Android Mobile Pentest 101

Lecture 7 – Hooking With Frida

Goal: Known how to use Frida to hook into function

Description

- Dynamic instrumentation (tracing, profiling, and debugging the execution of an app during runtime) toolkit for developers, reverse-engineers, and security researchers.
- Good things of Frida:

Scriptable

Inject your own scripts into black box processes. Hook any function, spy on crypto APIs or trace private application code, no source code needed. Edit, hit save, and instantly see the results. All without compilation steps or program restarts.

Portable

Works on Windows, macOS, GNU/Linux, iOS, Android, and QNX. Install the Node.js bindings from npm, grab a Python package from PyPI, or use Frida through its Swift bindings, .NET bindings, Qt/Qml bindings, or C API.

Free

Frida is and will always be free software (free as in freedom). We want to empower the next generation of developer tools, and help other free software developers achieve interoperability through reverse engineering.

Battle-tested

We are proud that NowSecure is using Frida to do fast, deep analysis of mobile apps at scale. Frida has a comprehensive test-suite and has gone through years of rigorous testing across a broad range of use-cases.

Installation -> Client (real pc)

- Install Frida CLI Tools:

pip3 install frida-tools

- Install Frida Python bindings (we mainly use it in this lecture)

pip3 install frida

- Test if frida successful installed

```
 ~/ frida
Usage: frida [options] target

frida: error: target file, process name or pid must be specified
 ~/ python3
Python 3.7.0 (default, Jul 23 2018, 20:22:55)
[Clang 9.1.0 (clang-902.0.39.2)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import frida
>>>
```

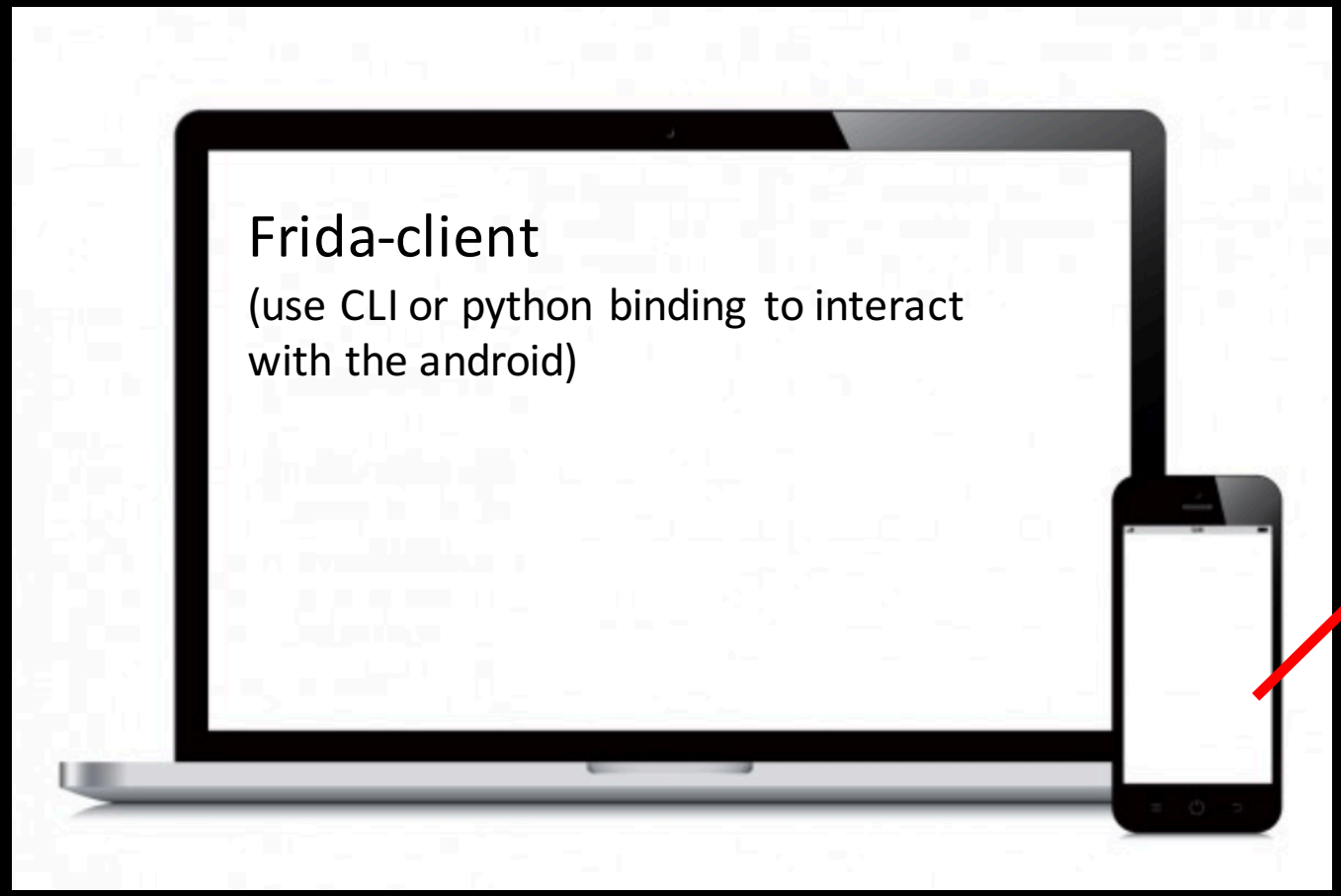
Installation -> Server (virtual phone)

- A frida-server binary from the release page, frida-server-12.1.2-android-x86.xz at this lecture (if you don't know your phone architecture, just try all binary arm/arm64/x86-64/x86)
- The frida-server version should match your Frida version.

```
 ~/Downloads/ adb push frida-server-12.1.2-android-x86 /data/local/tmp/frida-server
frida-server-12.1.2-android-x86: 1 file pushed. 44.3 MB/s (27961592 bytes in 0.602s)
 ~/Downloads/ adb shell
root@vbox86p:/ # cd /data/local/tmp/
root@vbox86p:/data/local/tmp # ./frida-server &
[1] 5220
```

Installation

- Our setup will look like this:



Frida-server
(listen on port 27042 by default)

Installation

- Check if our setup correctly, from real pc, type command:

frida-ps -U

```
 ~/Desktop/mobile/tools/ frida-ps -U

 PID  Name
----  ----------------------------------------
 128  adbd
 456  batteryd
3387  com.android.calendar
2816  com.android.deskclock
1930  com.android.exchange
 893  com.android.inputmethod.latin
1086  com.android.launcher3
1011  com.android.phone
1251  com.android.smspush
 855  com.android.systemui
1503  com.android.vending
3434  com.android.vending:instant_app_installer
1047  com.genymotion.genyd
 984  com.genymotion.systempatcher
3193  com.google.android.gms
 936  com.google.android.gms.persistent
4353  com.google.android.gms.ui
3662  com.google.android.gms.unstable
1292  com.google.process.gapps
```

success

Python bindings

- As I said above, we will mainly focus on “frida python bindings”, let learn about it.
- We will use original “InsecureBankv2” app as an example (so delete and reinstall the original apk first ☺)

Python bindings

- To use frida in python, we import it

```
import frida
```

- We use get_usb_device() function to get our device information

```
device = frida.get_usb_device()
```

```
Device(id="192.168.56.101:5555", name="Unknown Samsung Galaxy S6 - 5.1.0 - API 22 - 1440x2560", type='usb')
[Finished in 0.2s]
```

- Then we spawn the app:

```
pid=device.spawn("com.android.insecurebankv2")
device.resume(pid)
```

- We now got a pid of spawned app, then attach it to begin our pentest journey:

```
session = device.attach(id)
```

- We almost done the python template, let inject the javascript code:

```
script = session.create_script(hook_script)
script.load()
```

Python bindings

- The python code that we've created so far:

```python
import frida
import time

device = frida.get_usb_device() # get device information
pid = device.spawn("com.android.insecurebankv2") # spawn app
device.resume(pid) # resumes it pid

time.sleep(1) # sleep 1 to avoid crash (sometime)

session=device.attach(pid)

hook_script="""
"""

script=session.create_script(hook_script)
script.load()

raw_input('...?') # prevent terminate
```

Python bindings

- So you may wonder what the hook_script is, it's the commands that we provide to Frida using its Javascript API. With it, we can work with Java functions and objects directly.
- First notice here is, the code we entered is wrapped in a Java.perform(function(){ ... }) which is a requirement of Frida Java API. So hook_script will look like this:

```
hook_script="""
Java.perform(function ()
{
// do something
});
"""
```

- We are going to decide which we want to hook, back to our app, did you remember the weak crypto in lecture 3 (Static analysis)?
- We have to understand the crypt, simulate it in python, blah...blah.... what if you can't find the key, or the crypt is more complex? Im not a crypto guys so i will give up...
- Luckily, Frida save the day ☺

Python bindings

- We found the decrypt code in com/android/insecurebankv2/CryptoClass.class

```
public String aesDeccryptedString(String paramString)
  throws UnsupportedEncodingException, InvalidKeyException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidAlgorithmParameterExcept
{
  byte[] arrayOfByte = this.key.getBytes("UTF-8");
  this.cipherData = aes256decrypt(this.ivBytes, arrayOfByte, Base64.decode(paramString.getBytes("UTF-8"), 0));
  this.plainText = new String(this.cipherData, "UTF-8");
  return this.plainText;
}
```

- We know that it will take crypt-text, decrypt then return plainText, we don't even care what it does inside. The task is very clear now, just call it function using our crypt-text as input ☺

Python bindings

- So, how to call the function using Frida? We are going to use Java.choose()
- From the official document:

- `Java.choose(className, callbacks)`: enumerate live instances of the `className` class by scanning the Java heap, where `callbacks` is an object specifying:
  - `onMatch: function (instance)`: called once for each live instance found with a ready-to-use `instance` just as if you would have called `Java.cast()` with a raw handle to this particular instance. This function may return the string `stop` to cancel the enumeration early.
  - `onComplete: function ()`: called when all instances have been enumerated

- So, it will scan the heap, find instance of className we provide, on matching, the onMatch callback will be triggered, and we call the function of this class in the instance.

Python bindings

- Talk is cheap, I show you the code ☺

```
import frida
import time

device = frida.get_usb_device() # get device information
pid = device.spawn("com.android.insecurebankv2") # spawn app
device.resume(pid) # resumes it pid

time.sleep(1) # sleep 1 to avoid crash (sometime)

session=device.attach(pid)

hook_script="""
Java.perform
(
    function ()
    {
    console.log("Inside the hook_script");
    class_CryptoClass = Java.choose('com.android.insecurebankv2.CryptoClass',
        {
            onMatch : function(instance)
            {
                console.log("Found instance: "+instance);
                console.log("Result of decrypt:"+instance.aesDeccryptedString('DTrW2VXjSoFdg0e61fHxJg==' ));
            },
            onComplete: function(){ console.log("end")}
        });
    }
);
"""

script=session.create_script(hook_script)
script.load()

input('...?') # prevent terminate
```

className

Instance of
className
found on heap

Use instance
to Call function

Python bindings

- And here the result:

```
Inside the hook_script
Found instance: com.android.insecurebankv2.CryptoClass@2aec3a8b
Result of decrypt:Dinesh@123$
end
...?
```

- As you see, Dinesh@123$ is the password of user dinesh ☺

Python bindings

- We move to another fun ☺
- Remember the root detection code? This time we will modify it, let's Frida!
- We found the code in com/android/insecurebankv2/PostLogin.class

```
void showRootStatus()
{
  int i;
  if ((!doesSuperuserApkExist("/system/app/Superuser.apk")) && (!doesSUexist())) {
    i = 0;
  } else {
    i = 1;
  }
  if (i == 1)
  {
    this.root_status.setText("Rooted Device!!");
    return;
  }
  this.root_status.setText("Device not Rooted!!");
}
```

- As you see, if doesSuperuserApkExist(String paramString) return true, then root detected. By default, we pass this check, but to see some fun, we make it detect us ☺ => make it true!

Python bindings

- The return of this function is a bool val, so we hijack it, no need to care about the check code

```
private boolean doesSuperuserApkExist(String paramString)
{
  return Boolean.valueOf(new File("/system/app/Superuser.apk").exists()).booleanValue() == true;
}
```

- How? This time we use Java.use()
- From the official document:

- `Java.use(className)`: dynamically get a JavaScript wrapper for `className` that you can instantiate objects from by calling `$new()` on it to invoke a constructor. Call `$dispose()` on an instance to clean it up explicitly (or wait for the JavaScript object to get garbage-collected, or script to get unloaded). Static and non-static methods are available, and you can even replace a method implementation and throw an exception from it:

```
Java.perform(function () {
    var Activity = Java.use("android.app.Activity");
    var Exception = Java.use("java.lang.Exception");
    Activity.onResume.implementation = function () {
        throw Exception.$new("Oh noes!");
    };
});
```

Python bindings

- In short, Java.use() allow us to overwrite function in class.
- First, we use the class which got this function

```
Java.use('com.android.insecurebankv2.PostLogin')
```

- Then we override the code using implementation:

```
class_PostLogin.doesSuperuserApkExist.implementation  = function ()
{
        //whatever, in this case, return true
}
```

- Done

Python bindings

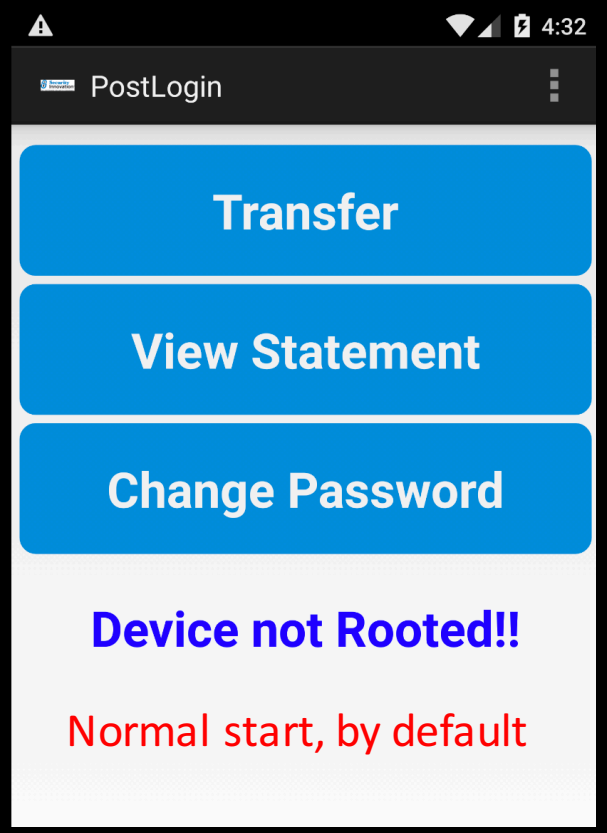
- Here the code:

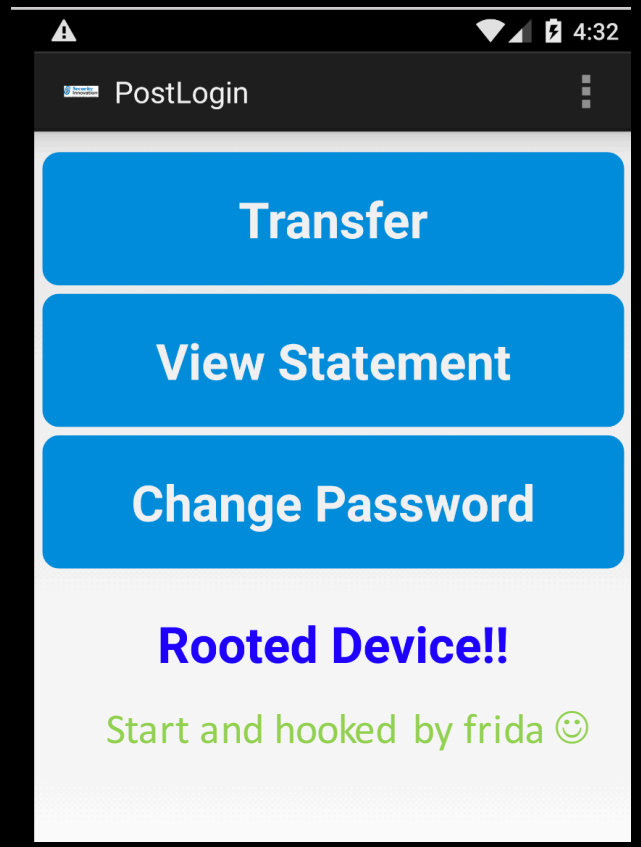
```python
import frida
import time

device = frida.get_usb_device() # get device information
pid = device.spawn("com.android.insecurebankv2") # spawn app
device.resume(pid) # resumes it pid

time.sleep(1) # sleep 1 to avoid crash (sometime)

session=device.attach(pid)

hook_script="""
Java.perform
(
    function ()
    {
    console.log("Inside the hook_script");
    class_PostLogin = Java.use('com.android.insecurebankv2.PostLogin');

    class_PostLogin.doesSuperuserApkExist.implementation = function (x)
        {
        return true;
        };
    }
);
"""

script=session.create_script(hook_script)
script.load()

input('...?') # prevent terminate
```

Use implementation to override function doesSuperuserApkExist content

Python bindings

- Come to our virtual phone, login in, and we got root detected status (>_<!)





Python bindings

- Another example, when we pentest the app, we always meet some client-side check, with frida, bypass it's easy!
- Look at this code in com/android/insecurebankv2/ChangePassword$RequestChangePasswordTask.class

```
public void postData(String paramString)
  throws ClientProtocolException, IOException, JSONException, InvalidKeyException, NoSuchAlgorithmException, NoSuch
{
  DefaultHttpClient localDefaultHttpClient = new DefaultHttpClient();
  StringBuilder localStringBuilder = new StringBuilder();
  localStringBuilder.append(this.this$0.protocol);
  localStringBuilder.append(this.this$0.serverip);
  localStringBuilder.append(":");
  localStringBuilder.append(this.this$0.serverport);
  localStringBuilder.append("/changepassword");
  HttpPost localHttpPost = new HttpPost(localStringBuilder.toString());
  ArrayList localArrayList = new ArrayList(2);
  localArrayList.add(new BasicNameValuePair("username", this.this$0.uname));
  localArrayList.add(new BasicNameValuePair("newpassword", this.this$0.changePassword_text.getText().toString()));
  localHttpPost.setEntity(new UrlEncodedFormEntity(localArrayList));
  ChangePassword.access$002(this.this$0, Pattern.compile("((?=.*\\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%]).{6,20})"));
  ChangePassword.access$102(this.this$0, ChangePassword.access$000(this.this$0).matcher(this.this$0.changePassword_
```

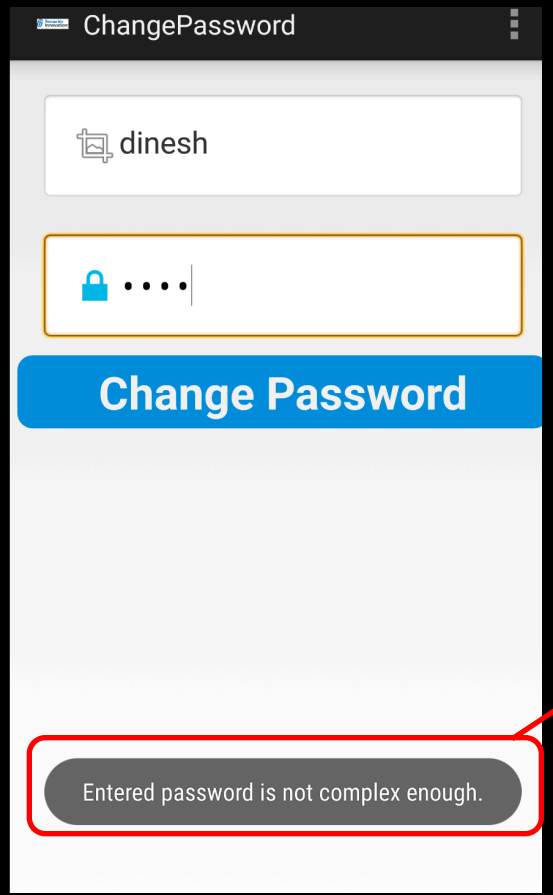
Python bindings

- Notice the regex:

((?=.*\\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%]).{6,20})

This regex is made to check the complexity of the password, so if we change our password to something simple, this regex prevent us

- I'm simple man, I want to use "1234" as my new pass:



Failed because password require length 6->20, "1234" is just 4

Python bindings

- Of course its failed because of the regex, we will use frida to change this regex. This time we don't inject to app class, because the check use method from java package, not our code.

```
Pattern.compile("((?=.*\\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%]).{6,20})"));
```

from

```
import java.util.regex.Pattern;
```

- So we hook it:

regex_Pattern_hook = Java.use('java.util.regex.Pattern')

- We cannot simply use regex_Pattern_hook.compile.implementation because in Pattern package there are many compile methods (compile(String x); compile(String x,int y); etc…), so frida will be confused
- So we have to tell frida which method we are going to use, in this case: compile(String x)
- We use overload:

regex_Pattern_hook.compile.overload("java.lang.String").implementation

Python bindings

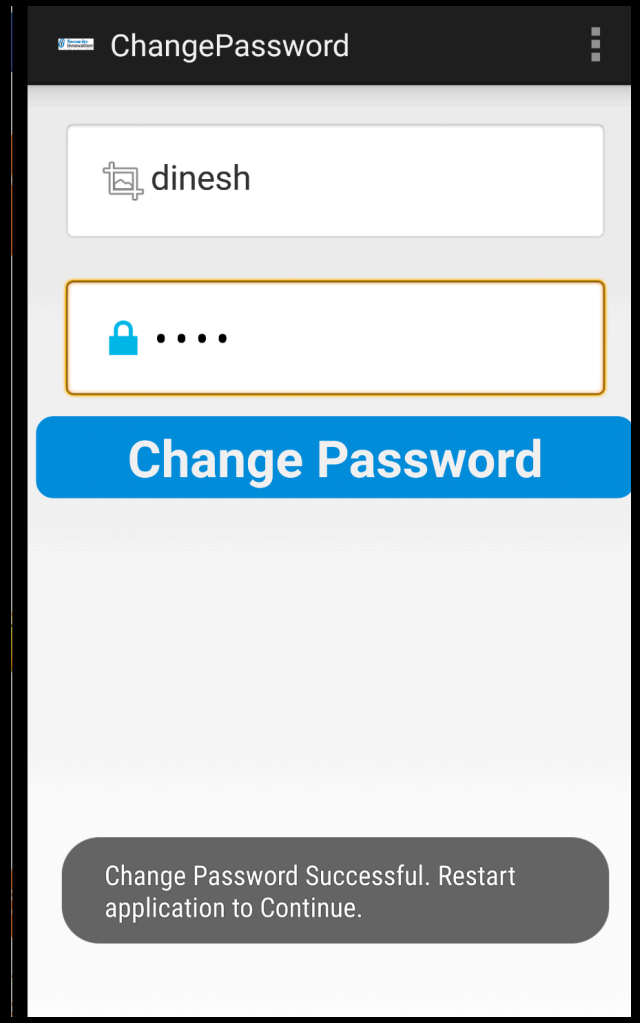
- Here the code:

```python
import frida
import time

device = frida.get_usb_device() # get device information
pid = device.spawn("com.android.insecurebankv2") # spawn app
device.resume(pid) # resumes it pid
time.sleep(1) # sleep 1 to avoid crash (sometime)
session=device.attach(pid)

hook_script="""

Java.perform
(
    function ()
    {
    console.log("Inside the hook script");
    regex_Pattern_hook = Java.use('java.util.regex.Pattern');
    regex_Pattern_hook.compile.overload("java.lang.String").implementation = function (x)
        {
            return this.compile(".*");
        }
    }
);
"""

script=session.create_script(hook_script)
script.load()

input('...?') # prevent terminate
```

Change compile method by calling the original compile with ".*" regex provided

Python bindings

- ***Sometime frida hooking will make the app suspended or idle (maybe caused by old android mobile), we just reload the script and it works like a charm***
- Let test it:



Python bindings

- Login to confirm:

```
.IInputMethodClient$Stub$Proxy@3e595ebe attribute=null, token = android.os.BinderProxy@de2ffc2
D/Successful Login:(19113): , account=dinesh:1234
```

- There are some more useful frida api when dealing with python: send, recv and rpc

Read full docs here:
https://www.frida.re/docs/javascript-api/