

Android Mobile Pentest 101

© tsug0d, September 2018

Bài 7 – Hooking Với Frida

Mục tiêu: Sử dụng được Frida để hook

Description

- Là một Dynamic instrumentation (tracing, profiling, và debugging trong lúc app chạy) toolkit cho developers, reverse-engineers, và security researchers.
- Một số đặc tính nổi bật của 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 (máy thật)

- Cài đặt Frida CLI Tools:

`pip3 install frida-tools`

- Cài đặt Frida Python bindings (Sử dụng chính trong bài viết)

`pip3 install frida`

- Kiểm tra lại xem frida đã cài thành công chưa:

```
❯ ~/ 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 (điện thoại ảo)

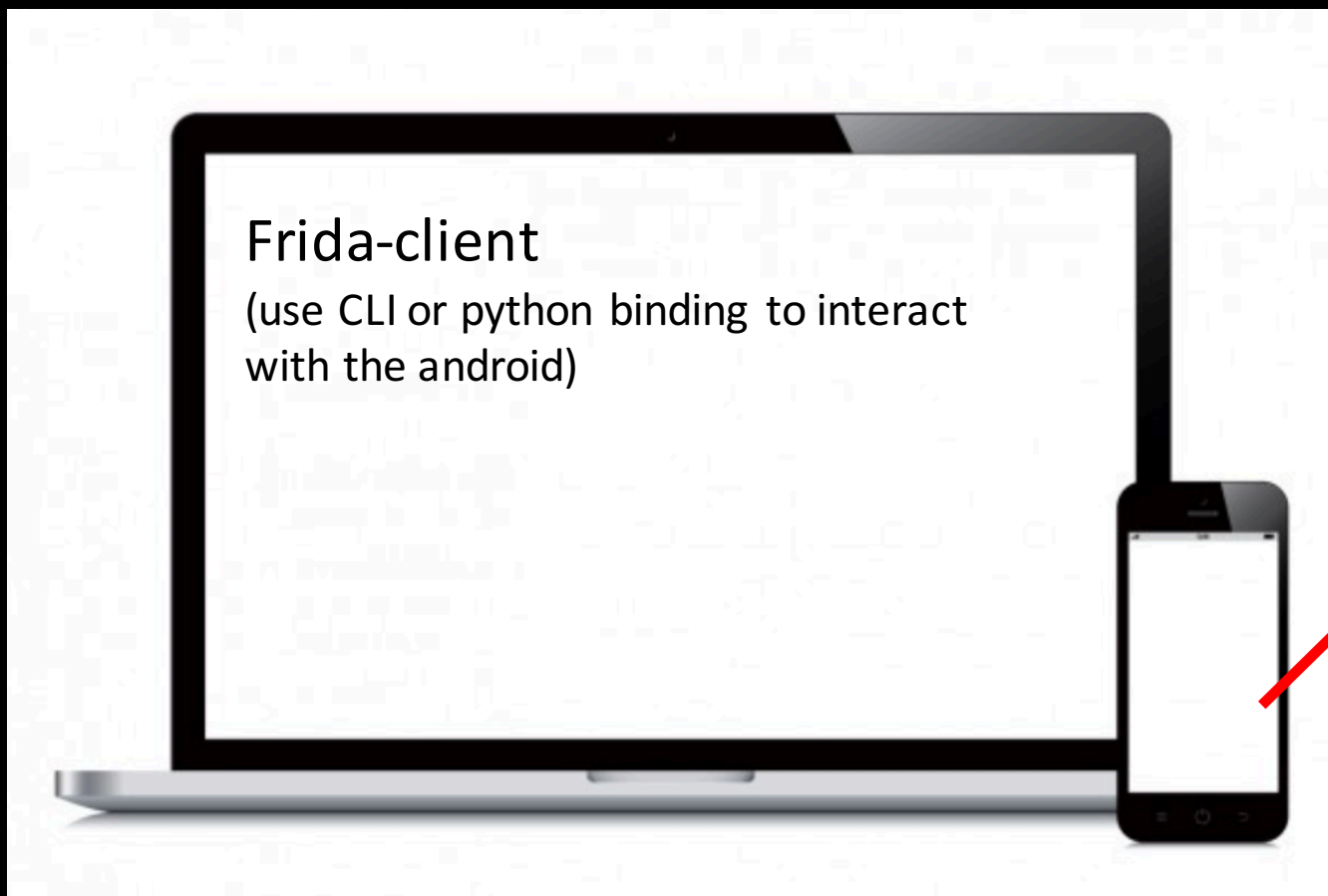
- Tải **frida-server** binary từ trang [release page](#), cụ thể ở bài này là **frida-server-12.1.2-android-x86.xz** (Nếu không rõ thì thử hết binary luôn arm/arm64/x86-64/x86)
- **frida-server** version phải đúng với Frida version.

```
🍏 ~/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

- Tổng quan thì setup nó sẽ như thế này



Frida-server
(mặc định listen trên port 27042)

Installation

- Kiểm tra xem setup đúng chưa, từ máy thật, gõ lệnh:

`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

Thành công

Python bindings

- Như đã nói ở trên, chúng ta sẽ tập trung vào “**frida python bindings**”.
- Sử dụng file apk gốc “**InsecureBankv2**” để làm ví dụ (nhớ xóa cái cũ đi và cài lại)

Python bindings

- Để sử dụng frida trong python, ta import nó:

```
import frida
```

- Bây giờ sử dụng `get_usb_device()` function để lấy thông tin về device

```
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]
```

- Rồi spawn app lên:

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

```
device.resume(pid)
```

- Bây giờ chúng ta đã có được pid của app vừa spawn, attach nó tạo session:

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

- Giờ thì inject đoạn hook_script của chúng ta vào chương trình thôi:

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

```
script.load()
```

Python bindings

- Đoạn python code diễn giải ở trên:

```
1  import frida
2  import time
3
4  device = frida.get_usb_device() # get device information
5  pid = device.spawn("com.android.insecurebankv2") # spawn app
6  device.resume(pid) # resumes it pid
7
8  time.sleep(1) # sleep 1 to avoid crash (sometime)
9
10 session=device.attach(pid)
11
12 hook_script="""
13 """
14
15 script=session.create_script(hook_script)
16 script.load()
17
18 raw_input('...?') # prevent terminate
```

Python bindings

- Tới lúc này bạn sẽ tự hỏi `hook_script` là cái gì đúng không, nó là các lệnh chúng ta cung cấp cho Frida sử dụng Javascript API. Với nó, chúng ta có thể tương tác trực tiếp với Java functions và objects.
- Lưu ý đoạn code chúng ta cung cấp cho frida sẽ nằm bên trong `Java.perform(function(){ ... })`, cái này là bắt buộc, kiểu cú pháp của Frida Java API. `hook_script` sẽ như thế này:

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

- Bây giờ quyết định hook cái gì nhé, trở về cái app nè, nhớ lỗi weak crypto ở [bài 3 không \(Phân tích tĩnh\)?](#)
- Lúc đó thì cần phải hiểu crypto, rồi mô phỏng lại trên python, blah...blah.... Lỡ lúc đó không tìm được key, hay đoạn crypto quá vãi hà thì chịu hả?
- Rất may mắn, Frida không xoắn mấy cái này 😊

Python bindings

- Chúng ta tìm được đoạn code decrypt trong [com/android/insecurebankv2/CryptoClass.class](https://github.com/AndroidInsecureBankv2/CryptoClass.class)

```
public String aesDecryptedString(String paramString)
    throws UnsupportedOperationException, InvalidKeyException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidAlgorithmParameterException
{
    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;
}
```

- Biết rằng nó nhận vào crypt-text, decrypt và trả về plainText, không cần quan tâm nó làm gì bên trong. Rất rõ ràng, chỉ cần gọi function đó lên sử dụng crypt-text ta muốn là okay 😊

Python bindings

- Vậy gọi function lên bằng Frida như nào? Chúng ta sẽ sử dụng `Java.choose()`
- Trích từ document frida gốc:

- `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

- Nó sẽ scan trên heap, tìm instance của `className` ta muốn, nếu khớp, `onMatch` callback sẽ được thực thi, dựa vào đó ta gọi hàm của instance tìm được.

Python bindings

- Code hiện giờ nó sẽ như thế này:

```
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.aesDecryptedString('DTrw2VXjSoFdg0e61fHxJg=='));
            },
            onComplete: function(){ console.log("end")}
        });
    }
);
""";

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

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

className

Instance của
className tìm
được trên heap

Sử dụng
instance để
gọi function

Python bindings

- Kết quả:

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

- Như đã biết, **Dinesh@123\$** là password của user **dinesh** 😊

Python bindings

- Thêm vài ví dụ nữa cho vui 😊
- Nhớ đoạn detect root code hông? Chúng ta sẽ hook và sửa nó, let's Frida!
- Đoạn code đó nằm trong `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!!");  
}
```

- Như hình, nếu `doesSuperuserApkExist(String paramString)` trả về true, thì root detected. Mặc định thì chúng ta đã vượt qua đoạn check này rồi, nhưng mà mình hem thích, mình muốn sửa nó lại để nó detect mình cho vui 😊 => make it true!

Python bindings

- Hàm này trả về bool val, hijack giá trị trả về là xong

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

- Như nào? Lần này sử dụng `Java.use()`
- Trích từ document frida gốc:

- `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

- Ngắn gọn là `Java.use()` cho phép chúng ta ghi đè 1 method trong class.
- Đầu tiên chọn class có method đó:

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

- Sau đó thì ghi đè sử dụng implementation:

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

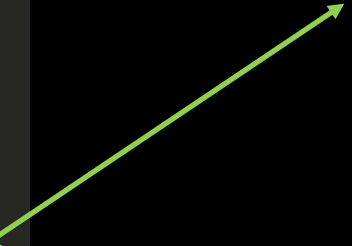
- Done

Python bindings

- Code như sau:

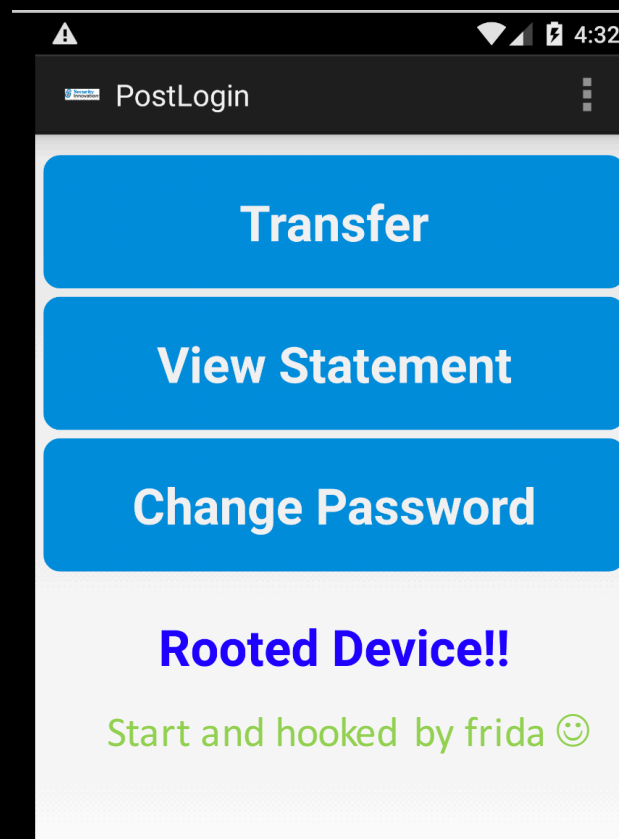
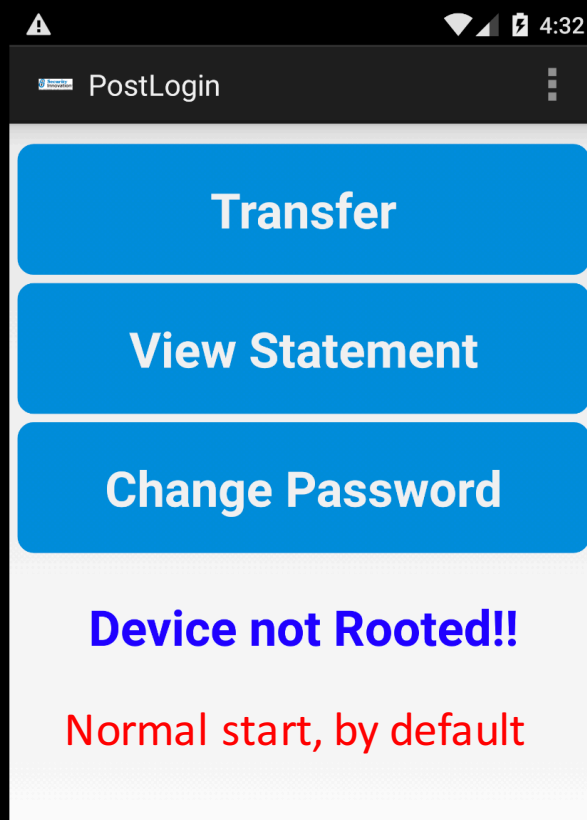
```
1  import frida
2  import time
3
4  device = frida.get_usb_device() # get device information
5  pid = device.spawn("com.android.insecurebankv2") # spawn app
6  device.resume(pid) # resumes it pid
7
8  time.sleep(1) # sleep 1 to avoid crash (sometime)
9
10 session=device.attach(pid)
11
12 hook_script="""
13 Java.perform
14 (
15     function ()
16     {
17         console.log("Inside the hook_script");
18         class_PostLogin = Java.use('com.android.insecurebankv2.PostLogin');
19
20         class_PostLogin.doesSuperuserApkExist.implementation = function (x)
21         {
22             return true;
23         };
24     }
25 );
26 """
27
28 script=session.create_script(hook_script)
29 script.load()
30
31 input('...?') # prevent terminate
```

Use implementation to override
function `doesSuperuserApkExist` content



Python bindings

- Vào điện thoại ảo, đăng nhập, bị phát hiện root roài (>_<!)



Python bindings

- Một ví dụ khác nhé, đa số các app đều có client-side filter, giờ sử dụng frida để bypass thử xem
- Nhìn đoạn code này trong `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_
    if (ChangePassword.access$100(this.this$0).matcher(this.this$0.changePassword_
```

Python bindings

- Để ý đoạn regex:

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

Đoạn này kiểm tra độ phức tạp của password, nghĩa là nếu chúng ta đổi password dễ quá thì sẽ không được

- Mình là người đơn giản nên mình đổi thử password thành “1234”:

ChangePassword

dinesh

Change Password

Entered password is not complex enough.

Fail, vì pw phải từ 6->20 kí tự
“1234” chỉ có 4 thôi

Python bindings

- Chúng ta sẽ sử dụng frida để bypass. Lần này không inject vào app class nữa, vì regex method là từ java package, không phải từ code của dev.

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

from

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

- Hook:

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

- Khác với lần trước, ta không thể xài `regex_Pattern_hook.compile.implementation` bởi vì trong Pattern package có nhiều loại compile methods (`compile(String x)`; `compile(String x,int y)`; etc...), do đó frida sẽ bị rối!
- Chúng ta phải nói với frida chính xác method nào để hook, ở đây là: `compile(String x)`
- Sử dụng overload:

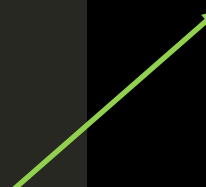
```
regex_Pattern_hook.compile.overload("java.lang.String").implementation
```

Python bindings

- Code:

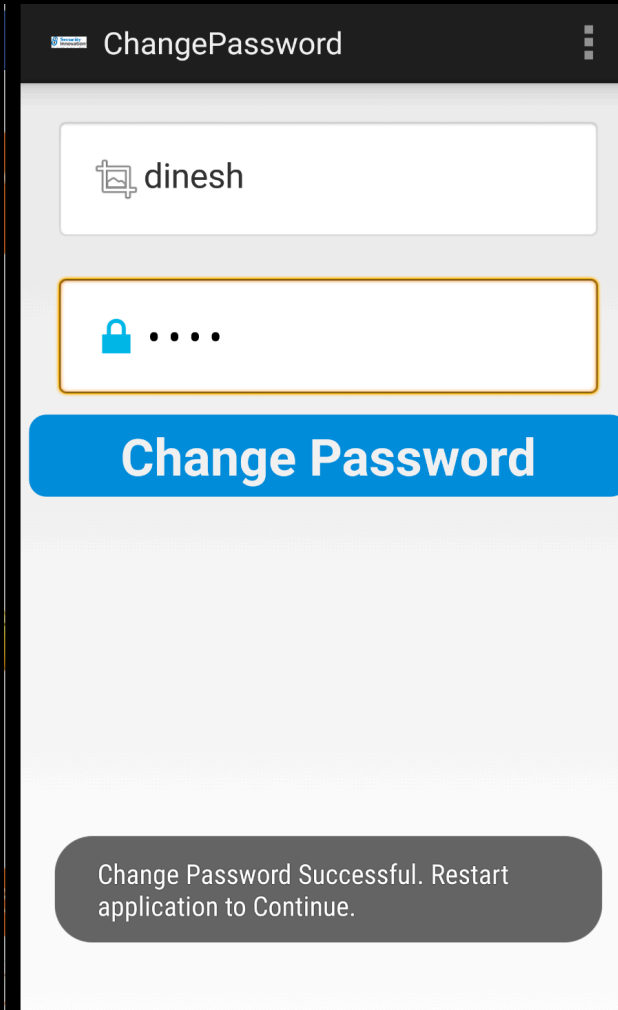
```
1  import frida
2  import time
3
4  device = frida.get_usb_device() # get device information
5  pid = device.spawn("com.android.insecurebankv2") # spawn app
6  device.resume(pid) # resumes it pid
7  time.sleep(1) # sleep 1 to avoid crash (sometime)
8  session=device.attach(pid)
9
10 hook_script="""
11
12 Java.perform
13 (
14     function ()
15     {
16         console.log("Inside the hook script");
17         regex_pattern_hook = Java.use('java.util.regex.Pattern');
18         regex_pattern_hook.compile.overload("java.lang.String").implementation = function (x)
19         {
20             return this.compile(".*");
21         }
22     }
23 );
24 """
25
26 script=session.create_script(hook_script)
27 script.load()
28
29 input('...?') # prevent terminate
```

Sửa regex thành ".*", nghĩa là cho phép mọi trường hợp



Python bindings

- *Trong một số trường hợp, sau khi hook app sẽ bị treo (chắc do máy ver cũ), thì reload script 1 lần nữa là app sẽ chạy bình thường (hook đúng nha, chứ hook sai thì fail chắc rồi)*
- Thử nào:



Python bindings

- Đăng nhập và xem trong logcat để confirm:

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

- Còn nhiều frida api hữu dụng khác khi xài với python: **send**, **recv** and **rpc**

Full docs:

<https://www.frida.re/docs/javascript-api/>