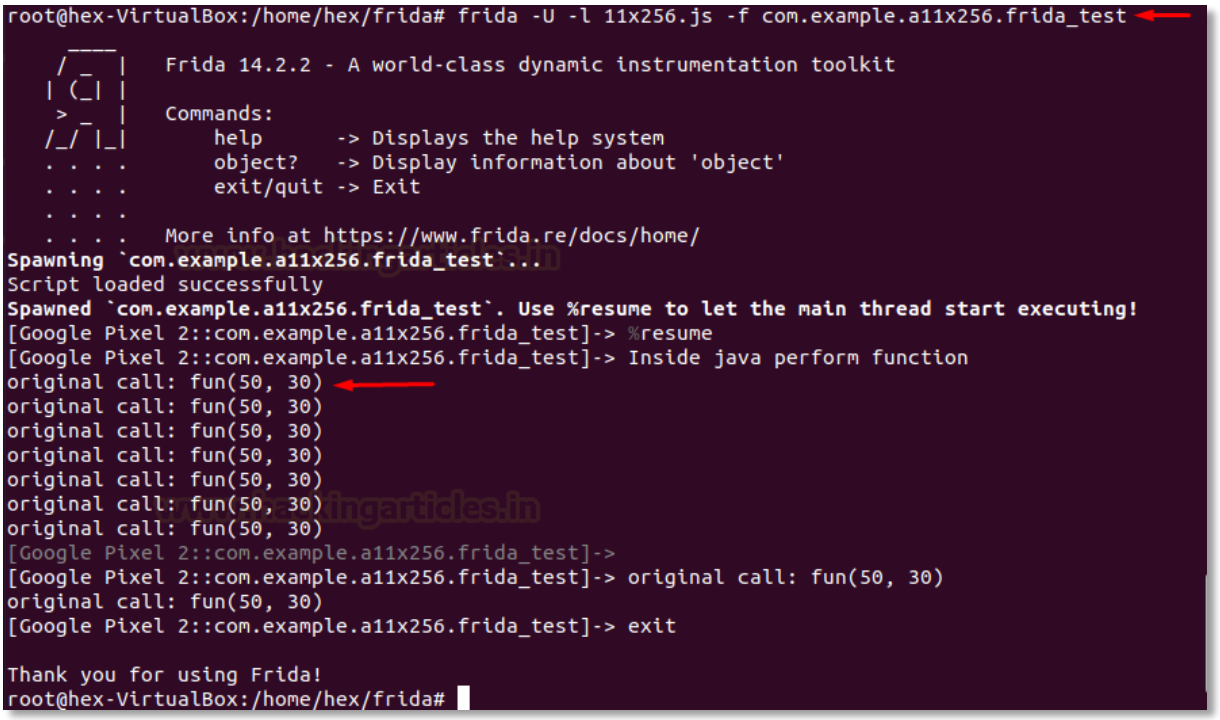
This code does nothing but defines fun() function and specifies 2 and 5 as our own integers on which some mathematical function will be performed. but before that, the script also intercepts and displays the original call and obviously the original integers!

```
root@hex-VirtualBox:/home/hex/frida# cat 11x256.js
console.log("Script loaded successfully ");
Java.perform(function x() { //Silently fails without the sleep from the python code
    console.log("Inside java perform function");
    //get a wrapper for our class
    var my_class = Java.use("com.example.a11x256.frida_test.my_activity");
    //replace the original implmenetation of the function `fun` with our custom function
    my_class.fun.implementation = function (x, y) {
        //print the original arguments
        console.log("original call: fun(" + x + ", " + y + ")");
        //call the original implementation of `fun` with args (2,5)
        var ret_value = this.fun(2, 5);
        return ret_value;
    }
});
root@hex-VirtualBox:/home/hex/frida#
```

Let's fire it up using frida:

```
frida -U -l 11x256.js -f com.example.a11x256.frida_test
```

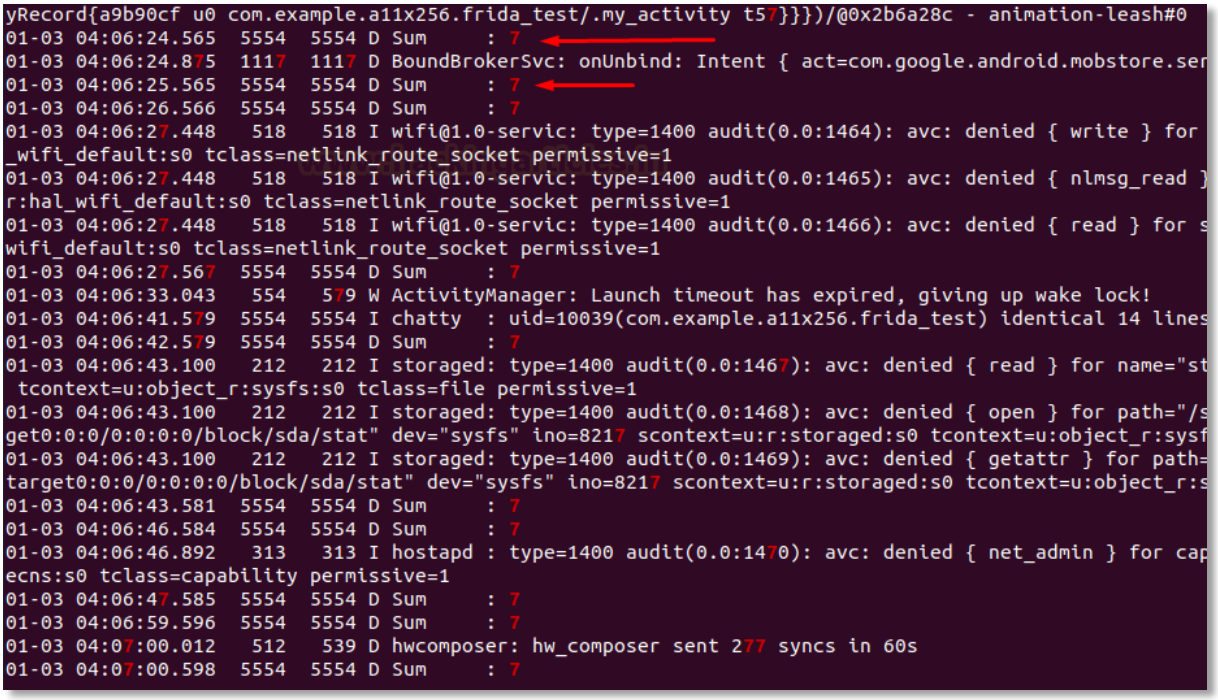
As we can see, the original call had two integers namely, 50 and 30.



Let's quickly check logcat and see what is happening in the background.

```
adb logcat | grep frida_test
```

As we can see in the screenshot down below, a mathematical Sum of type Double is being repeatedly called. This is similar to the behaviour of the app we just installed that was calling a method called fun after every second. Hence, it is safe to conclude that fun() is adding two integers. Original numbers to be added were 50 and 30, which we not only intercepted and dumped but also changed to 2 and 5 and the sum of 2 and 5 is now being called as evident in logcat.



If you want an advanced look into Frida and reverse engineering,In this publication, we'll explain the basics of Frida, how to create your own Frida script, hook it into processes and perform various functions. Needless to say, there is no end to what a program can do, therefore, there is no limit on frida's applications, hence, this publication is only restricted to basics.