Frida

https://github.com/frida/frida

Frida环境

pyenv

python全版本随机切换，这里提供macOS上的配置方法

```
brew update
brew install pyenv
echo -e 'if command -v pyenv 1>/dev/null 2>&1; then\n  eval "$(pyenv init -
)"\nfi' >> ~/.bash_profile
```

```
下载一个3.8.2，下载真的很慢，要慢慢等
pyenv install 3.8.2

pyenv versions
sakura@sakuradeMacBook-Pro:~$ pyenv versions
  system
* 3.8.2 (set by /Users/sakura/.python-version)
切换到我们装的
pyenv local 3.8.2
python -V
pip -V
原本系统自带的
python local system
python -V
```

另外当你需要临时禁用pyenv的时候

```
    8.0_141.jdk/Contents/Home
17  export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk-14.
    jdk/Contents/Home
18  # export PATH="$HOME/.cargo/bin:$PATH"
19
20  if command -v pyenv 1>/dev/null 2>&1; then
21    eval "$(pyenv init -)"
22  fi
23
```

把这个注释了然后另开终端就好了。

关于卸载某个python版本

```
Uninstalling Python Versions
As time goes on, you will accumulate Python versions in your $(pyenv
root)/versions directory.

To remove old Python versions, pyenv uninstall command to automate the removal
process.

Alternatively, simply rm -rf the directory of the version you want to remove.
You can find the directory of a particular Python version with the pyenv
prefix command, e.g. pyenv prefix 2.6.8.
```

frida安装

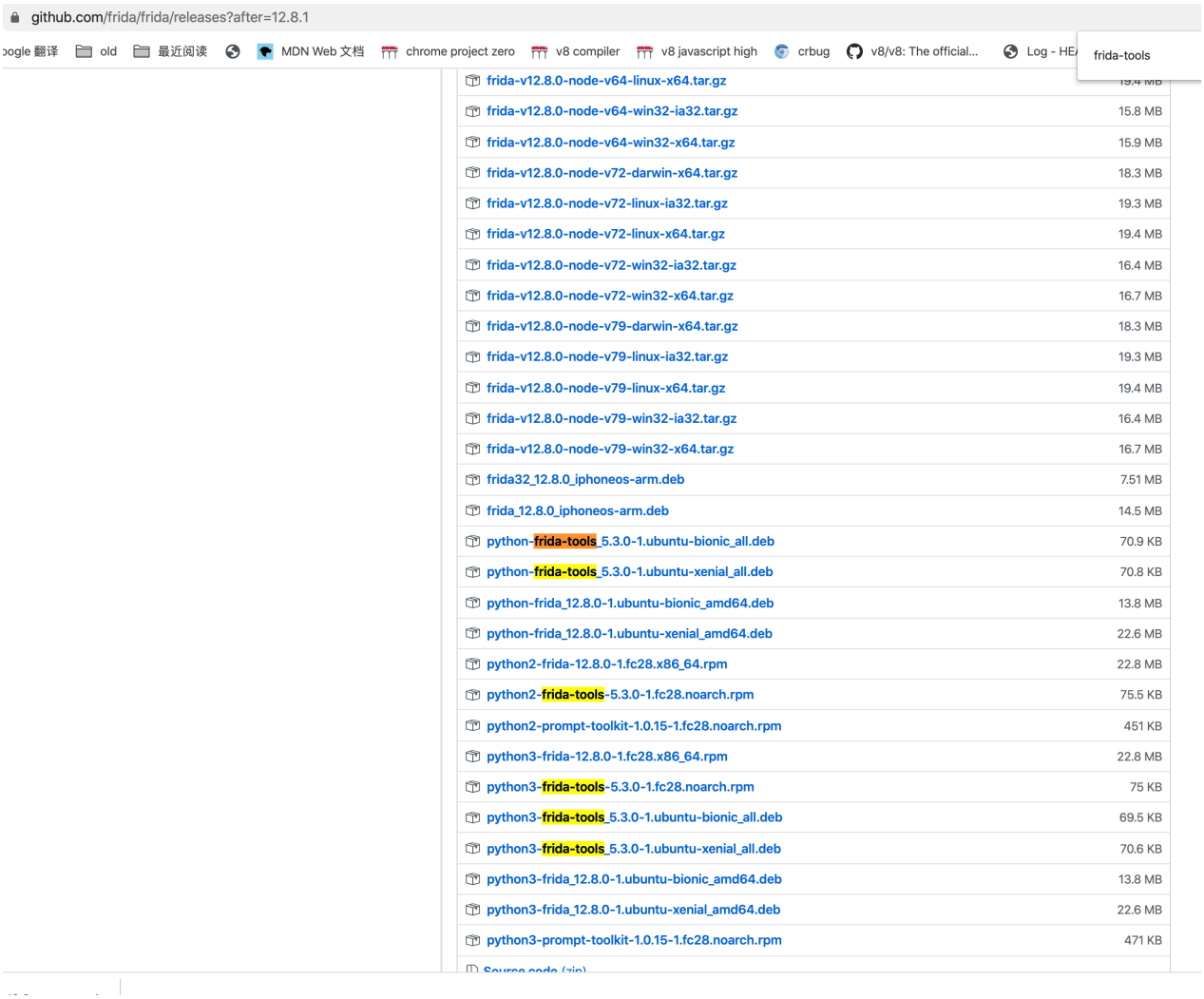
如果直接按下述安装则会直接安装frida和frida-tools的最新版本。

```
pip install frida-tools
frida --version
frida-ps --version
```

我们也可以自由安装旧版本的frida，例如12.8.0

```
pyenv install 3.7.7
pyenv local 3.7.7
pip install frida==12.8.0
pip install frida-tools==5.3.0
```

老版本frida和对应关系 对应关系很好找



安装objection

```
pyenv local 3.8.2
pip install objection
objection -h
```

```
pyenv local 3.7.7
pip install objection==1.8.4
objection -h
```

frida使用

下载frida-server并解压，在这里下载[frida-server-12.8.0]

```
sakura@sakuradeMacBook-Pro:~$ adb push
/Users/sakura/Desktop/lab/alpha/tools/android/frida-server-12.8.0-android-
arm64 /data/local/tmp
```

先adb shell，然后切换到root权限,把之前push进来的frida server改个名字叫fs 然后运行frida

```
chmod +x fs
./fs
```

如果要监听端口，就

```
./fs -l 0.0.0.0:8888
```

frida开发环境搭建

1. 安装

```
git clone https://github.com/oleavr/frida-agent-example.git
cd frida-agent-example/
npm install
```

2. 使用vscode打开此工程，在agent文件夹下编写js，会有智能提示。
3. `npm run watch`会监控代码修改自动编译生成js文件
4. python脚本或者cli加载_agent.js `frida -U -f com.example.android --no-pause -l _agent.js`

下面是测试脚本

`s1.js`

```
function main() {
    Java.perform(function x() {
        console.log("sakura")
    })
}
setImmediate(main)
```

`loader.py`

```
import time
import frida

device8 = frida.get_device_manager().add_remote_device("192.168.0.9:8888")
pid = device8.spawn("com.android.settings")
device8.resume(pid)
time.sleep(1)
session = device8.attach(pid)
with open("si.js") as f:
    script = session.create_script(f.read())
script.load()
input() #等待输入
```

解释一下，这个脚本就是先通过`frida.get_device_manager().add_remote_device`来找到device，然后spawn方式启动settings，然后attach到上面，并执行frida脚本。

FRIDA基础

frida查看当前存在的进程

`frida-ps -U`查看通过usb连接的android手机上的进程。

```
sakura@sakuradeMacBook-Pro:~$ frida-ps --help
Usage: frida-ps [options]

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -D ID, --device=ID    connect to device with the given ID
  -U, --usb             connect to USB device
  -R, --remote          connect to remote frida-server
  -H HOST, --host=HOST  connect to remote frida-server on HOST
  -a, --applications    list only applications
  -i, --installed       include all installed applications
```

```
sakura@sakuradeMacBook-Pro:~$ frida-ps -U
  PID  Name
-----  ------------------------------------------------
 3640  ATFWD-daemon
  707  adbd
  728  adsprpcd
26041  android.hardware.audio@2.0-service
  741  android.hardware.biometrics.fingerprint@
```

通过grep过滤就可以找到我们想要的包名。

frida打印参数和修改返回值

```
package myapplication.example.com.frida_demo;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends AppCompatActivity {

    private String total = "@@@#####@@@";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        while (true){

            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            fun(50,30);
            Log.d("sakura.string" , fun("LoWeRcAsE Me!!!!!!!!!"));
        }
    }

    void fun(int x , int y ){
        Log.d("sakura.Sum" , String.valueOf(x+y));
    }

    String fun(String x){
        total +=x;
        return x.toLowerCase();
    }

    String secret(){
        return total;
    }
}
```

```
function main() {
    console.log("Enter the Script!");
    Java.perform(function x() {
        console.log("Inside Java perform");
        var MainActivity =
Java.use("myapplication.example.com.frida_demo.MainActivity");
        // 重载找到指定的函数
        MainActivity.fun.overload('java.lang.String').implementation =
function (str) {
            //打印参数
            console.log("original call : str:" + str);
            //修改结果
            var ret_value = "sakura";
            return ret_value;
        };
    })
}
setImmediate(main);
```

```
sakura@sakuradeMacBook-Pro:~$ frida-ps -U | grep frida
8738  frida-helper-32
8897  myapplication.example.com.frida_demo

// -f是通过spawn，也就是重启apk注入js
sakura@sakuradeMacBook-Pro:~$ frida -U -f myapplication.example.com.frida_demo
-l frida_demo.js
...
original call : str:LoWeRcAsE Me!!!!!!!!!
12-21 04:46:49.875 9594-9594/myapplication.example.com.frida_demo
D/sakura.string: sakura
```

frida寻找instance，主动调用。

```
function main() {
    console.log("Enter the Script!");
    Java.perform(function x() {
        console.log("Inside Java perform");
        var MainActivity =
Java.use("myapplication.example.com.frida_demo.MainActivity");
        //overload 选择被重载的对象
        MainActivity.fun.overload('java.lang.String').implementation =
function (str) {
            //打印参数
            console.log("original call : str:" + str);
            //修改结果
            var ret_value = "sakura";
            return ret_value;
        };
        // 寻找类型为classname的实例
        Java.choose("myapplication.example.com.frida_demo.MainActivity", {
            onMatch: function (x) {
                console.log("find instance :" + x);
                console.log("result of secret func:" + x.secret());
            },
            onComplete: function () {
                console.log("end");
            }
        });
    });
}
setImmediate(main);
```

frida rpc

```
function callFun() {
    Java.perform(function fn() {
```

```
        console.log("begin");
        Java.choose("myapplication.example.com.frida_demo.MainActivity", {
            onMatch: function (x) {
                console.log("find instance :" + x);
                console.log("result of fun(string) func:" +
x.fun(Java.use("java.lang.String").$new("sakura")));
            },
            onComplete: function () {
                console.log("end");
            }
        })
    })
}
rpc.exports = {
    callfun: callFun
};
```

```
import time
import frida

device = frida.get_usb_device()
pid = device.spawn(["myapplication.example.com.frida_demo"])
device.resume(pid)
time.sleep(1)
session = device.attach(pid)
with open("frida_demo_rpc_call.js") as f:
    script = session.create_script(f.read())

def my_message_handler(message, payload):
    print(message)
    print(payload)

script.on("message", my_message_handler)
script.load()

script.exports.callfun()
```

```
sakura@sakuradeMacBook-Pro:~/gitsource/frida-agent-example/agent$ python
frida_demo_rpc_loader.py
begin
find instance :myapplication.example.com.frida_demo.MainActivity@1d4b09d
result of fun(string):sakura
end
```

frida动态修改

即将手机上的app的内容发送到PC上的frida python程序，然后处理后返回给app，然后app再做后续的流程，核心是理解`send/recv`函数

```
<TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="please input username and password"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />


    <EditText
        android:id="@+id/editText"
        android:layout_width="fill_parent"
        android:layout_height="40dp"
        android:hint="username"
        android:maxLength="20"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="1.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.095" />

    <EditText
        android:id="@+id/editText2"
        android:layout_width="fill_parent"
        android:layout_height="40dp"
        android:hint="password"
        android:maxLength="20"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.239" />

    <Button
        android:id="@+id/button"
        android:layout_width="100dp"
        android:layout_height="35dp"
        android:layout_gravity="right|center_horizontal"
        android:text="提交"
        android:visibility="visible"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.745" />
```

```
public class MainActivity extends AppCompatActivity {

    EditText username_et;
    EditText password_et;
    TextView message_tv;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        password_et = (EditText) this.findViewById(R.id.editText2);
        username_et = (EditText) this.findViewById(R.id.editText);
        message_tv = ((TextView) findViewById(R.id.textView));

        this.findViewById(R.id.button).setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {

                if (username_et.getText().toString().compareTo("admin") == 0)
{
                    message_tv.setText("You cannot login as admin");
                    return;
                }
                //hook target
                message_tv.setText("Sending to the server :" +
Base64.encodeToString((username_et.getText().toString() + ":" +
password_et.getText().toString()).getBytes(), Base64.DEFAULT));

            }
        });

    }
}
```

先分析问题，我的最终目标是让message_tv.setText可以"发送"username为admin的base64字符串。那肯定是hook TextView.setText这个函数。

```
console.log("Script loaded successfully ");
Java.perform(function () {
    var tv_class = Java.use("android.widget.TextView");
    tv_class.setText.overload("java.lang.CharSequence").implementation =
function (x) {
        var string_to_send = x.toString();
        var string_to_recv;
        send(string_to_send); // send data to python code
        recv(function (received_json_object) {
```

```
            string_to_recv = received_json_object.my_data
            console.log("string_to_recv: " + string_to_recv);
        }).wait(); //block execution till the message is received
        var my_string = Java.use("java.lang.String").$new(string_to_recv);
        this.setText(my_string);
    }
});
```

```
import time
import frida
import base64

def my_message_handler(message, payload):
    print(message)
    print(payload)
    if message["type"] == "send":
        print(message["payload"])
        data = message["payload"].split(":")[1].strip()
        print( 'message:', message)
        #data = data.decode("base64")
        #data = data
        data = str(base64.b64decode(data))
        print( 'data:',data)
        user, pw = data.split(":")
        print( 'pw:',pw)
        #data = ("admin" + ":" + pw).encode("base64")
        data = str(base64.b64encode(("admin" + ":" + pw).encode()))
        print( "encoded data:", data)
        script.post({"my_data": data})  # send JSON object
        print( "Modified data sent")

device = frida.get_usb_device()
pid = device.spawn(["myapplication.example.com.frida_demo"])
device.resume(pid)
time.sleep(1)
session = device.attach(pid)
with open("frida_demo2.js") as f:
    script = session.create_script(f.read())
script.on("message", my_message_handler)
script.load()
input()
```

```
sakura@sakuradeMacBook-Pro:~/gitsource/frida-agent-example/agent$ python
frida_demo_rpc_loader2.py
Script loaded successfully
{'type': 'send', 'payload': 'Sending to the server :c2FrdXJhOjEyMzQ1Ng==\n'}
None
Sending to the server :c2FrdXJhOjEyMzQ1Ng==

message: {'type': 'send', 'payload': 'Sending to the server
:c2FrdXJhOjEyMzQ1Ng==\n'}
data: b'sakura:123456'
pw: 123456'
encoded data: b'YWRtaW46MTIzNDU2Jw=='
Modified data sent
string_to_recv: b'YWRtaW46MTIzNDU2Jw=='
```

参考链接：https://github.com/Mind0xP/Frida-Python-Binding

API List

- `Java.choose(className: string, callbacks: Java.ChooseCallbacks): void` 通过扫描Java VM的堆来枚举className类的live instance。
- `Java.use(className: string): Java.Wrapper<{}>` 动态为className生成JavaScript Wrapper，可以通过调用`$new()`来调用构造函数来实例化对象。 在实例上调用`$dispose()`以对其进行显式清理，或者等待JavaScript对象被gc。
- `Java.perform(fn: () => void): void` Function to run while attached to the VM. Ensures that the current thread is attached to the VM and calls fn. (This isn't necessary in callbacks from Java.) Will defer calling fn if the app's class loader is not available yet. Use Java.performNow() if access to the app's classes is not needed.
- `send(message: any, data?: ArrayBuffer | number[]): void` 任何JSON可序列化的值。将JSON序列化后的message发送到您的基于Frida的应用程序，并包含(可选)一些原始二进制数据。 The latter is useful if you e.g. dumped some memory using NativePointer#readByteArray().
- `recv(callback: MessageCallback): MessageRecvOperation` Requests callback to be called on the next message received from your Frida-based application. This will only give you one message, so you need to call recv() again to receive the next one.
- `wait(): void` 堵塞，直到message已经receive并且callback已经执行完毕并返回

Frida动静态结合分析

Objection

- 参考这篇文章 实用FRIDA进阶：内存漫游、hook anywhere、抓包
- objection https://pypi.org/project/objection/

objection启动并注入内存

`objection -d -g package_name explore`

```
sakura@sakuradeMacBook-Pro:~$ objection -d -g com.android.settings explore
[debug] Agent path is: /Users/sakura/.pyenv/versions/3.7.7/lib/python3.7/site-
packages/objection/agent.js
[debug] Injecting agent...
Using USB device `Google Pixel`
[debug] Attempting to attach to process: `com.android.settings`
[debug] Process attached!
Agent injected and responds ok!


     _   _         _   _
 ___| |_|_|___ ___| |_|_|___ ___
| . | . | | -_|  _|  _| | . |   |
|___|___| |___|___|_| |_|___|_|_|
      |___|(object)inject(ion) v1.8.4

     Runtime Mobile Exploration
        by: @leonjza from @sensepost

[tab] for command suggestions
com.android.settings on (google: 8.1.0) [usb] #
```

objection memory

查看内存中加载的module `memory list modules`

```
com.android.settings on (google: 8.1.0) [usb] # memory list modules
Save the output by adding `--json modules.json` to this command
Name                                                Base          Size
       Path
-------------------------------------------------  ------------  ---------------
-----  ----------------------------------------------------------------
app_process64                                       0x64ce143000  32768 (32.0
KiB)      /system/bin/app_process64
libandroid_runtime.so                               0x7a90bc3000  1990656 (1.9
MiB)     /system/lib64/libandroid_runtime.so
libbinder.so                                        0x7a9379f000  557056 (544.0
KiB)    /system/lib64/libbinder.so
```

查看库的导出函数 `memory list exports libssl.so`

```
com.android.settings on (google: 8.1.0) [usb] # memory list exports libssl.so
Save the output by adding `--json exports.json` to this command
Type      Name                                                  Address
--------  ----------------------------------------------------  ----------
function  SSL_use_certificate_ASN1                              0x7c8ff006f8
function  SSL_CTX_set_dos_protection_cb                         0x7c8ff077b8
function  SSL_SESSION_set_ex_data                               0x7c8ff098f4
function  SSL_CTX_set_session_psk_dhe_timeout                   0x7c8ff0a754
function  SSL_CTX_sess_accept                                   0x7c8ff063b8
function  SSL_select_next_proto                                 0x7c8ff06a74
```

dump内存空间

- `memory dump all 文件名`
- `memory dump from_base 起始地址 字节数 文件名`

搜索内存空间

```
Usage: memory search "<pattern eg: 41 41 41 ?? 41>" (--string) (--offsets-only)
```

objection android

内存堆搜索实例 `android heap search instances 类名`

在堆上搜索类的实例

```
sakura@sakuradeMacBook-Pro:~$ objection -g
myapplication.example.com.frida_demo explore
Using USB device `Google Pixel`
Agent injected and responds ok!

[usb] # android heap search instances myapplication.example.com.frida_demo
.MainActivity
Class instance enumeration complete for
myapplication.example.com.frida_demo.MainActivity
Handle    Class                                               toString()
--------  --------------------------------------------------  -----------------
----------------------------------------
0x2102    myapplication.example.com.frida_demo.MainActivity
myapplication.example.com.frida_demo.MainActivity@5b1b0af
```

调用实例的方法 `android heap execute 实例ID 实例方法`

查看当前可用的activity或者service `android hooking list activities/services`

直接启动activity或者服务 `android intent launch_activity/launch_service activity/服务`

`android intent launch_activity com.android.settings.DisplaySettings` 这个命令比较有趣的是用在如果有些设计的不好，可能就直接绕过了密码锁屏等直接进去。

```
com.android.settings on (google: 8.1.0) [usb] # android hooking list services
com.android.settings.SettingsDumpService
com.android.settings.TetherService
com.android.settings.bluetooth.BluetoothPairingService
```

列出内存中所有的类 `android hooking list classes`

在内存中所有已加载的类中搜索包含特定关键词的类。 `android hooking search classes display`

```
com.android.settings on (google: 8.1.0) [usb] # android hooking search classes
display
[Landroid.icu.text.DisplayContext$Type;
[Landroid.icu.text.DisplayContext;
[Landroid.view.Display$Mode;
android.hardware.display.DisplayManager
android.hardware.display.DisplayManager$DisplayListener
android.hardware.display.DisplayManagerGlobal
```

内存中搜索指定类的所有方法 `android hooking list class_methods 类名`

```
com.android.settings on (google: 8.1.0) [usb] # android hooking list
class_methods java.nio.charset.Charset
private static java.nio.charset.Charset
java.nio.charset.Charset.lookup(java.lang.String)
private static java.nio.charset.Charset
java.nio.charset.Charset.lookup2(java.lang.String)
private static java.nio.charset.Charset
java.nio.charset.Charset.lookupViaProviders(java.lang.String)
```

在内存中所有已加载的类的方法中搜索包含特定关键词的方法 `android hooking search methods display`

知道名字开始在内存里搜就很有用

```
com.android.settings on (google: 8.1.0) [usb] # android hooking search methods
display
Warning, searching all classes may take some time and in some cases, crash the
target application.
Continue? [y/N]: y
Found 5529 classes, searching methods (this may take some time)...
android.app.ActionBar.getDisplayOptions
android.app.ActionBar.setDefaultDisplayHomeAsUpEnabled
android.app.ActionBar.setDisplayHomeAsUpEnabled
```

hook类的方法（hook类里的所有方法/具体某个方法）

- `android hooking watch class 类名` 这样就可以hook这个类里面的所有方法，每次调用都会

被log出来。

- `android hooking watch class 类名 --dump-args --dump-backtrace --dump-return` 在上面的基础上，额外dump参数，栈回溯，返回值

```
android hooking watch class xxx.MainActivity --dump-args --dump-backtrace --
dump-return
```

- `android hooking watch class_method 方法名`

```
//可以直接hook到所有重载
android hooking watch class_method xxx.MainActivity.fun --dump-args --dump-
backtrace --dump-return
```

grep trick和文件保存

objection log默认是不能用grep过滤的，但是可以通过`objection run xxx | grep yyy的`方式，从终端通过管道来过滤。 用法如下

```
sakura@sakuradeMacBook-Pro:~$ objection -g com.android.settings run memory
list modules | grep libc
Warning: Output is not to a terminal (fd=1).
libcutils.so                                    0x7a94a1c000  81920 (80.0
KiB)      /system/lib64/libcutils.so
libc++.so                                       0x7a9114e000  983040 (960.0
KiB)    /system/lib64/libc++.so
libc.so                                         0x7a9249d000  892928 (872.0
KiB)    /system/lib64/libc.so
libcrypto.so                                    0x7a92283000  1155072 (1.1
MiB)     /system/lib64/libcrypto.so
```

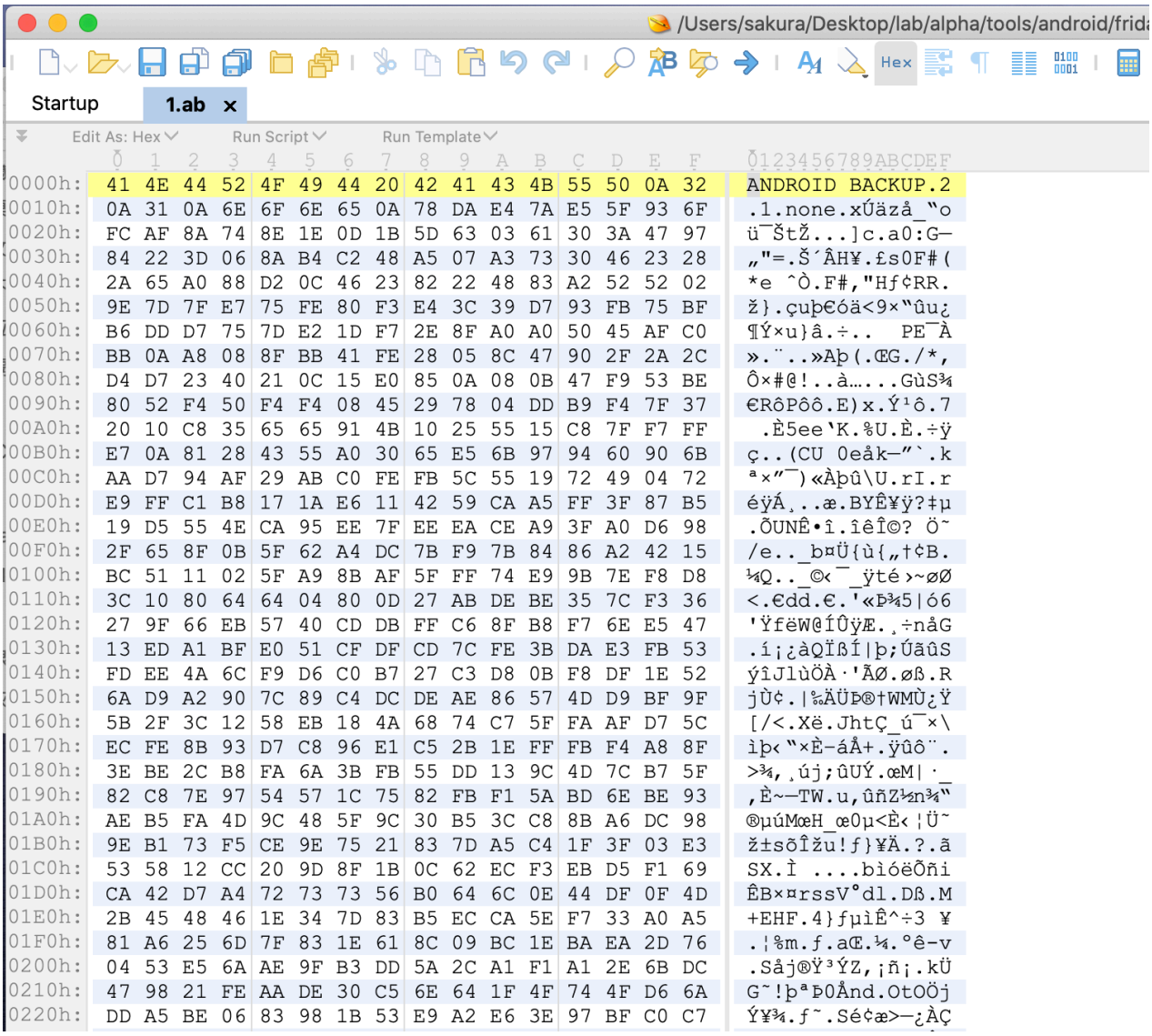
有的命令后面可以通过`--json logfile`来直接保存结果到文件里。 有的可以通过查看`.objection`文件里的输出log来查看结果。

```
sakura@sakuradeMacBook-Pro:~/.objection$ cat *log | grep -i display
android.hardware.display.DisplayManager
android.hardware.display.DisplayManager$DisplayListener
android.hardware.display.DisplayManagerGlobal
```

案例学习

案例学习case1:《仿VX数据库原型取证逆向分析》

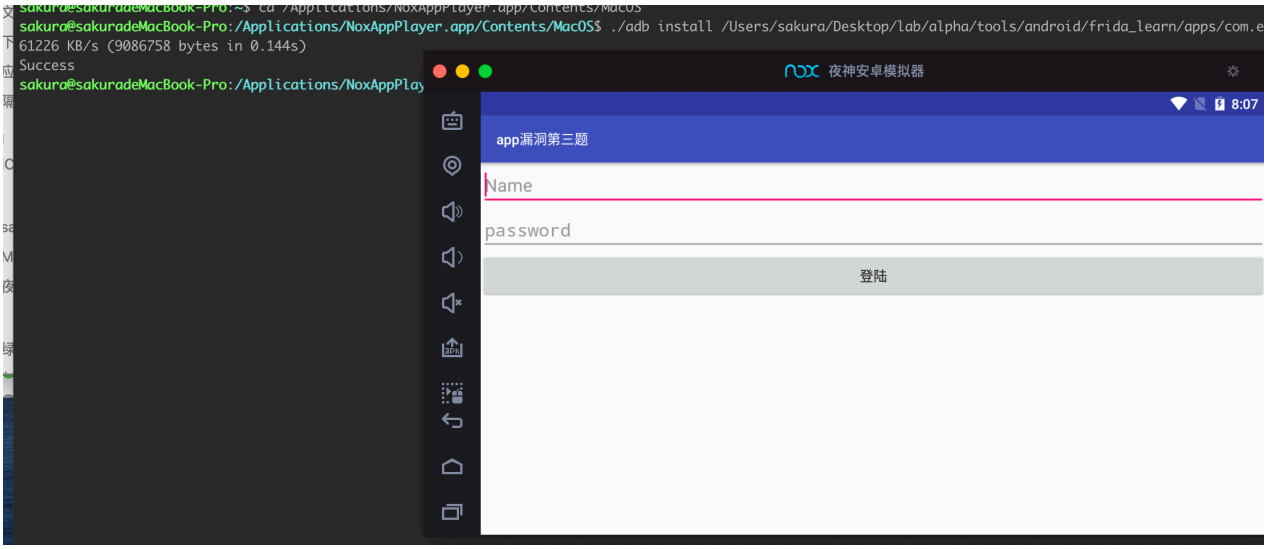
附件链接 android-backup-extractor工具链接

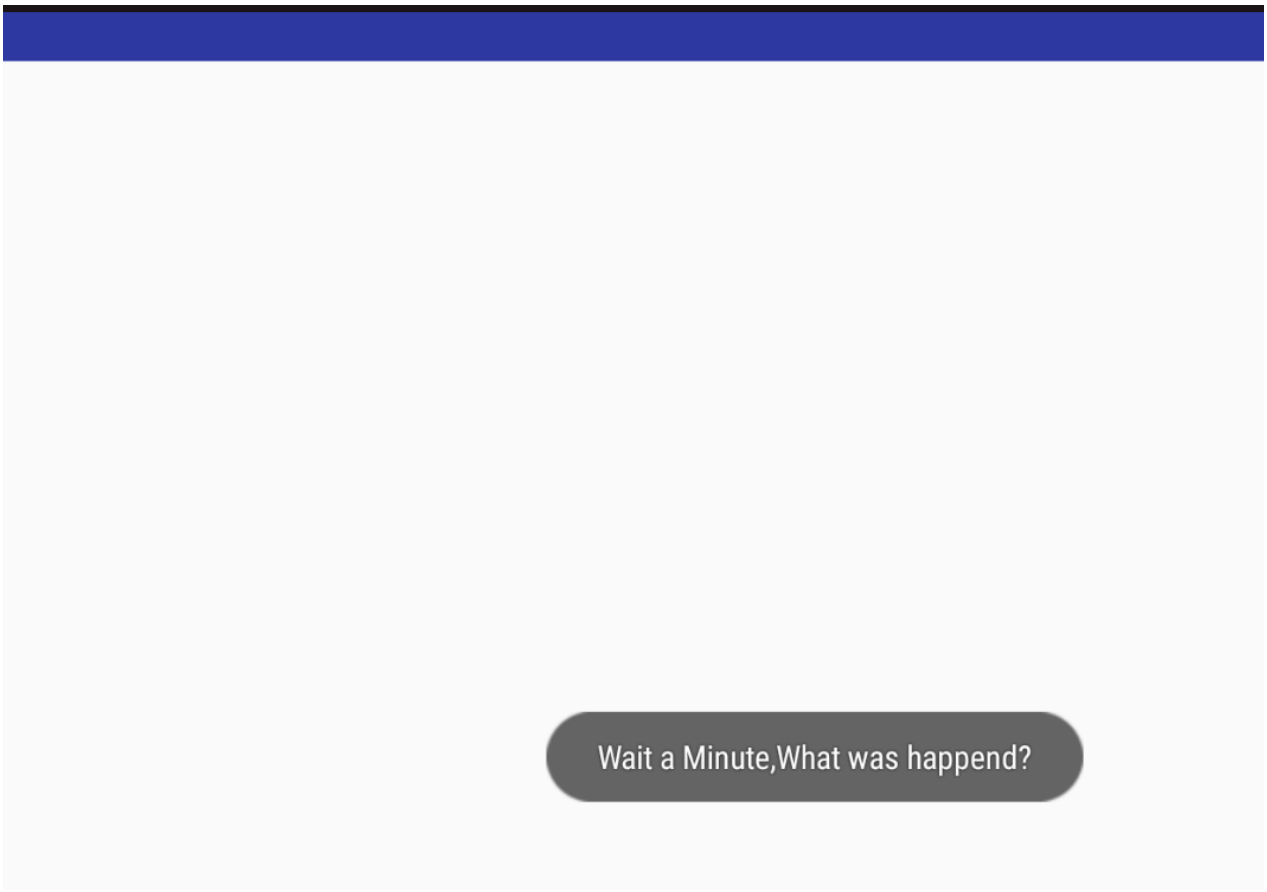



```
sakura@sakuradeMacBook-Pro:~/Desktop/lab/alpha/tools/android/frida_learn$ java
-version
java version "1.8.0_141"

sakura@sakuradeMacBook-Pro:~/Desktop/lab/alpha/tools/android/frida_learn$ java
-jar abe-all.jar unpack 1.ab 1.tar
0% 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% 29% 30% 31% 32% 33% 34% 35% 36% 37% 38% 39% 40%
41% 42% 43% 44% 45% 46% 47% 48% 49% 50% 51% 52% 53% 54% 55% 56% 57% 58% 59%
60% 61% 62% 63% 64% 65% 66% 67% 68% 69% 70% 71% 72% 73% 74% 75% 76% 77% 78%
79% 80% 81% 82% 83% 84% 85% 86% 87% 88% 89% 90% 91% 92% 93% 94% 95% 96% 97%
98% 99% 100%
9097216 bytes written to 1.tar.

...
sakura@sakuradeMacBook-
Pro:~/Desktop/lab/alpha/tools/android/frida_learn/apps/com.example.yaphetshan.
tencentwelcome$ ls
Encryto.db _manifest  a          db
```

装个夜神模拟器玩

```
sakura@sakuradeMacBook-Pro:/Applications/NoxAppPlayer.app/Contents/MacOS$
./adb connect 127.0.0.1:62001
* daemon not running. starting it now on port 5037 *
adb E  5139 141210 usb_osx.cpp:138] Unable to create an interface plug-in
(e00002be)
* daemon started successfully *
connected to 127.0.0.1:62001
sakura@sakuradeMacBook-Pro:/Applications/NoxAppPlayer.app/Contents/MacOS$
./adb shell
dream2qltechn:/ # whoami
root
dream2qltechn:/ # uname -a
Linux localhost 4.0.9+ #222 SMP PREEMPT Sat Mar 14 18:24:36 HKT 2020 i686
```





肯定还是先定位目标字符串`Wait a Minute,What was happend?` jadx搜索字符串


```
31      public void onCreate(Bundle bundle) {
32          super.onCreate(bundle);
33          setContentView((int) R.layout.activity_main);
34          this.c = (Button) findViewById(R.id.add_data);
35          this.c.setOnClickListener(this);
37          SharedPreferences.Editor edit = getSharedPreferences("test"
38          edit.putString("Is_Encroty", "1");
39          edit.putString("Encryto", "SqlCipher");
40          edit.putString("ver_sion", "3_4_0");
41          edit.apply();
42          a();
        }

45      private void a() {
46          SQLiteDatabase.loadLibs(this);
47          this.b = new a(this, "Demo.db", (SQLiteDatabase.CursorFacto
48          ContentValues contentValues = new ContentValues();
49          contentValues.put("name", "Stranger");
50          contentValues.put("password", 123456);
51          a aVar = new a();
52          String a2 = aVar.a(contentValues.getAsString("name"), conte
55          this.a = this.b.getWritableDatabase(aVar.a(a2 + aVar.b(a2,
56          this.a.insert("TencentMicrMsg", (String) null, contentValue
        }

61      public void onClick(View view) {
62          if (view == this.c) {
64              Intent intent = new Intent();
65              intent.putExtra("name", "name");
66              intent.putExtra("password", "pass");
67              intent.setClass(this, AnotherActivity.class);
68              startActivity(intent);
            }
        }
    }
```

重点在a()代码里，其实是根据明文的name和password，然后`aVar.a(a2 + aVar.b(a2, contentValues.getAsString("password"))).substring(0, 7)`再做一遍复杂的计算并截取7位当做密码，传入getWritableDatabase去解密demo.db数据库。

所以我们hook一下getWritableDatabase即可。

```
frida-ps -U
...
5662  com.example.yaphetshan.tencentwelcome


objection -d -g com.example.yaphetshan.tencentwelcome explore
```

看一下源码

```
package net.sqlcipher.database;
...
public abstract class SQLiteOpenHelper {
    ...
    public synchronized SQLiteDatabase getWritableDatabase(char[] cArr) {
```

也可以objection search一下这个method

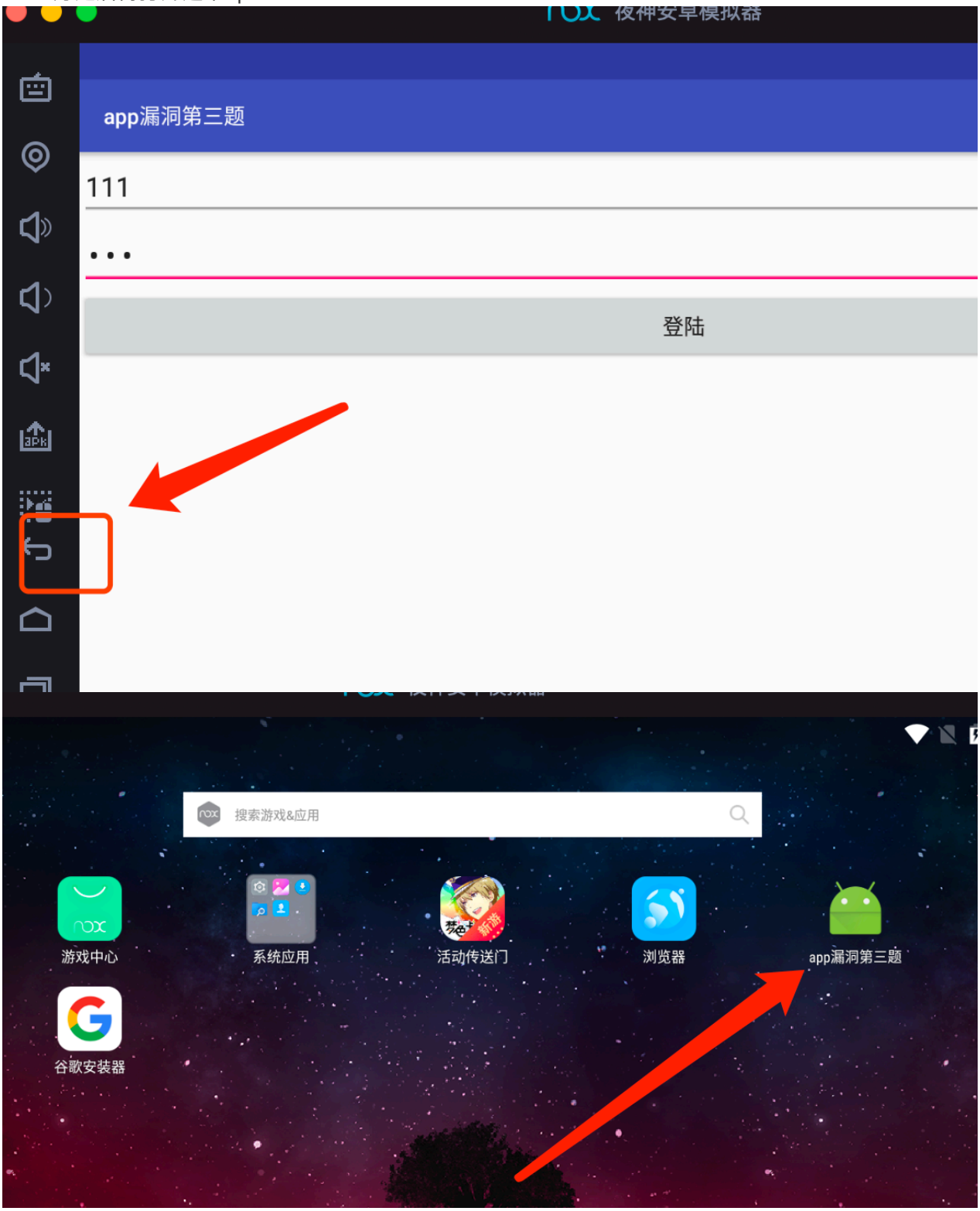
```
...mple.yaphetshan.tencentwelcome on (samsung: 7.1.2) [usb] # android hooking
search methods getWritableDatabase
Warning, searching all classes may take some time and in some cases, crash the
target application.
Continue? [y/N]: y
Found 4650 classes, searching methods (this may take some time)...

android.database.sqlite.SQLiteOpenHelper.getWritableDatabase
...
net.sqlcipher.database.SQLiteOpenHelper.getWritableDatabase
```

hook一下这个method

```
[usb] # android hooking watch class_method
net.sqlcipher.database.SQLiteOpenHelper.getWritableDatabase --dump-args --
dump-backtrace --dump-return
- [incoming message] ------------------
{
  "payload": "Attempting to watch class
\u001b[32mnet.sqlcipher.database.SQLiteOpenHelper\u001b[39m and method
\u001b[32mgetWritableDatabase\u001b[39m.",
  "type": "send"
}
- [./incoming message] ----------------
(agent) Attempting to watch class net.sqlcipher.database.SQLiteOpenHelper and
method getWritableDatabase.
- [incoming message] ------------------
{
  "payload": "Hooking
\u001b[32mnet.sqlcipher.database.SQLiteOpenHelper\u001b[39m.\u001b[92mgetWrita
bleDatabase\u001b[39m(\u001b[31mjava.lang.String\u001b[39m)",
  "type": "send"
}
- [./incoming message] ----------------
(agent) Hooking
net.sqlcipher.database.SQLiteOpenHelper.getWritableDatabase(java.lang.String)
- [incoming message] ------------------
{
```

```
  "payload": "Hooking
\u001b[32mnet.sqlcipher.database.SQLiteOpenHelper\u001b[39m.\u001b[92mgetWrita
bleDatabase\u001b[39m(\u001b[31m[C\u001b[39m)",
  "type": "send"
}
- [./incoming message] ----------------
(agent) Hooking
net.sqlcipher.database.SQLiteOpenHelper.getWritableDatabase([C)
- [incoming message] ------------------
{
  "payload": "Registering job \u001b[94mjytq1qeyllq\u001b[39m. Type:
\u001b[92mwatch-method for:
net.sqlcipher.database.SQLiteOpenHelper.getWritableDatabase\u001b[39m",
  "type": "send"
}
- [./incoming message] ----------------
(agent) Registering job jytq1qeyllq. Type: watch-method for:
net.sqlcipher.database.SQLiteOpenHelper.getWritableDatabase
...mple.yaphetshan.tencentwelcome on (samsung: 7.1.2) [usb] #
```

hook好之后再打开这个apk




```
(agent) [1v488x28gcs] Called
net.sqlcipher.database.SQLiteOpenHelper.getWritableDatabase(java.lang.String)
...
(agent) [1v488x28gcs] Backtrace:
  net.sqlcipher.database.SQLiteOpenHelper.getWritableDatabase(Native Method)
  com.example.yaphetshan.tencentwelcome.MainActivity.a(MainActivity.java:55)

com.example.yaphetshan.tencentwelcome.MainActivity.onCreate(MainActivity.java:
42)
```

```
  android.app.Activity.performCreate(Activity.java:6692)
...
(agent) [1v488x28gcs] Arguments
net.sqlcipher.database.SQLiteOpenHelper.getWritableDatabase(ae56f99)

...
...mple.yaphetshan.tencentwelcome on (samsung: 7.1.2) [usb] # jobs list
Job ID         Hooks  Type
-----------  -------  ------------------------------------------------------------
---------------------
1v488x28gcs        2  watch-method for:
net.sqlcipher.database.SQLiteOpenHelper.getWritableDatabase
```

找到参数`ae56f99`剩下的就是用这个密码去打开加密的db。

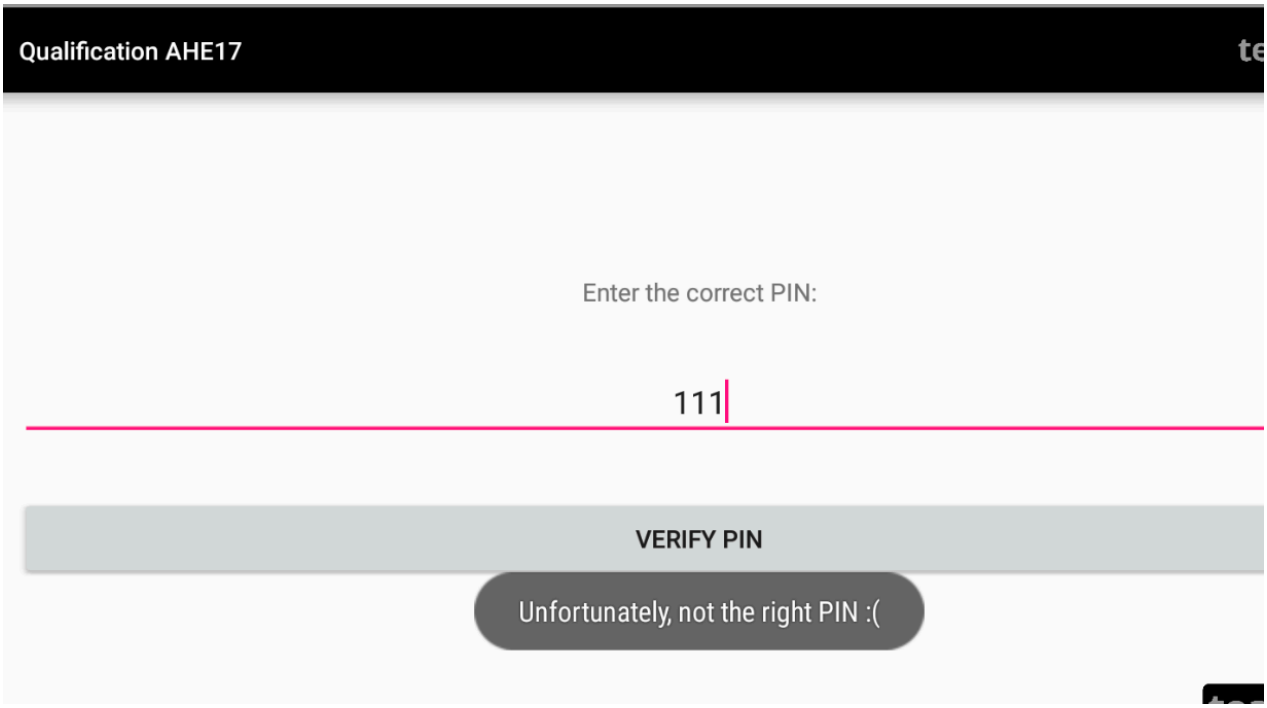
然后base64解密一下就好了。

还有一种策略是主动调用,基于数据流的主动调用分析是非常有意思的。 即自己去调用a函数以触发getWritableDatabase的数据库解密。 先寻找a所在类的实例，然后hook getWritableDatabase，最终主动调用a。 这里幸运的是a没有什么奇奇怪怪的参数需要我们传入，主动调用这种策略在循环注册等地方可能就会有需求8.

```
 [usb] # android heap search instances
com.example.yaphetshan.tencentwelcome.MainActivity
Class instance enumeration complete for
com.example.yaphetshan.tencentwelcome.MainActivity
Handle    Class                                                 toString()
--------  ----------------------------------------------------  ----------------
-----------------------------------------
0x20078a  com.example.yaphetshan.tencentwelcome.MainActivity
com.example.yaphetshan.tencentwelcome.MainActivity@1528f80

 [usb] # android hooking watch class_method
net.sqlcipher.database.SQLiteOpenHelper.getWritableDatabase --dump-args --
dump-backtrace --dump-return

[usb] # android heap execute 0x20078a a

(agent) [taupgwkum4h] Arguments
net.sqlcipher.database.SQLiteOpenHelper.getWritableDatabase(ae56f99)
```

案例学习case2:主动调用爆破密码

附件链接



因为直接找`Unfortunately,note the right PIN :(`找不到，可能是把字符串藏在什么资源文件里了。 review代码之后找到校验的核心函数，逻辑就是将input编码一下之后和密码比较，这肯定是什么不可逆的加密。

```
    public static boolean verifyPassword(Context context, String input) {
        if (input.length() != 4) {
            return false;
        }
        byte[] v = encodePassword(input);
        byte[] p = "09042ec2c2c08c4cbece042681caf1d13984f24a".getBytes();
        if (v.length != p.length) {
            return false;
        }
        for (int i = 0; i < v.length; i++) {
            if (v[i] != p[i]) {
                return false;
            }
        }
        return true;
    }
```

这里就爆破一下密码。

```
frida-ps -U | grep qualification
7660  org.teamsik.ahe17.qualification.easy

frida -U org.teamsik.ahe17.qualification.easy -l force.js
```

```
function main() {
    Java.perform(function x() {
        console.log("In Java perform")
        var verify = Java.use("org.teamsik.ahe17.qualification.Verifier")
        var stringClass = Java.use("java.lang.String")
        var p = stringClass.$new("09042ec2c2c08c4cbece042681caf1d13984f24a")
        var pSign = p.getBytes()
        // var pStr = stringClass.$new(pSign)
        // console.log(parseInt(pStr))
        for (var i = 999; i < 10000; i++){
            var v = stringClass.$new(String(i))
            var vSign = verify.encodePassword(v)
            if (parseInt(stringClass.$new(pSign)) ==
parseInt(stringClass.$new(vSign))) {
                console.log("yes: " + v)
                break
            }
            console.log("not :" + v)
        }
    })
}
setImmediate(main)
```

```
...
not :9080
not :9081
not :9082
yes: 9083
```

这里注意parseInt

Frida hook基础(一)

- 调用静态函数和调用非静态函数
- 设置(同名)成员变量
- 内部类，枚举类的函数并hook，trace原型1
- 查找接口，hook动态加载dex
- 枚举class，trace原型2
- objection不能切换classloader

Frida hook : 打印参数、返回值/设置返回值/主动调用

demo就不贴了，还是先定位登录失败点，然后搜索字符串。

```
public class LoginActivity extends AppCompatActivity {
    /* access modifiers changed from: private */
    public Context mContext;
```

```
    public void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        this.mContext = this;
        setContentView((int) R.layout.activity_login);
        final EditText editText = (EditText) findViewById(R.id.username);
        final EditText editText2 = (EditText) findViewById(R.id.password);
        ((Button) findViewById(R.id.login)).setOnClickListener(new
View.OnClickListener() {
            public void onClick(View view) {
                String obj = editText.getText().toString();
                String obj2 = editText2.getText().toString();
                if (TextUtils.isEmpty(obj) || TextUtils.isEmpty(obj2)) {
                    Toast.makeText(LoginActivity.this.mContext, "username or
password is empty.", 1).show();
                } else if (LoginActivity.a(obj, obj).equals(obj2)) {
                    LoginActivity.this.startActivity(new
Intent(LoginActivity.this.mContext, FridaActivity1.class));
                    LoginActivity.this.finishActivity(0);
                } else {
                    Toast.makeText(LoginActivity.this.mContext, "Login
failed.", 1).show();
                }
            }
        });
    }
```

`LoginActivity.a(obj, obj).equals(obj2)`分析之后可得obj2来自password，由从username得来的obj，经过a函数运算之后得到一个值，这两个值相等则登录成功。 所以这里关键是hook a函数的参数，最简脚本如下。

```
//打印参数、返回值
function Login(){
    Java.perform(function(){

 Java.use("com.example.androiddemo.Activity.LoginActivity").a.overload('java.l
ang.String', 'java.lang.String').implementation = function (str, str2){
            var result = this.a(str, str2);
            console.log("args0:"+str+" args1:"+str2+" result:"+result);
            return result;
        }
    })
}
setImmediate(Login)
```

观察输入和输出,这里也可以直接主动调用。

```
function login() {
    Java.perform(function () {
        console.log("start")
        var login = Java.use("com.example.androiddemo.Activity.LoginActivity")
        var result = login.a("1234","1234")
        console.log(result)
    })
}
setImmediate(login)
```

```
...
start
4e4feaea959d426155a480dc07ef92f4754ee93edbe56d993d74f131497e66fb
然后
adb shell input text
"4e4feaea959d426155a480dc07ef92f4754ee93edbe56d993d74f131497e66fb"
```

接下来是第一关

```
public abstract class BaseFridaActivity extends AppCompatActivity implements
View.OnClickListener {
    public Button mNextCheck;

    public void CheckSuccess() {
    }

    public abstract String getNextCheckTitle();

    public abstract void onCheck();

    /* access modifiers changed from: protected */
    public void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        setContentView((int) R.layout.activity_frida);
        this.mNextCheck = (Button) findViewById(R.id.next_check);
        this.mNextCheck.setOnClickListener(this);
        Button button = this.mNextCheck;
        button.setText(getNextCheckTitle() + ", 点击进入下一关");
    }

    public void onClick(View view) {
        onCheck();
    }

    public void CheckFailed() {
        Toast.makeText(this, "Check Failed!", 1).show();
    }
}
```

```
...

public class FridaActivity1 extends BaseFridaActivity {
    private static final char[] table = {'L', 'K', 'N', 'M', 'O', 'Q', 'P',
'R', 'S', 'A', 'T', 'B', 'C', 'E', 'D', 'F', 'G', 'H', 'I', 'J', 'U', 'V',
'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'o', 'd', 'p', 'q', 'r', 's', 't', 'u',
'v', 'w', 'x', 'e', 'f', 'g', 'h', 'j', 'i', 'k', 'l', 'm', 'n', 'y', 'z',
'0', '1', '2', '3', '4', '6', '5', '7', '8', '9', '+', '/'};

    public String getNextCheckTitle() {
        return "当前第1关";
    }

    public void onCheck() {
        try {
            if (a(b("请输入密
码:")).equals("R4jSLLLLLLLLLOrLE7/5B+Z6fsl65yj6BgC6YWz66gO6g2t65Pk6a+P65NK44NN
ROl0wNOLLLL=")) {
                CheckSuccess();
                startActivity(new Intent(this, FridaActivity2.class));
                finishActivity(0);
                return;
            }
            super.CheckFailed();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static String a(byte[] bArr) throws Exception {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i <= bArr.length - 1; i += 3) {
            byte[] bArr2 = new byte[4];
            byte b = 0;
            for (int i2 = 0; i2 <= 2; i2++) {
                int i3 = i + i2;
                if (i3 <= bArr.length - 1) {
                    bArr2[i2] = (byte) (b | ((bArr[i3] & 255) >>> ((i2 * 2) +
2)));
                    b = (byte) ((((bArr[i3] & 255) << (((2 - i2) * 2) + 2)) &
255) >>> 2);
                } else {
                    bArr2[i2] = b;
                    b = 64;
                }
            }
            bArr2[3] = b;
            for (int i4 = 0; i4 <= 3; i4++) {
                if (bArr2[i4] <= 63) {
```

```
                    sb.append(table[bArr2[i4]]);
                } else {
                    sb.append('=');
                }
            }
        }
        return sb.toString();
    }

    public static byte[] b(String str) {
        try {
            ByteArrayOutputStream byteArrayOutputStream = new
ByteArrayOutputStream();
            GZIPOutputStream gZIPOutputStream = new
GZIPOutputStream(byteArrayOutputStream);
            gZIPOutputStream.write(str.getBytes());
            gZIPOutputStream.finish();
            gZIPOutputStream.close();
            byte[] byteArray = byteArrayOutputStream.toByteArray();
            try {
                byteArrayOutputStream.close();
                return byteArray;
            } catch (Exception e) {
                e.printStackTrace();
                return byteArray;
            }
        } catch (Exception unused) {
            return null;
        }
    }
}
```

关键函数在`a(b("请输入密码:")).equals("R4jSLLLLLLLLLOrLE7/5B+Z6fsl65yj6BgC6YWz66gO6g2t65Pk6a+P65NK44NNROl0wNOLLLL=")`这里应该直接hook a，让其返回值为`R4jSLLLLLLLLLOrLE7/5B+Z6fsl65yj6BgC6YWz66gO6g2t65Pk6a+P65NK44NNROl0wNOLLLL=`就可以进入下一关了。

```
function ch1() {
    Java.perform(function () {
        console.log("start")

 Java.use("com.example.androiddemo.Activity.FridaActivity1").a.implementation
= function (x) {
            return
"R4jSLLLLLLLLLOrLE7/5B+Z6fsl65yj6BgC6YWz66gO6g2t65Pk6a+P65NK44NNROl0wNOLLLL="
        }
    })
}
```

Frida hook : 主动调用静态/非静态函数 以及 设置静态/非静态成员变量的值

总结:

- 静态函数直接use class然后调用方法，非静态函数需要先choose实例然后调用
- 设置成员变量的值，写法是`xx.value = yy`，其他方面和函数一样。
- 如果有一个成员变量和成员函数的名字相同，则在其前面加一个`_`，如`_xx.value = yy`

然后是第二关

```
public class FridaActivity2 extends BaseFridaActivity {
    private static boolean static_bool_var = false;
    private boolean bool_var = false;

    public String getNextCheckTitle() {
        return "当前第2关";
    }

    private static void setStatic_bool_var() {
        static_bool_var = true;
    }

    private void setBool_var() {
        this.bool_var = true;
    }

    public void onCheck() {
        if (!static_bool_var || !this.bool_var) {
            super.CheckFailed();
            return;
        }
        CheckSuccess();
        startActivity(new Intent(this, FridaActivity3.class));
        finishActivity(0);
    }
```

```
}
```

这一关的关键在于下面的if判断要为false，则`static_bool_var`和`this.bool_var`都要为true。

```
if (!static_bool_var || !this.bool_var) {
            super.CheckFailed();
            return;
        }
```

这样就要调用`setBool_var`和`setStatic_bool_var`两个函数了。

```
function ch2() {
    Java.perform(function () {
        console.log("start")
        var FridaActivity2 =
Java.use("com.example.androiddemo.Activity.FridaActivity2")
        //hook静态函数直接调用
        FridaActivity2.setStatic_bool_var()
        //hook动态函数，找到instance实例，从实例调用函数方法
        Java.choose("com.example.androiddemo.Activity.FridaActivity2", {
            onMatch: function (instance) {
                instance.setBool_var()
            },
            onComplete: function () {
                console.log("end")
            }
        })
    })
}
setImmediate(ch2)
```

接下来是第三关

```
public class FridaActivity3 extends BaseFridaActivity {
    private static boolean static_bool_var = false;
    private boolean bool_var = false;
    private boolean same_name_bool_var = false;

    public String getNextCheckTitle() {
        return "当前第3关";
    }

    private void same_name_bool_var() {
        Log.d("Frida", static_bool_var + " " + this.bool_var + " " +
this.same_name_bool_var);
    }

    public void onCheck() {
```

```
        if (!static_bool_var || !this.bool_var || !this.same_name_bool_var) {
            super.CheckFailed();
            return;
        }
        CheckSuccess();
        startActivity(new Intent(this, FridaActivity4.class));
        finishActivity(0);
    }
}
```

关键还是让`if (!static_bool_var || !this.bool_var || !this.same_name_bool_var)`为false，则三个变量都要为true

```
function ch3() {
    Java.perform(function () {
        console.log("start")
        var FridaActivity3 =
Java.use("com.example.androiddemo.Activity.FridaActivity3")
        FridaActivity3.static_bool_var.value = true

        Java.choose("com.example.androiddemo.Activity.FridaActivity3", {
            onMatch: function (instance) {
                instance.bool_var.value = true
                instance._same_name_bool_var.value = true
            },
            onComplete: function () {
                console.log("end")
            }
        })
    })
}
```

这里要注意类里有一个成员函数和成员变量都叫做`same_name_bool_var`，这种时候在成员变量前加一个`_`，修改值的形式为`xx.value = yy`

Frida hook : 内部类，枚举类的函数并hook，trace原型1

总结:

- 对于内部类，通过`类名$内部类名`去use或者choose
- 对use得到的clazz应用反射，如`clazz.class.getDeclaredMethods()`可以得到类里面声明的所有方法，即可以枚举类里面的所有函数。

接下来是第四关

```
public class FridaActivity4 extends BaseFridaActivity {
    public String getNextCheckTitle() {
        return "当前第4关";
    }
```

```
    private static class InnerClasses {
        public static boolean check1() {
            return false;
        }

        public static boolean check2() {
            return false;
        }

        public static boolean check3() {
            return false;
        }

        public static boolean check4() {
            return false;
        }

        public static boolean check5() {
            return false;
        }

        public static boolean check6() {
            return false;
        }

        private InnerClasses() {
        }
    }

    public void onCheck() {
        if (!InnerClasses.check1() || !InnerClasses.check2() ||
!InnerClasses.check3() || !InnerClasses.check4() || !InnerClasses.check5() ||
!InnerClasses.check6()) {
            super.CheckFailed();
            return;
        }
        CheckSuccess();
        startActivity(new Intent(this, FridaActivity5.class));
        finishActivity(0);
    }
}
```

这一关的关键是让`if (!InnerClasses.check1() || !InnerClasses.check2() || !InnerClasses.check3() || !InnerClasses.check4() || !InnerClasses.check5() || !InnerClasses.check6())`中的所有check全部返回true。

其实这里唯一的问题就是寻找内部类`InnerClasses`，对于内部类的hook，通过`类名$内部类名`去use。

```
function ch4() {
    Java.perform(function () {
        var InnerClasses =
Java.use("com.example.androiddemo.Activity.FridaActivity4$InnerClasses")
        console.log("start")
        InnerClasses.check1.implementation = function () {
            return true
        }
        InnerClasses.check2.implementation = function () {
            return true
        }
        InnerClasses.check3.implementation = function () {
            return true
        }
        InnerClasses.check4.implementation = function () {
            return true
        }
        InnerClasses.check5.implementation = function () {
            return true
        }
        InnerClasses.check6.implementation = function () {
            return true
        }
    })
}
```

利用反射，获取类中的所有method声明，然后字符串拼接去获取到方法名，例如下面的check1，然后就可以批量hook，而不用像我上面那样一个一个写。

```
var inner_classes =
Java.use("com.example.androiddemo.Activity.FridaActivity4$InnerClasses")
var all_methods = inner_classes.class.getDeclaredMethods();

...
public static boolean
com.example.androiddemo.Activity.FridaActivity4$InnerClasses.check1(),public
static boolean
com.example.androiddemo.Activity.FridaActivity4$InnerClasses.check2(),public
static boolean
com.example.androiddemo.Activity.FridaActivity4$InnerClasses.check3(),public
static boolean
com.example.androiddemo.Activity.FridaActivity4$InnerClasses.check4(),public
static boolean
com.example.androiddemo.Activity.FridaActivity4$InnerClasses.check5(),public
static boolean
com.example.androiddemo.Activity.FridaActivity4$InnerClasses.check6()
```

Frida hook : hook动态加载的dex，与查找interface，

总结:

- 通过`enumerateClassLoaders`来枚举加载进内存的classloader，再`loader.findClass(xxx)`寻找是否包括我们想要的interface的实现类，最后通过`Java.classFactory.loader = loader`来切换classloader，从而加载该实现类。

第五关比较有趣，它的check函数是动态加载进来的。 java里有interface的概念，是指一系列抽象的接口，需要类来实现。

```
package com.example.androiddemo.Dynamic;

public interface CheckInterface {
    boolean check();
}
...

public class DynamicCheck implements CheckInterface {
    public boolean check() {
        return false;
    }
}
...
public class FridaActivity5 extends BaseFridaActivity {
    private CheckInterface DynamicDexCheck = null;
    ...
    public CheckInterface getDynamicDexCheck() {
        if (this.DynamicDexCheck == null) {
            loaddex();
        }
        return this.DynamicDexCheck;
    }

    /* access modifiers changed from: protected */
    public void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        loaddex();
        //this.DynamicDexCheck = (CheckInterface) new DexClassLoader(str,
filesDir.getAbsolutePath(), (String) null,
getClassLoader()).loadClass("com.example.androiddemo.Dynamic.DynamicCheck").ne
wInstance();
    }

    public void onCheck() {
        if (getDynamicDexCheck() == null) {
            Toast.makeText(this, "onClick loaddex Failed!", 1).show();
        } else if (getDynamicDexCheck().check()) {
            CheckSuccess();
            startActivity(new Intent(this, FridaActivity6.class));
            finishActivity(0);
```

```
        } else {
            super.CheckFailed();
        }
    }
}
```

这里有个loaddex其实就是先从资源文件加载classloader到内存里，再loadClass DynamicCheck，创建出一个实例，最终调用这个实例的check。 所以现在我们就要先枚举class loader，找到能实例化我们要的class的那个class loader，然后把它设置成Java的默认class factory的loader。 现在就可以用这个class loader来使用`.use`去import一个给定的类。

```
function ch5() {
    Java.perform(function () {
        // Java.choose("com.example.androiddemo.Activity.FridaActivity5",{
        //     onMatch:function(x){
        //         console.log(x.getDynamicDexCheck().$className)
        //     },onComplete:function(){}
        // })
        console.log("start")
        Java.enumerateClassLoaders({
            onMatch: function (loader) {
                try {

 if(loader.findClass("com.example.androiddemo.Dynamic.DynamicCheck")){
                        console.log("Successfully found loader")
                        console.log(loader);
                        Java.classFactory.loader = loader ;
                    }
                }
                catch(error){
                    console.log("find error:" + error)
                }
            },
            onComplete: function () {
                console.log("end1")
            }
        })

 Java.use("com.example.androiddemo.Dynamic.DynamicCheck").check.implementation
= function () {
            return true
        }
        console.log("end2")
    })
}
setImmediate(ch5)
```

todo有一个疑问 https://github.com/frida/frida/issues/1049

Frida hook : 枚举class，trace原型2

总结: 通过`Java.enumerateLoadedClasses`来枚举类，然后`name.indexOf(str)`过滤一下并hook。

接下来是第六关

```
import com.example.androiddemo.Activity.Frida6.Frida6Class0;
import com.example.androiddemo.Activity.Frida6.Frida6Class1;
import com.example.androiddemo.Activity.Frida6.Frida6Class2;

public class FridaActivity6 extends BaseFridaActivity {
    public String getNextCheckTitle() {
        return "当前第6关";
    }

    public void onCheck() {
        if (!Frida6Class0.check() || !Frida6Class1.check() ||
!Frida6Class2.check()) {
            super.CheckFailed();
            return;
        }
        CheckSuccess();
        startActivity(new Intent(this, FridaActivity7.class));
        finishActivity(0);
    }
}
```

这关是import了一些类，然后调用类里的静态方法，所以我们枚举所有的类，然后过滤一下，并把过滤出来的结果hook上，改掉其返回值。

```
function ch6() {
    Java.perform(function () {
        Java.enumerateLoadedClasses({
            onMatch: function (name, handle){
                if (name.indexOf("com.example.androiddemo.Activity.Frida6") !=
-1) {
                    console.log("name:" + name + " handle:" + handle)
                    Java.use(name).check.implementation = function () {
                        return true
                    }
                }
            },
            onComplete: function () {
                console.log("end")
            }
        })
    })
}
```

Frida hook : 搜索interface的具体实现类

利用反射得到类里面实现的interface数组，并打印出来。

```
function more() {
    Java.perform(function () {
        Java.enumerateLoadedClasses({
            onMatch: function (class_name){
                if (class_name.indexOf("com.example.androiddemo") < 0) {
                    return
                }
                else {
                    var hook_cls = Java.use(class_name)
                    var interfaces = hook_cls.class.getInterfaces()
                    if (interfaces.length > 0) {
                        console.log(class_name + ": ")
                        for (var i in interfaces) {
                            console.log("\t", interfaces[i].toString())
                        }
                    }
                }
            },
            onComplete: function () {
                console.log("end")
            }
        })
    })
}
```

Frida hook基础（二）

- spawn/attach
- 各种主动调用
- hook函数和hook构造函数
- 调用栈/简单脚本
- 动态加载自己的dex

题目下载地址: https://github.com/tlamb96/kgb_messenger

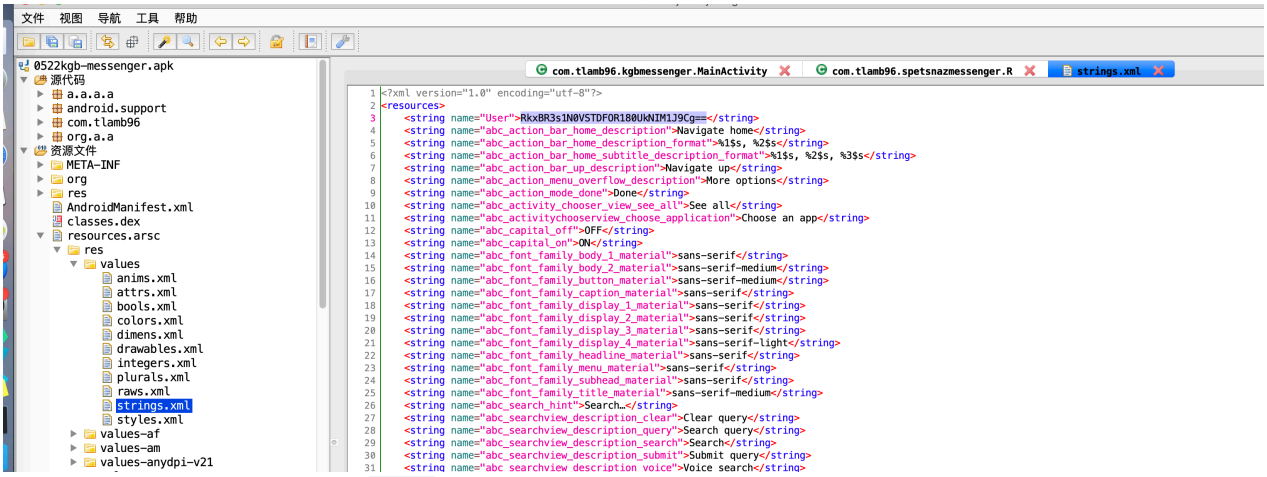
spawn/attach

firda的-f参数代表span启动 `frida -U -f com.tlamb96.spetsnazmessenger -l frida_russian.js --no-pause`

```
    /* access modifiers changed from: protected */
    public void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        setContentView((int) R.layout.activity_main);
```

```
        String property = System.getProperty("user.home");
        String str = System.getenv("USER");
        if (property == null || property.isEmpty() ||
!property.equals("Russia")) {
            a("Integrity Error", "This app can only run on Russian devices.");
        } else if (str == null || str.isEmpty() ||
!str.equals(getResources().getString(R.string.User))) {
            a("Integrity Error", "Must be on the user whitelist.");
        } else {
            a.a(this);
            startActivity(new Intent(this, LoginActivity.class));
        }
    }
}
```

这个题目比较简单，但是因为这个check是在`onCreate`里，所以app刚启动就自动检查，所以这里需要用spawn的方式去启动frida脚本hook，而不是attach。 这里有两个检查，一个是检查property的值，一个是检查str的值。 分别从`System.getProperty`和`System.getenv`里获取，hook住这两个函数就行。



这里要注意从资源文件里找到`User`的值。

```
function main() {
    Java.perform(function () {

 Java.use("java.lang.System").getProperty.overload('java.lang.String').impleme
ntation = function (str) {
            return "Russia";
        }

 Java.use("java.lang.System").getenv.overload('java.lang.String').implementati
on = function(str){
            return "RkxBR3s1N0VSTDFOR180UkNIM1J9Cg==";
        }
    })
}
setImmediate(main)
```

接下来进入到login功能

```
    public void onLogin(View view) {
        EditText editText = (EditText) findViewById(R.id.login_username);
        EditText editText2 = (EditText) findViewById(R.id.login_password);
        this.n = editText.getText().toString();
        this.o = editText2.getText().toString();
        if (this.n != null && this.o != null && !this.n.isEmpty() &&
!this.o.isEmpty()) {
            if (!this.n.equals(getResources().getString(R.string.username))) {
                Toast.makeText(this, "User not recognized.", 0).show();
                editText.setText("");
                editText2.setText("");
            } else if (!j()) {
                Toast.makeText(this, "Incorrect password.", 0).show();
                editText.setText("");
                editText2.setText("");
            } else {
                i();
                startActivity(new Intent(this, MessengerActivity.class));
            }
        }
    }
...
    private boolean j() {
        String str = "";
        for (byte b : this.m.digest(this.o.getBytes())) {
            str = str + String.format("%x", new Object[]{Byte.valueOf(b)});
        }
        return str.equals(getResources().getString(R.string.password));
    }
...
    private void i() {
        char[] cArr = {'(', 'W', 'D', ')', 'T', 'P', ':', '#', '?', 'T'};
        cArr[0] = (char) (cArr[0] ^ this.n.charAt(1));
        cArr[1] = (char) (cArr[1] ^ this.o.charAt(0));
        cArr[2] = (char) (cArr[2] ^ this.o.charAt(4));
        cArr[3] = (char) (cArr[3] ^ this.n.charAt(4));
        cArr[4] = (char) (cArr[4] ^ this.n.charAt(7));
        cArr[5] = (char) (cArr[5] ^ this.n.charAt(0));
        cArr[6] = (char) (cArr[6] ^ this.o.charAt(2));
        cArr[7] = (char) (cArr[7] ^ this.o.charAt(3));
        cArr[8] = (char) (cArr[8] ^ this.n.charAt(6));
        cArr[9] = (char) (cArr[9] ^ this.n.charAt(8));
        Toast.makeText(this, "FLAG{" + new String(cArr) + "}", 1).show();
    }
```

从资源文件里找到username,密码则是要算一个j()函数，要让它返回true，顺便打印一下i函数toast到界面的flag。

```
Java.use("com.tlamb96.kgbmessenger.LoginActivity").j.implementation = function
() {
            return true
        }
...
Java.use("android.widget.Toast").makeText.overload('android.content.Context',
'java.lang.CharSequence', 'int').implementation = function (x, y, z) {
    var flag = Java.use("java.lang.String").$new(y)
    console.log(flag)
}
...
[Google Pixel::com.tlamb96.spetsnazmessenger]-> FLAG{G&qG13     R0}
```

Frida hook :hook构造函数/打印栈回溯

总结: hook构造函数实现通过use取得类，然后`clazz.$init.implementation = callback` hook构造函数。

我们先学习一下怎么hook构造函数。

```
add(new com.tlamb96.kgbmessenger.b.a(R.string.katya, "Archer, you up?", "2:20
am", true));
...
package com.tlamb96.kgbmessenger.b;
public class a {
...
    public a(int i, String str, String str2, boolean z) {
        this.f448a = i;
        this.b = str;
        this.c = str2;
        this.d = z;
    }
...
}
```

用`$init`来hook构造函数

```
Java.use("com.tlamb96.kgbmessenger.b.a").$init.implementation = function (i,
str1, str2, z) {
            this.$init(i, str1, str2, z)
            console.log(i, str1, str2, z)
            printStack("com.tlamb96.kgbmessenger.b.a")
        }
```

Frida hook : 打印栈回溯

打印栈回溯

```
function printStack(name) {
    Java.perform(function () {
        var Exception = Java.use("java.lang.Exception");
        var ins = Exception.$new("Exception");
        var straces = ins.getStackTrace();
        if (straces != undefined && straces != null) {
            var strace = straces.toString();
            var replaceStr = strace.replace(/,/g, "\\n");
            console.log("============================" + name + " Stack
strat======================");
            console.log(replaceStr);
            console.log("============================" + name + " Stack
end======================\r\n");
            Exception.$dispose();
        }
    });
}
```

输出就是这样

```
[Google Pixel::com.tlamb96.spetsnazmessenger]-> 2131558449 111 02:27 下午 false
============================com.tlamb96.kgbmessenger.b.a Stack
strat======================
com.tlamb96.kgbmessenger.b.a.<init>(Native Method)
com.tlamb96.kgbmessenger.MessengerActivity.onSendMessage(Unknown Source:40)
java.lang.reflect.Method.invoke(Native Method)
android.support.v7.app.m$a.onClick(Unknown Source:25)
android.view.View.performClick(View.java:6294)
android.view.View$PerformClick.run(View.java:24770)
android.os.Handler.handleCallback(Handler.java:790)
android.os.Handler.dispatchMessage(Handler.java:99)
android.os.Looper.loop(Looper.java:164)
android.app.ActivityThread.main(ActivityThread.java:6494)
java.lang.reflect.Method.invoke(Native Method)
com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:4
38)
com.android.internal.os.ZygoteInit.main(ZygoteInit.java:807)
```

```
============================com.tlamb96.kgbmessenger.b.a Stack
end=======================
```

Frida hook : 手动加载dex并调用

总结： 编译出dex之后，通过`Java.openClassFile("xxx.dex").load()`加载，这样我们就可以正常通过`Java.use`调用里面的方法了。

现在我们来继续解决这个问题。

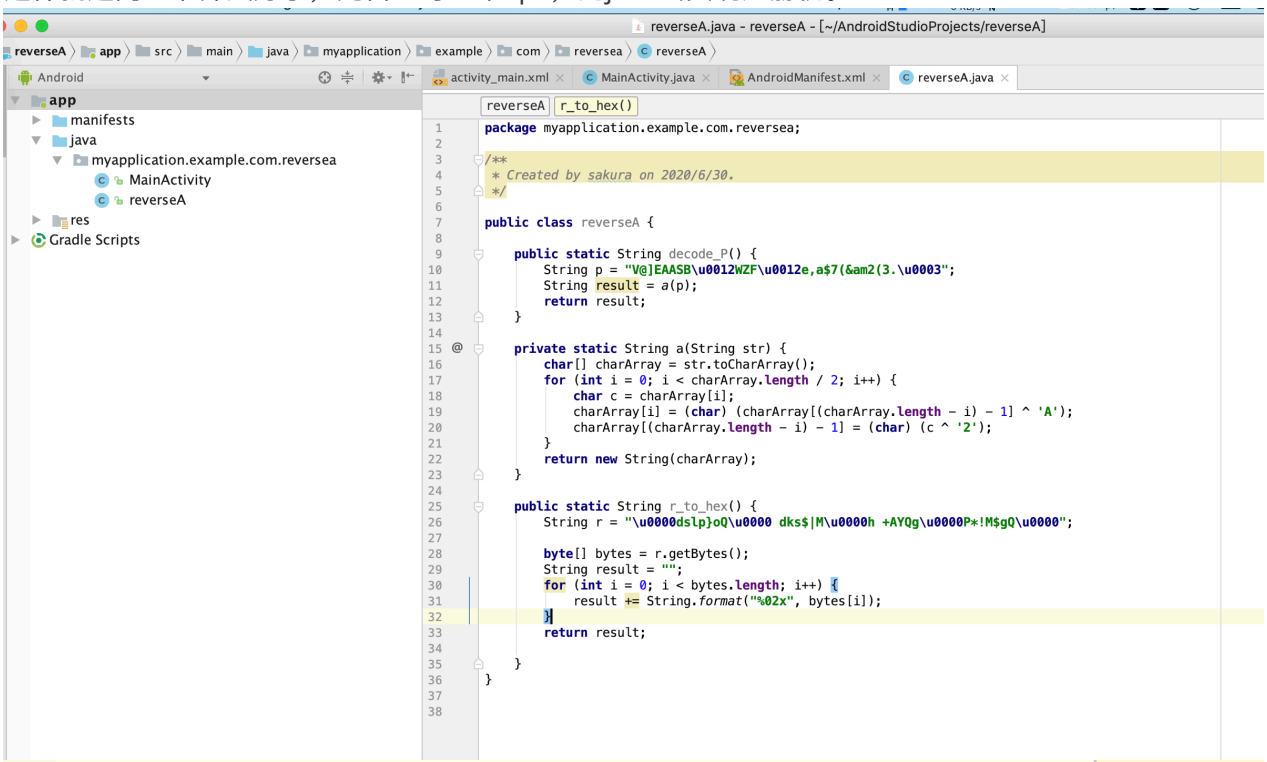
```
    public void onSendMessage(View view) {
        EditText editText = (EditText) findViewById(R.id.edittext_chatbox);
        String obj = editText.getText().toString();
        if (!TextUtils.isEmpty(obj)) {
            this.o.add(new com.tlamb96.kgbmessenger.b.a(R.string.user, obj,
j(), false));
            this.n.c();
            if (a(obj.toString()).equals(this.p)) {
                Log.d("MessengerActivity", "Successfully asked Boris for the
password.");
                this.q = obj.toString();
                this.o.add(new com.tlamb96.kgbmessenger.b.a(R.string.boris,
"Only if you ask nicely", j(), true));
                this.n.c();
            }
            if (b(obj.toString()).equals(this.r)) {
                Log.d("MessengerActivity", "Successfully asked Boris nicely
for the password.");
                this.s = obj.toString();
                this.o.add(new com.tlamb96.kgbmessenger.b.a(R.string.boris,
"Wow, no one has ever been so nice to me! Here you go friend: FLAG{" + i() +
"}", j(), true));
                this.n.c();
            }
            this.m.b(this.m.getAdapter().a() - 1);
            editText.setText("");
        }
    }
```

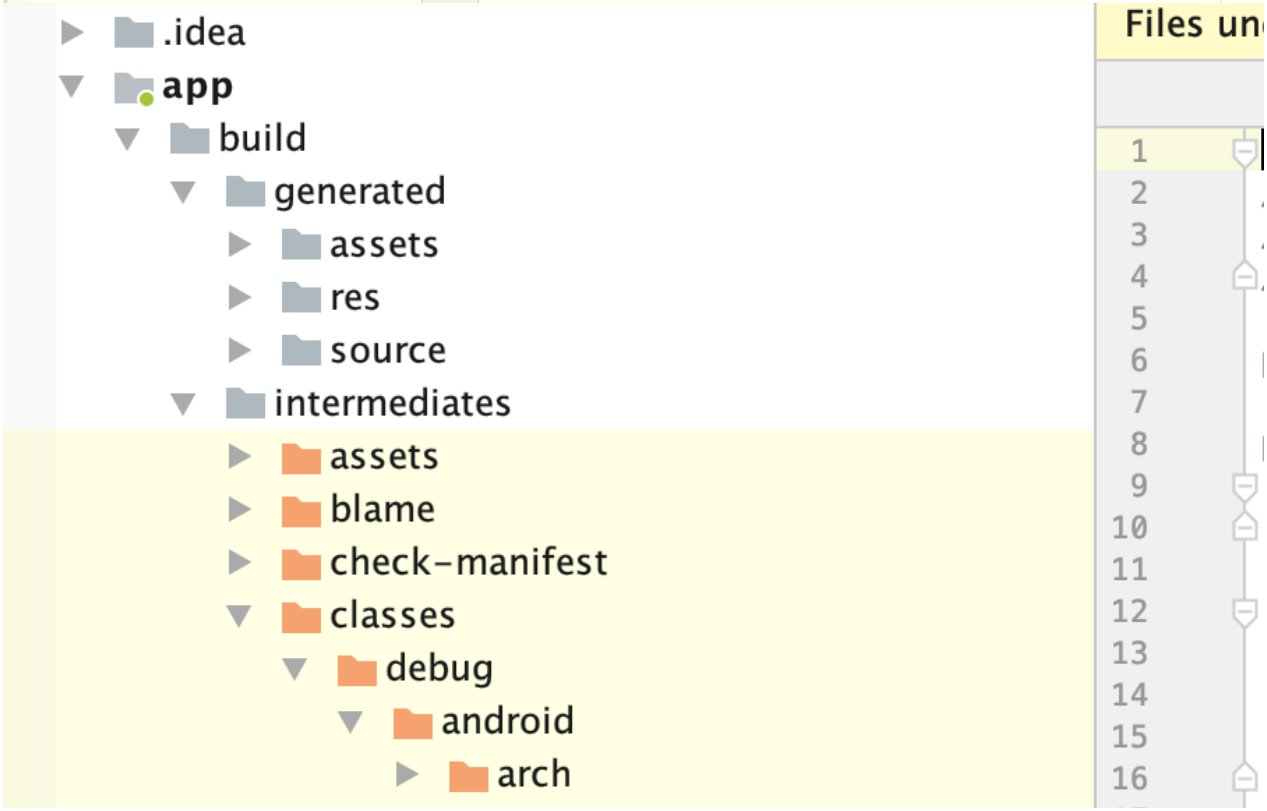
新的一关是一个聊天框，分析一下代码可知，obj是我们输入的内容，输入完了之后，加到一个`this.o`的ArrayList里。 关键的if判断就是`if (a(obj.toString()).equals(this.p))`和`if (b(obj.toString()).equals(this.r))`，所有hook a和b函数，让它们的返回值等于下面的字符串即可。

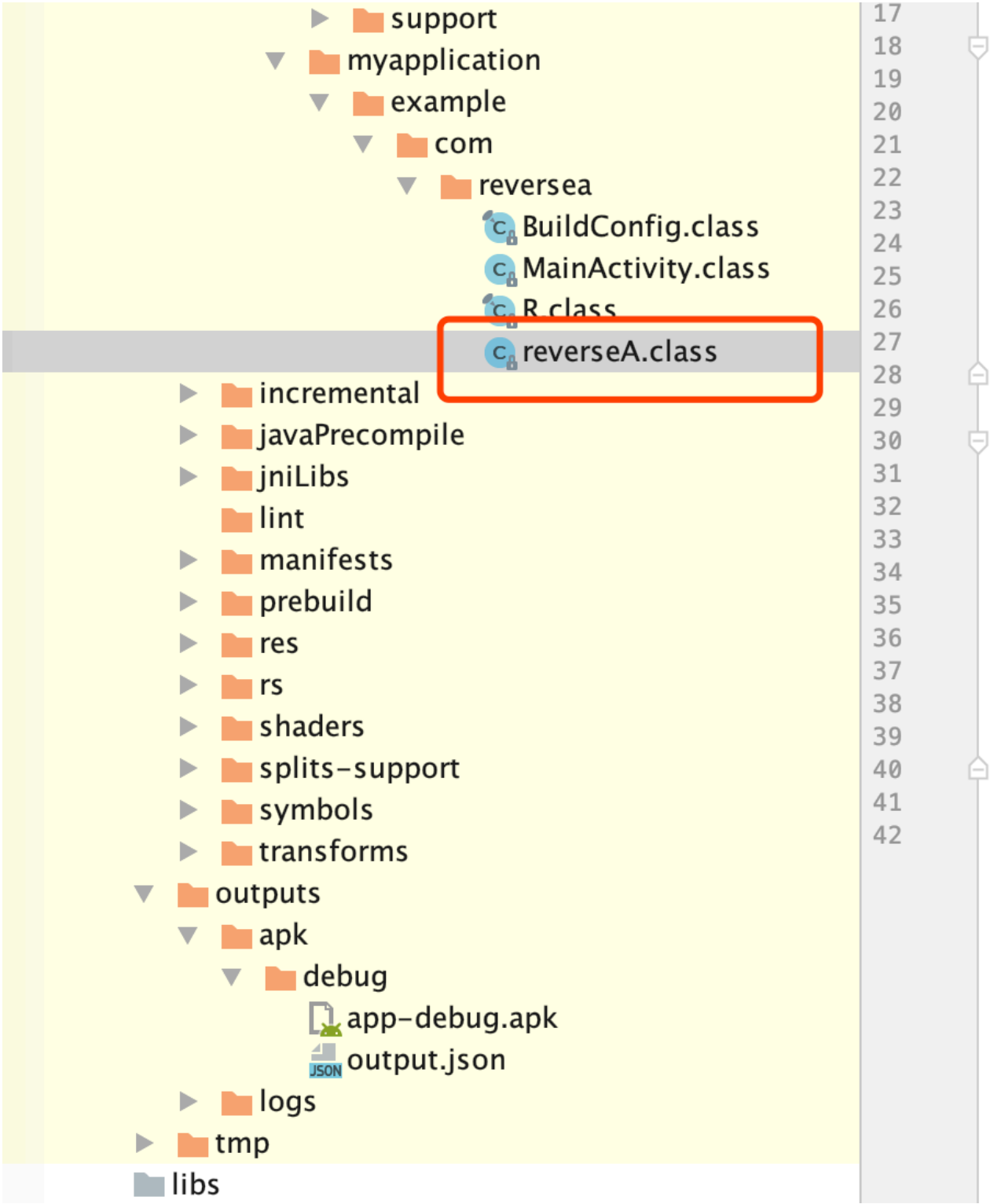
```
private String p = "V@]EAASB\u0012WZF\u0012e,a$7(&am2(3.\u0003";
private String q;
private String r = "\u0000dslp}oQ\u0000 dks$|M\u0000h
+AYQg\u0000P*!M$gQ\u0000";
private String s;
```

但实际上这题比我想象中的还要麻烦，这题的逻辑上是如果通过了a和b这两个函数的计算，等于对应的值之后，会把用来计算的obj的值赋值给q和s，然后根据这个q和s来计算出最终的flag。 所以如果不逆向算法，通过hook的方式通过了a和b的计算，obj的值还是错误的，也计算不出正确的flag。

这样就逆向一下算法好了，先自己写一个apk，用java去实现注册机。







可以直接把class文件转成dex，不复述，我比较懒，所以我直接解压apk找到`classes.dex`，并push到手机上。 然后用frida加载这个dex，并调用里面的方法。

```
var dex = Java.openClassFile("/data/local/tmp/classes.dex").load();

 console.log("decode_P:"+Java.use("myapplication.example.com.reversea.reverseA
").decode_P());

 console.log("r_to_hex:"+Java.use("myapplication.example.com.reversea.reverseA
").r_to_hex());
...
...
decode_P:Boris, give me the password
r_to_hex:0064736c707d6f510020646b73247c4d0068202b4159516700502a214d24675100
```

Frida打印与参数构造

- 数组/(字符串)对象数组/gson/Java.array
- 对象/多态、强转Java.cast/接口Java.register
- 泛型、List、Map、Set、迭代打印
- non-ascii 、 child-gating、rpc 上传到PC上打印

char[]/[Object Object]

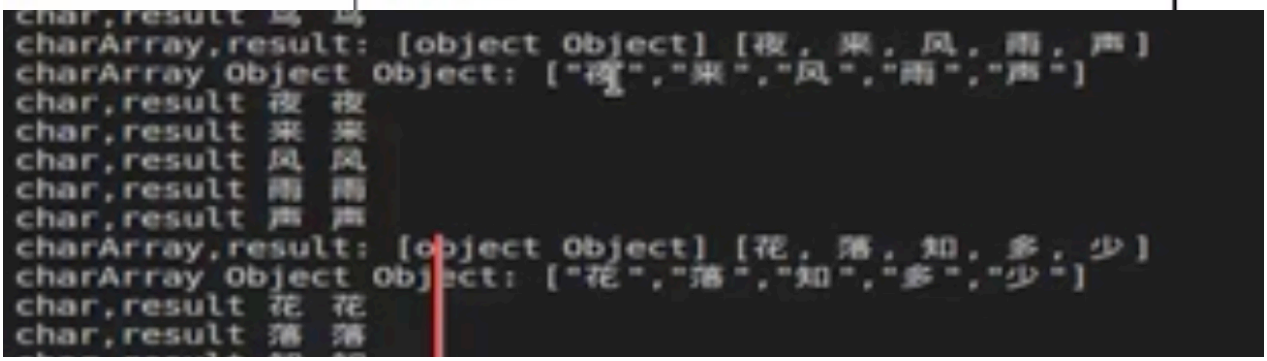
```
Log.d("SimpleArray", "onCreate: SImpleArray");
char arr[][] = new char[4][]; // 创建一个4行的二维数组
arr[0] = new char[] { '春', '眠', '不', '觉', '晓' }; // 为每一行赋值
arr[1] = new char[] { '处', '处', '闻', '啼', '鸟' };
arr[2] = new char[] { '夜', '来', '风', '雨', '声' };
arr[3] = new char[] { '花', '落', '知', '多', '少' };
Log.d("SimpleArray", "-----横版-----");
for (int i = 0; i < 4; i++) { // 循环4行
    Log.d("SimpleArraysToString", Arrays.toString(arr[i]));
    Log.d("SimpleStringBytes", Arrays.toString (Arrays.toString
(arr[i]).getBytes()));
    for (int j = 0; j < 5; j++) { // 循环5列
        Log.d("SimpleArray", Character.toString(arr[i][j])); // 输出数组中的元素
    }
    if (i % 2 == 0) {
        Log.d("SimpleArray", ",");// 如果是一、三句，输出逗号
    } else {
        Log.d("SimpleArray", "。");// 如果是二、四句，输出句号
    }
}
```

```
Java.openClassFile("/data/local/tmp/r0gson.dex").load();
const gson = Java.use('com.r0ysue.gson.Gson');

Java.use("java.lang.Character").toString.overload('char').implementation =
function(char){
```

```
    var result = this.toString(char);
    console.log("char,result",char,result);
    return result;
}

Java.use("java.util.Arrays").toString.overload('[C').implementation =
function(charArray){
    var result = this.toString(charArray);
    console.log("charArray,result:",charArray,result)
    console.log("charArray Object Object:",gson.$new().toJson(charArray));
    return result;
}
```

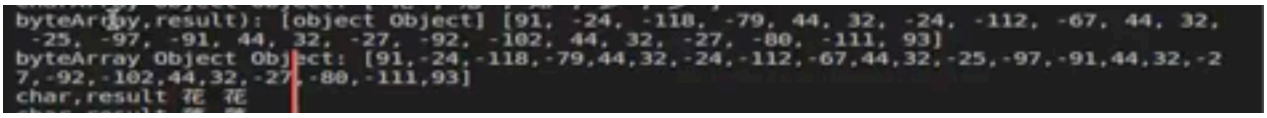
这里的`[C`是JNI函数签名

Field Descriptor	Java Language Type
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double



byte[]

```
Java.openClassFile("/data/local/tmp/r0gson.dex").load();
const gson = Java.use('com.r0ysue.gson.Gson');

Java.use("java.util.Arrays").toString.overload('[B').implementation =
function(byteArray){
    var result = this.toString(byteArray);
    console.log("byteArray,result):",byteArray,result)
    console.log("byteArray Object Object:",gson.$new().toJson(byteArray));
    return result;
}
```



java array构造

如果不只是想打印出结果，而是要替换原本的参数，就要先自己构造出一个charArray,使用`Java.array`API

```
    /**
     * Creates a Java array with elements of the specified `type`, from a
     * JavaScript array `elements`. The resulting Java array behaves like
     * a JS array, but can be passed by reference to Java APIs in order to
     * allow them to modify its contents.
     *
     * @param type Type name of elements.
     * @param elements Array of JavaScript values to use for constructing the
     *                 Java array.
     */
    function array(type: string, elements: any[]): any[];
```

```
Java.use("java.util.Arrays").toString.overload('[C').implementation =
function(charArray){
    var newCharArray = Java.array('char', [ '一','去','二','三','里' ]);
    var result = this.toString(newCharArray);
    console.log("newCharArray,result:",newCharArray,result)
    console.log("newCharArray Object
Object:",gson.$new().toJson(newCharArray));
    var newResult = Java.use('java.lang.String').$new(Java.array('char', [
'烟','村','四','五','家']))
    return newResult;
}
```

可以用来构造参数重发包，用在爬虫上。

类的多态：转型/Java.cast

可以通过`getClass().getName().toString()`来查看当前实例的类型。 找到一个instance，通过`Java.cast`来强制转换对象的类型。

```
    /**
     * Creates a JavaScript wrapper given the existing instance at `handle` of
     * given class `klass` as returned from `Java.use()`.
     *
     * @param handle An existing wrapper or a JNI handle.
     * @param klass Class wrapper for type to cast to.
     */
    function cast(handle: Wrapper | NativePointerValue, klass: Wrapper):
Wrapper;
```

```
public class Water { // 水 类
    public static String flow(Water W) { // 水 的方法
        // SomeSentence
        Log.d("2Object", "water flow: I`m flowing");
        return "water flow: I`m flowing";
    }

    public String still(Water W) { // 水 的方法
        // SomeSentence
        Log.d("2Object", "water still: still water runs deep!");
        return "water still: still water runs deep!";
    }
}
...
public class Juice extends Water { // 果汁 类 继承了水类

    public String fillEnergy(){
        Log.d("2Object", "Juice: i`m fillingEnergy!");
        return "Juice: i`m fillingEnergy!";
    }
```

```
var JuiceHandle = null ;
Java.choose("com.r0ysue.a0526printout.Juice",{
    onMatch:function(instance){
        console.log("found juice instance",instance);
        console.log("juice instance call fill",instance.fillEnergy());
        JuiceHandle = instance;
    },onComplete:function(){
        console.log("juice handle search completed!")
    }
})
console.log("Saved juice handle :",JuiceHandle);
var WaterHandle =
Java.cast(JuiceHandle,Java.use("com.r0ysue.a0526printout.Water"))
console.log("call Waterhandle still method:",WaterHandle.still(WaterHandle));
```

interface/Java.registerClass

```
public interface liquid {
    public String flow();
}
```

frida提供能力去创建一个新的java class

```
/**
    * Creates a new Java class.
    *
    * @param spec Object describing the class to be created.
    */
function registerClass(spec: ClassSpec): Wrapper;
```

首先获取要实现的interface，然后调用registerClass来实现interface。

```
    Java.perform(function(){
        var liquid = Java.use("com.r0ysue.a0526printout.liquid");
        var beer = Java.registerClass({
            name: 'com.r0ysue.a0526printout.beer',
            implements: [liquid],
            methods: {
                flow: function () {
                    console.log("look, beer is flowing!")
                    return "look, beer is flowing!";
                }
            }
        });
        console.log("beer.bubble:",beer.$new().flow())
    })
}
```

成员内部类/匿名内部类

看smali或者枚举出来的类。

hook enum

关于java枚举，从这篇文章了解。 https://www.cnblogs.com/jingmoxukong/p/6098351.html

```
enum Signal {
    GREEN, YELLOW, RED
}
public class TrafficLight {
    public static Signal color = Signal.RED;
    public static void main() {
        Log.d("4enum", "enum "+ color.getClass().getName().toString());
        switch (color) {
            case RED:
                color = Signal.GREEN;
                break;
            case YELLOW:
                color = Signal.RED;
                break;
            case GREEN:
                color = Signal.YELLOW;
                break;
        }
    }
}
```

```
Java.perform(function(){
        Java.choose("com.r0ysue.a0526printout.Signal",{
            onMatch:function(instance){
                console.log("instance.name:",instance.name());

 console.log("instance.getDeclaringClass:",instance.getDeclaringClass());

            },onComplete:function(){
                console.log("search completed!")
            }
        })
    })
```

打印hash map

```
Java.perform(function(){
        Java.choose("java.util.HashMap",{
            onMatch:function(instance){
                if(instance.toString().indexOf("ISBN")!= -1){
                    console.log("instance.toString:",instance.toString());
                }
            },onComplete:function(){
                console.log("search complete!")
            }
        })
    })
```

打印non-ascii

https://api-caller.com/2019/03/30/frida-note/#non-ascii 类名非ASCII字符串时，先编码打印出来, 再用编码后的字符串去 hook.

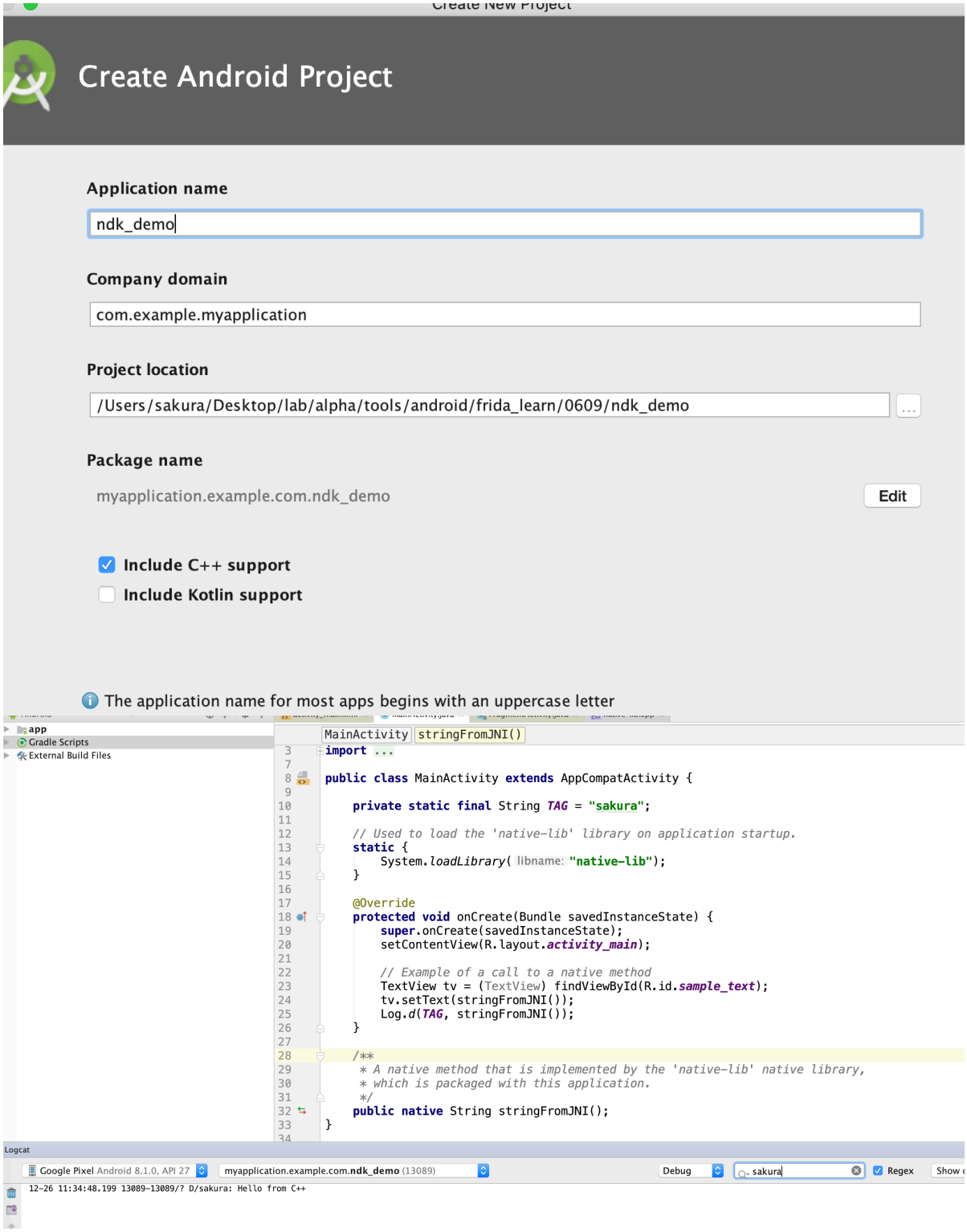
```
//场景hook cls.forName寻找目标类的classloader。
    cls.forName.overload('java.lang.String', 'boolean',
'java.lang.ClassLoader').implementation = function (arg1, arg2, arg3) {
        var clsName = cls.forName(arg1, arg2, arg3);
        console.log('oriClassName:' + arg1)
        var base64Name = encodeURIComponent(arg1)
        console.log('encodeName:' + base64Name);
        //通过日志确认base64后的非ascii字符串，下面对比并打印classloader
        //clsName为特殊字符o.îɛ«
        if ('o.%CE%99%C9%AB' == base64Name) {
            //打印classloader
            console.log(arg3);
        }
        return clsName;
    }
```

Frida native hook : NDK开发入门

https://www.jianshu.com/p/87ce6f565d37

- extern "C"与名称修饰
    - 通过c++filt工具可以直接还原得到原来的函数名
    - https://zh.wikipedia.org/zh-hans/%E5%90%8D%E5%AD%97%E4%BF%AE%E9%A5%B0
    - 通过extern "C"导出的JNI函数不会被name mangling
- JNI参数与基本类型

- 第一个NDK程序



- JNI log

```
#define TAG "sakura1328"
#define LOGI(...) __android_log_print(ANDROID_LOG_INFO,TAG,__VA_ARGS__)
#define LOGD(...) __android_log_print(ANDROID_LOG_DEBUG, TAG, __VA_ARGS__)
#define LOGE(...) __android_log_print(ANDROID_LOG_ERROR,TAG,__VA_ARGS__)

extern "C" JNIEXPORT jstring JNICALL
Java_myapplication_example_com_ndk_1demo_MainActivity_stringFromJNI(
        JNIEnv *env,
```

```
        jobject /* this */) {
    std::string hello = "Hello from C++";
    LOGD("sakura1328");
    return env->NewStringUTF(hello.c_str());
}
...
public class MainActivity extends AppCompatActivity {

    private static final String TAG = "sakura";

    // Used to load the 'native-lib' library on application startup.
    static {
        System.loadLibrary("native-lib");
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Example of a call to a native method
        TextView tv = (TextView) findViewById(R.id.sample_text);
        tv.setText(stringFromJNI());
        Log.d(TAG, stringFromJNI());
    }

    /**
     * A native method that is implemented by the 'native-lib' native library,
     * which is packaged with this application.
     */
    public native String stringFromJNI();
}
```

Frida native hook : JNIEnv和反射

以jni字符串来掌握基本的JNIEnv用法

```
public native String stringWithJNI(String context);
...

extern "C"
JNIEXPORT jstring JNICALL
Java_myapplication_example_com_ndk_1demo_MainActivity_stringWithJNI(JNIEnv
*env, jobject instance,
                                                                    jstring
context_) {
    const char *context = env->GetStringUTFChars(context_, 0);
```

```
    int context_size = env->GetStringUTFLength(context_);

    if (context_size > 0) {
        LOGD("%s\n", context);
    }

    env->ReleaseStringUTFChars(context_, context);

    return env->NewStringUTF("sakura1328");
}

12-26 22:30:00.548 15764-15764/myapplication.example.com.ndk_demo
D/sakura1328: sakura
```

Java反射

总结: 多去读一下java的反射API。

Java高级特性——反射

- 查找调用各种API接口、JNI、frida/xposed原理的一部分
- 反射基本API
- 反射修改访问控制、修改属性值
- JNI so调用反射进入java世界
- xposed/Frida hook原理

这里其实有一个伏笔，就是为什么我们要trace artmethod，hook artmethod是因为有些so混淆得非常厉害，然后也就很难静态分析看出so里面调用了哪些java函数，也不是通过类似JNI的GetMethodID这样来调用的。 而是通过类似findclass这种方法先得到类，然后再反射调用app里面的某个java函数。

所以去hook它执行的位置，每一个java函数对于Android源码而言都是一个artmethod结构体，然后hook拿到artmethod实例以后调用类函数，打印这个函数的名称。

```
public class MainActivity extends AppCompatActivity {

    private static final String TAG = "sakura";

    // Used to load the 'native-lib' library on application startup.
    static {
        System.loadLibrary("native-lib");
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Example of a call to a native method
        TextView tv = (TextView) findViewById(R.id.sample_text);
        tv.setText(stringWithJNI("sakura"));
```

```
//        Log.d(TAG, stringFromJNI());
//        Log.d(TAG, stringWithJNI("sakura"));
        try {
            testClass();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (NoSuchFieldException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        } catch (NoSuchMethodException e) {
            e.printStackTrace();
        } catch (InvocationTargetException e) {
            e.printStackTrace();
        }
    }

    public void testClass() throws ClassNotFoundException,
NoSuchFieldException, IllegalAccessException, NoSuchMethodException,
InvocationTargetException {
        Test sakuraTest = new Test();
        // 获得Class的方法（三种）
        Class testClazz =
MainActivity.class.getClassLoader().loadClass("myapplication.example.com.ndk_d
emo.Test");
        Class testClazz2 =
Class.forName("myapplication.example.com.ndk_demo.Test");
        Class testClazz3 = Test.class;
        Log.i(TAG, "Classloader.loadClass->" + testClazz);
        Log.i(TAG, "Classloader.loadClass->" + testClazz2);
        Log.i(TAG, "Classloader.loadClass->" + testClazz3.getName());

        // 获得类中属性相关的方法
        Field publicStaticField =
testClazz3.getDeclaredField("publicStaticField");
        Log.i(TAG, "testClazz3.getDeclaredField->" + publicStaticField);

        Field publicField = testClazz3.getDeclaredField("publicField");
        Log.i(TAG, "testClazz3.getDeclaredField->" + publicField);

        //对于Field的get方法，如果是static，则传入null即可;如果不是，则需要传入一个类的
实例
        String valueStaticPublic = (String) publicStaticField.get(null);
        Log.i(TAG, "publicStaticField.get->" + valueStaticPublic);

        String valuePublic = (String) publicField.get(sakuraTest);
        Log.i(TAG, "publicField.get->" + valuePublic);

        //对于private属性，需要设置Accessible
```

```
        Field privateStaticField =
testClazz3.getDeclaredField("privateStaticField");
        privateStaticField.setAccessible(true);

        String valuePrivte = (String) privateStaticField.get(null);
        Log.i(TAG, "modified before privateStaticField.get->" + valuePrivte);

        privateStaticField.set(null, "modified");

        valuePrivte = (String) privateStaticField.get(null);
        Log.i(TAG, "modified after privateStaticField.get->" + valuePrivte);

        Field[] fields = testClazz3.getDeclaredFields();
        for (Field i : fields) {
            Log.i(TAG, "testClazz3.getDeclaredFields->" + i);
        }

        // 获得类中method相关的方法
        Method publicStaticMethod =
testClazz3.getDeclaredMethod("publicStaticFunc");
        Log.i(TAG, "testClazz3.getDeclaredMethod->" + publicStaticMethod);

        publicStaticMethod.invoke(null);

        Method publicMethod = testClazz3.getDeclaredMethod("publicFunc",
java.lang.String.class);
        Log.i(TAG, "testClazz3.getDeclaredMethod->" + publicMethod);

        publicMethod.invoke(sakuraTest, " sakura");
    }

    /**
     * A native method that is implemented by the 'native-lib' native library,
     * which is packaged with this application.
     */
    public native String stringFromJNI();

    public native String stringWithJNI(String context);
}
...
public class Test {
    private static final String TAG = "sakura_test";

    public static String publicStaticField = "i am a publicStaticField";
    public String publicField = "i am a publicField";

    private static String privateStaticField = "i am a privateStaticField";
    private String privateField = "i am a privateField";
```

```
    public static void publicStaticFunc() {
        Log.d(TAG, "I`m from publicStaticFunc");
    }

    public void publicFunc(String str) {
        Log.d(TAG, "I`m from publicFunc" + str);
    }

    private static void privateStaticFunc() {
        Log.i(TAG, "I`m from privateFunc");
    }

    private void privateFunc() {
        Log.i(TAG, "I`m from privateFunc");
    }
}
...
...
12-26 23:57:11.784 17682-17682/myapplication.example.com.ndk_demo I/sakura:
Classloader.loadClass->class myapplication.example.com.ndk_demo.Test
12-26 23:57:11.784 17682-17682/myapplication.example.com.ndk_demo I/sakura:
Classloader.loadClass->class myapplication.example.com.ndk_demo.Test
12-26 23:57:11.784 17682-17682/myapplication.example.com.ndk_demo I/sakura:
Classloader.loadClass->myapplication.example.com.ndk_demo.Test
12-26 23:57:11.785 17682-17682/myapplication.example.com.ndk_demo I/sakura:
testClazz3.getDeclaredField->public static java.lang.String
myapplication.example.com.ndk_demo.Test.publicStaticField
12-26 23:57:11.785 17682-17682/myapplication.example.com.ndk_demo I/sakura:
testClazz3.getDeclaredField->public java.lang.String
myapplication.example.com.ndk_demo.Test.publicField
12-26 23:57:11.785 17682-17682/myapplication.example.com.ndk_demo I/sakura:
publicStaticField.get->i am a publicStaticField
12-26 23:57:11.785 17682-17682/myapplication.example.com.ndk_demo I/sakura:
publicField.get->i am a publicField
12-26 23:57:11.785 17682-17682/myapplication.example.com.ndk_demo I/sakura:
modified before privateStaticField.get->i am a privateStaticField
12-26 23:57:11.785 17682-17682/myapplication.example.com.ndk_demo I/sakura:
modified after privateStaticField.get->modified
12-26 23:57:11.785 17682-17682/myapplication.example.com.ndk_demo I/sakura:
testClazz3.getDeclaredFields->private java.lang.String
myapplication.example.com.ndk_demo.Test.privateField
12-26 23:57:11.785 17682-17682/myapplication.example.com.ndk_demo I/sakura:
testClazz3.getDeclaredFields->public java.lang.String
myapplication.example.com.ndk_demo.Test.publicField
12-26 23:57:11.785 17682-17682/myapplication.example.com.ndk_demo I/sakura:
testClazz3.getDeclaredFields->private static final java.lang.String
myapplication.example.com.ndk_demo.Test.TAG
```

```
12-26 23:57:11.785 17682-17682/myapplication.example.com.ndk_demo I/sakura:
testClazz3.getDeclaredFields->private static java.lang.String
myapplication.example.com.ndk_demo.Test.privateStaticField
12-26 23:57:11.785 17682-17682/myapplication.example.com.ndk_demo I/sakura:
testClazz3.getDeclaredFields->public static java.lang.String
myapplication.example.com.ndk_demo.Test.publicStaticField
12-26 23:57:11.785 17682-17682/myapplication.example.com.ndk_demo I/sakura:
testClazz3.getDeclaredMethod->public static void
myapplication.example.com.ndk_demo.Test.publicStaticFunc()
12-26 23:57:11.785 17682-17682/myapplication.example.com.ndk_demo
D/sakura_test: I`m from publicStaticFunc
12-26 23:57:11.786 17682-17682/myapplication.example.com.ndk_demo I/sakura:
testClazz3.getDeclaredMethod->public void
myapplication.example.com.ndk_demo.Test.publicFunc(java.lang.String)
12-26 23:57:11.786 17682-17682/myapplication.example.com.ndk_demo
D/sakura_test: I`m from publicFunc sakura
```

```
memory list modules
```


```
},
{
  "name": "libnative-lib.so",
  "base": "0x79f9fe7000",
  "size": 323584,
  "path": "/data/app/myapplication.example.com.ndk_demo-p9oGsr1LAib7mAlbSWKM_A==/lib/arm64/libnative-lib.so"
},
{
  "name": "gralloc.msm8996.so",
  "base": "0x79f9e66000",
  "size": 77824,
  "path": "/vendor/lib64/hw/gralloc.msm8996.so"
},
{
  "name": "libqdMetaData.so",
  "base": "0x79f9e8b000",
  "size": 16384,
  "path": "/vendor/lib64/libqdMetaData.so"
},
{
  "name": "frida-agent-64.so",
  "base": "0x79f7ae2000",
  "size": 20021248,
  "path": "/data/local/tmp/re.frida.server/frida-agent-64.so"
},
{
```


Frida反调试

这一节的主要内容就是关于反调试的原理和如何破解反调试，重要内容还是看文章理解即可。 因为我并不需要做反调试相关的工作，所以部分内容略过。

- Frida反调试与反反调试基本思路 （Java层API、Native层API、Syscall）
  - AntiFrida
  - frida-detection-demo
  - 多种特征检测Frida
  - 来自高维的对抗 - 逆向TinyTool自制
  - Unicorn 在 Android 的应用

Frida native hook : 符号hook JNI、art&libc

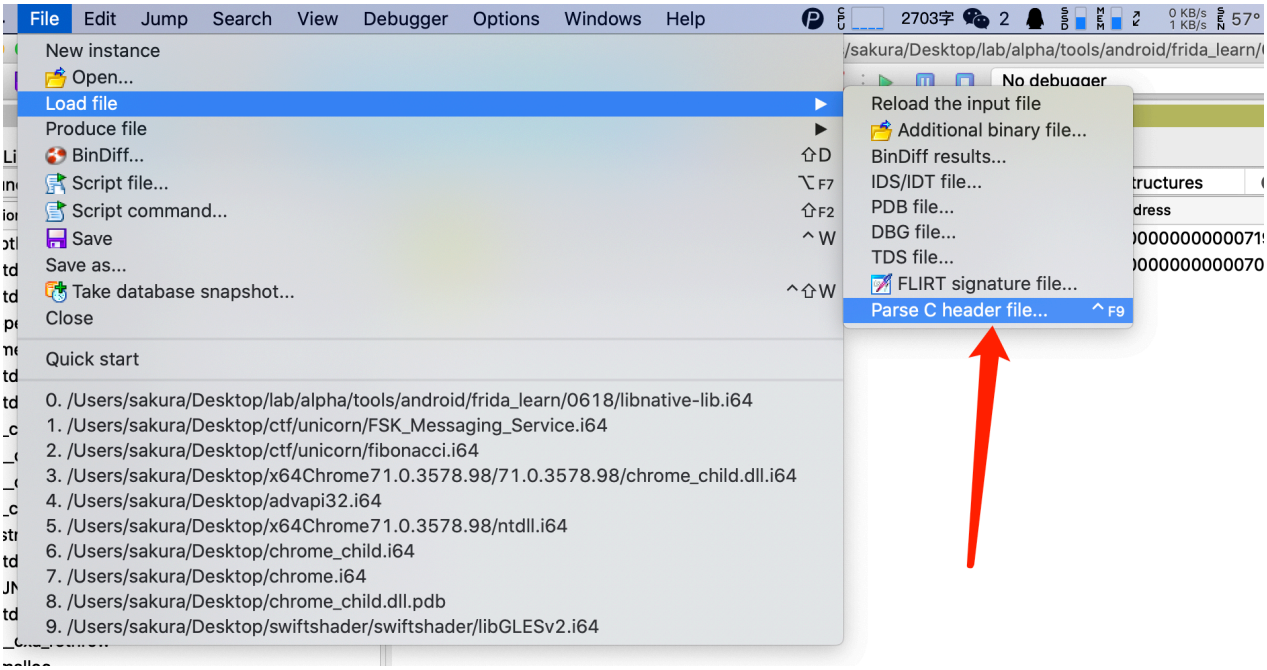
Native函数的Java Hook及主动调用

对native函数的java层hook和主动调用和普通java函数完全一致，略过。

`jni.h`头文件导入

导入jni.h，先search一下这个文件在哪。

```
sakura@sakuradeMacBook-Pro:~/Library/Android/sdk$ find ./ -name "jni.h"
.//ndk-bundle/sysroot/usr/include/jni.h
```



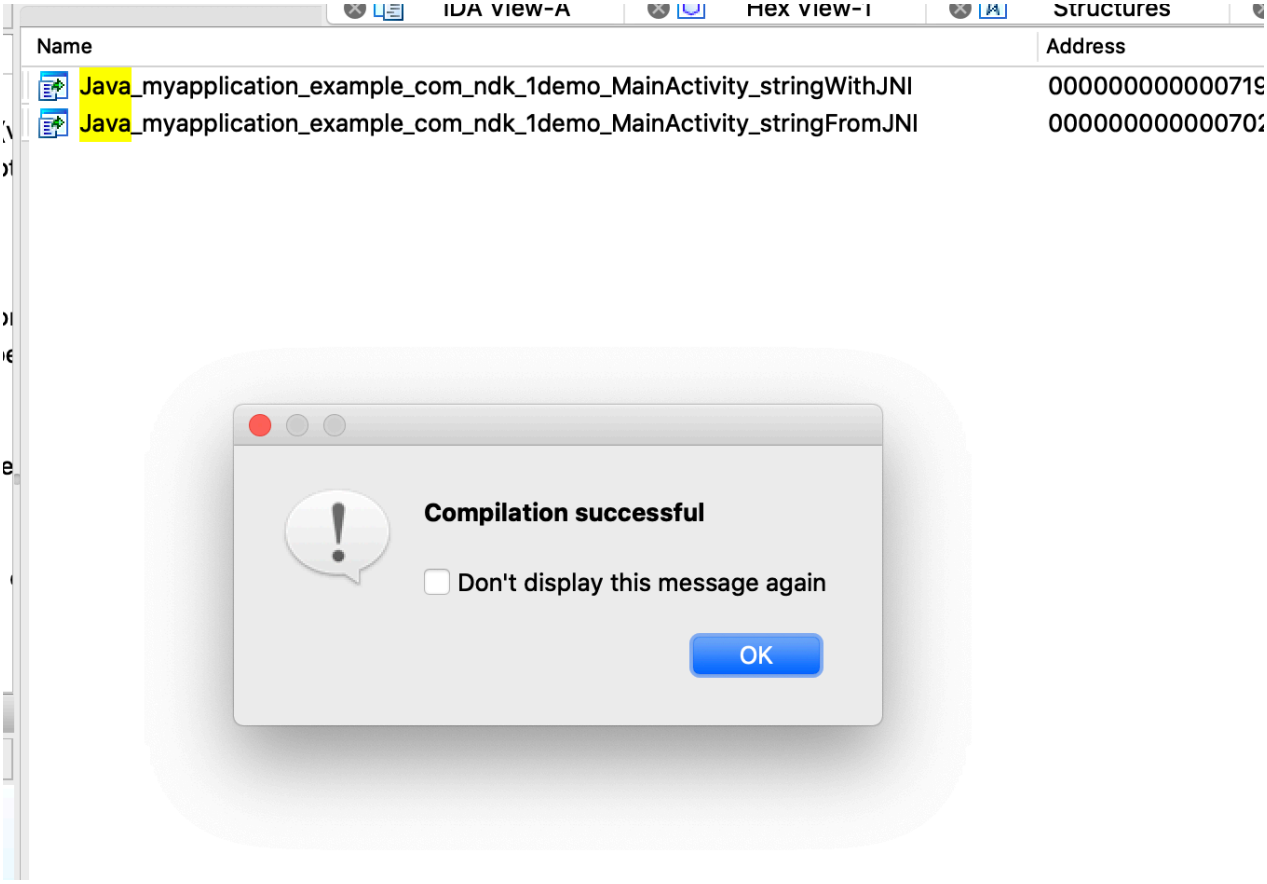

```
Error /Users/sakura/Library/Android/sdk/ndk-
bundle/sysroot/usr/include/jni.h,27: Can't open include file 'stdarg.h'
Total 1 errors
Caching 'Exports'... ok
```

报错，所以拷贝一份jni.h出来

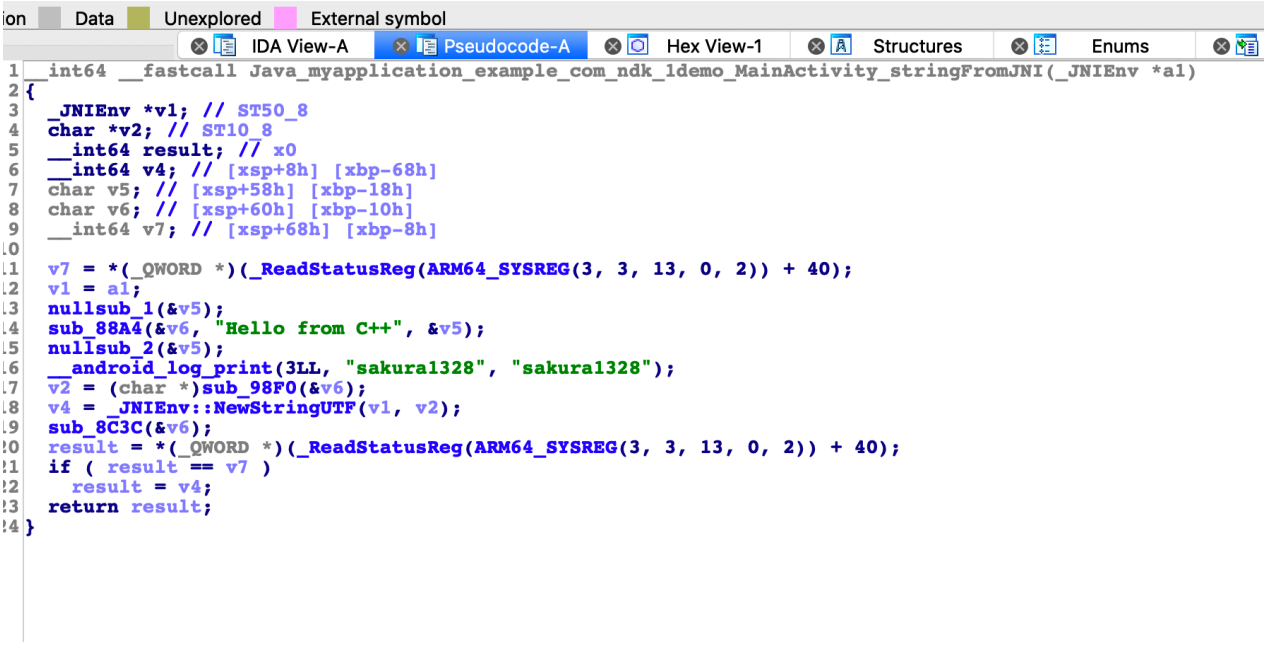
将这两个头文件导入删掉

```
 */

#ifndef JNI_H_
#define JNI_H_

#include <stdarg.h>
#include <stdint.h>

/* Primitive types that match up with Java equivalents. */
typedef uint8_t  jboolean; /* unsigned 8 bits */
typedef int8_t   jbyte;    /* signed 8 bits */
typedef uint16_t jchar;    /* unsigned 16 bits */
typedef int16_t  jshort;   /* signed 16 bits */
```

导入成功

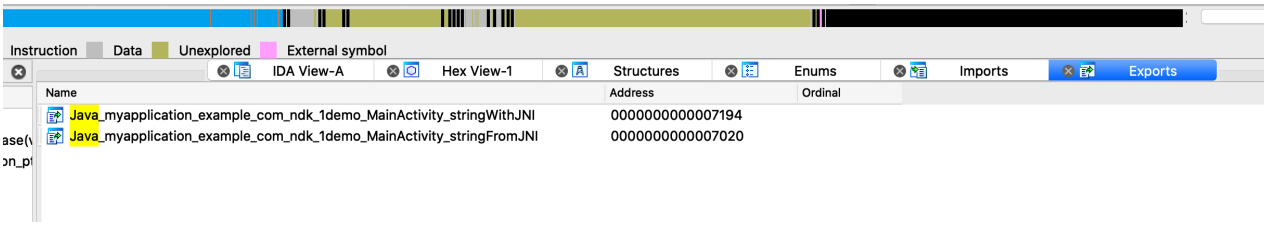


现在就能识别_JNIEnv了，如图



JNI函数符号hook

先查看一下导出了哪些函数。




```
extern "C" JNIEXPORT jstring JNICALL
Java_myapplication_example_com_ndk_1demo_MainActivity_stringFromJNI(
        JNIEnv *env,
        jobject /* this */) {
    std::string hello = "Hello from C++";
    LOGD("sakura1328");
    return env->NewStringUTF(hello.c_str());
}
extern "C"
JNIEXPORT jstring JNICALL
Java_myapplication_example_com_ndk_1demo_MainActivity_stringWithJNI(JNIEnv
*env, jobject instance,
                                                                    jstring
context_) {
    const char *context = env->GetStringUTFChars(context_, 0);

    int context_size = env->GetStringUTFLength(context_);

    if (context_size > 0) {
        LOGD("%s\n", context);
    }
```

```
    env->ReleaseStringUTFChars(context_, context);

    return env->NewStringUTF("sakura1328");
}
```

这里有几个需要的API。

- 首先是找到是否so被加载，通过`Process.enumerateModules()`,这个API可以枚举被加载到内存的modules。
- 然后通过`Module.findBaseAddress(module name)`来查找要hook的函数所在的so的基地址，如果找不到就返回null。
- 然后可以通过`findExportByName(moduleName: string, exportName: string): NativePointer`来查找导出函数的绝对地址。如果不知道moduleName是什么，可以传入一个null进入，但是会花费一些时间遍历所有的module。如果找不到就返回null。
- 找到地址之后，就可以拦截function/instruction的执行。通过`Interceptor.attach`。使用方法见下代码。
- 另外为了将jstring的值打印出来，可以使用jenv的函数getStringUtfChars，就像正常的写native程序一样。`Java.vm.getEnv().getStringUtfChars(args[2], null).readCString()`

这里我是循环调用的string_with_jni，如果不循环调用，那就要主动调用一下这个函数，或者hook dlopen。 hook dlopen的方法在这个代码可以参考。

```
function hook_native() {
    // console.log(JSON.stringify(Process.enumerateModules()));
    var libnative_addr = Module.findBaseAddress("libnative-lib.so")
    console.log("libnative_addr is: " + libnative_addr)

    if (libnative_addr) {
        var string_with_jni_addr = Module.findExportByName("libnative-lib.so",
        "Java_myapplication_example_com_ndk_1demo_MainActivity_stringWithJNI")
        console.log("string_with_jni_addr is: " + string_with_jni_addr)
    }

    Interceptor.attach(string_with_jni_addr, {
        onEnter: function (args) {
            console.log("string_with_jni args: " + args[0], args[1], args[2])
            console.log(Java.vm.getEnv().getStringUtfChars(args[2],
null).readCString())
        },
        onLeave: function (retval) {
            console.log("retval:", retval)
            console.log(Java.vm.getEnv().getStringUtfChars(retval,
null).readCString())
            var newRetval = Java.vm.getEnv().newStringUtf("new retval from
hook_native");
            retval.replace(ptr(newRetval));
        }
    })
}
```

```
libnative_addr is: 0x7a0842f000
string_with_jni_addr is: 0x7a08436194
[Google Pixel::myapplication.example.com.ndk_demo]-> string_with_jni args:
0x7a106cc1c0 0x7ff0b71da4 0x7ff0b71da8
sakura
retval: 0x75
sakura1328
```

这里还写了一个hook env里的GetStringUTFChars的代码，和上面一样，不赘述了。

```
function hook_art(){
    var addr_GetStringUTFChars = null;
    //console.log( JSON.stringify(Process.enumerateModules()));
    var symbols = Process.findModuleByName("libart.so").enumerateSymbols();
    for(var i = 0;i<symbols.length;i++){
        var symbol = symbols[i].name;
        if((symbol.indexOf("CheckJNI")==-1)&&(symbol.indexOf("JNI")>=0)){
            if(symbol.indexOf("GetStringUTFChars")>=0){
                console.log(symbols[i].name);
                console.log(symbols[i].address);
                addr_GetStringUTFChars = symbols[i].address;
            }
        }
    }
    console.log("addr_GetStringUTFChars:", addr_GetStringUTFChars);
    Java.perform(function (){
        Interceptor.attach(addr_GetStringUTFChars, {
            onEnter: function (args) {
                console.log("addr_GetStringUTFChars OnEnter
args[0],args[1]",args[0],args[1]);
                //console.log(hexdump(args[0].readPointer()));

 //console.log(Java.vm.tryGetEnv().getStringUtfChars(args[0]).readCString());
            }, onLeave: function (retval) {
                console.log("addr_GetStringUTFChars
OnLeave",ptr(retval).readCString());
            }
        })
    })
}
```

JNI函数参数、返回值打印和替换

- libc函数符号hook
- libc函数参数、返回值打印和替换 hook libc的也和上面的完全一样，也不赘述了。 所以看到这里，究其本质就是找到导出符号和它所在的so基地址了。

```
function hook_libc(){
    var pthread_create_addr = null;
    var symbols = Process.findModuleByName("libc.so").enumerateSymbols();
    for(var i = 0;i<symbols.length;i++){
        var symbol = symbols[i].name;

        if(symbol.indexOf("pthread_create")>=0){
            //console.log(symbols[i].name);
            //console.log(symbols[i].address);
            pthread_create_addr = symbols[i].address;
        }

    }
    console.log("pthread_create_addr,",pthread_create_addr);
    Interceptor.attach(pthread_create_addr,{
        onEnter:function(args){
            console.log("pthread_create_addr
args[0],args[1],args[2],args[3]:",args[0],args[1],args[2],args[3]);

        },onLeave:function(retval){
            console.log("retval is:",retval)
        }
    })
}
```

Frida native hook : JNI_Onload/动态注册/inline_hook/native层调用栈打印

https://github.com/android/ndk-samples

JNI_Onload/动态注册原理

- JNI_Onload/动态注册/Frida hook RegisterNative
  - JNI与动态注册
  - native 方法的动态注册
  - Frida hook art

详细的内容参见我写的文章，这里只给出栗子。

```
Log.d(TAG,stringFromJNI2());
public native String stringFromJNI2();
```

```
JNIEXPORT jstring JNICALL stringFromJNI2(
        JNIEnv *env,
        jclass clazz) {
    jclass testClass = env-
>FindClass("myapplication/example/com/ndk_demo/Test");
```

```
    jfieldID publicStaticField = env->GetStaticFieldID(testClass,
"publicStaticField",
                                                       "Ljava/lang/String;");
    jstring publicStaticFieldValue = (jstring) env-
>GetStaticObjectField(testClass,

publicStaticField);
    const char *value_ptr = env->GetStringUTFChars(publicStaticFieldValue,
NULL);
    LOGD("now content is %s", value_ptr);
    std::string hello = "Hello from C++ stringFromJNI2";
    return env->NewStringUTF(hello.c_str());
}
...
JNIEXPORT jint JNI_OnLoad(JavaVM *vm, void *reserved) {
    JNIEnv *env;
    vm->GetEnv((void **) &env, JNI_VERSION_1_6);
    JNINativeMethod methods[] = {
            {"stringFromJNI2", "()Ljava/lang/String;", (void *)
stringFromJNI2},
    };
    env->RegisterNatives(env-
>FindClass("myapplication/example/com/ndk_demo/MainActivity"), methods,
                         1);
    return JNI_VERSION_1_6;
}
```

Frida hook RegisterNative

使用下面这个脚本来打印出RegisterNatives的参数，这里需要注意的是使用了enumerateSymbolsSync,它是enumerateSymbols的同步版本。 另外和我们之前通过`Java.vm.tryGetEnv().getStringUtfChars`来调用env里的方法不同。 这里则是通过将之前找到的getStringUtfChars函数地址和参数信息封装起来，直接调用，具体的原理我没有深入分析，先记住用法。 原理其实是一样的，都是**根据符号找到地址，然后hook符号地址，然后打印参数**。

```
declare const NativeFunction: NativeFunctionConstructor;

interface NativeFunctionConstructor {
    new(address: NativePointerValue, retType: NativeType, argTypes:
NativeType[], abiOrOptions?: NativeABI | NativeFunctionOptions):
NativeFunction;
    readonly prototype: NativeFunction;
}
...
var funcGetStringUTFChars = new NativeFunction(addrGetStringUTFChars,
"pointer", ["pointer", "pointer", "pointer"]);
```

```
var ishook_libart = false;
```

```
function hook_libart() {
    if (ishook_libart === true) {
        return;
    }
    var symbols = Module.enumerateSymbolsSync("libart.so");
    var addrGetStringUTFChars = null;
    var addrNewStringUTF = null;
    var addrFindClass = null;
    var addrGetMethodID = null;
    var addrGetStaticMethodID = null;
    var addrGetFieldID = null;
    var addrGetStaticFieldID = null;
    var addrRegisterNatives = null;
    var addrAllocObject = null;
    var addrCallObjectMethod = null;
    var addrGetObjectClass = null;
    var addrReleaseStringUTFChars = null;
    for (var i = 0; i < symbols.length; i++) {
        var symbol = symbols[i];
        if (symbol.name ==
"_ZN3art3JNI17GetStringUTFCharsEP7_JNIEnvP8_jstringPh") {
            addrGetStringUTFChars = symbol.address;
            console.log("GetStringUTFChars is at ", symbol.address,
symbol.name);
        } else if (symbol.name == "_ZN3art3JNI12NewStringUTFEP7_JNIEnvPKc") {
            addrNewStringUTF = symbol.address;
            console.log("NewStringUTF is at ", symbol.address, symbol.name);
        } else if (symbol.name == "_ZN3art3JNI9FindClassEP7_JNIEnvPKc") {
            addrFindClass = symbol.address;
            console.log("FindClass is at ", symbol.address, symbol.name);
        } else if (symbol.name ==
"_ZN3art3JNI11GetMethodIDEP7_JNIEnvP7_jclassPKcS6_") {
            addrGetMethodID = symbol.address;
            console.log("GetMethodID is at ", symbol.address, symbol.name);
        } else if (symbol.name ==
"_ZN3art3JNI17GetStaticMethodIDEP7_JNIEnvP7_jclassPKcS6_") {
            addrGetStaticMethodID = symbol.address;
            console.log("GetStaticMethodID is at ", symbol.address,
symbol.name);
        } else if (symbol.name ==
"_ZN3art3JNI10GetFieldIDEP7_JNIEnvP7_jclassPKcS6_") {
            addrGetFieldID = symbol.address;
            console.log("GetFieldID is at ", symbol.address, symbol.name);
        } else if (symbol.name ==
"_ZN3art3JNI16GetStaticFieldIDEP7_JNIEnvP7_jclassPKcS6_") {
            addrGetStaticFieldID = symbol.address;
            console.log("GetStaticFieldID is at ", symbol.address,
symbol.name);
```

```
        } else if (symbol.name ==
"_ZN3art3JNI15RegisterNativesEP7_JNIEnvP7_jclassPK15JNINativeMethodi") {
            addrRegisterNatives = symbol.address;
            console.log("RegisterNatives is at ", symbol.address,
symbol.name);
        } else if
(symbol.name.indexOf("_ZN3art3JNI11AllocObjectEP7_JNIEnvP7_jclass") >= 0) {
            addrAllocObject = symbol.address;
            console.log("AllocObject is at ", symbol.address, symbol.name);
        }  else if
(symbol.name.indexOf("_ZN3art3JNI16CallObjectMethodEP7_JNIEnvP8_jobjectP10_jme
thodIDz") >= 0) {
            addrCallObjectMethod = symbol.address;
            console.log("CallObjectMethod is at ", symbol.address,
symbol.name);
        } else if
(symbol.name.indexOf("_ZN3art3JNI14GetObjectClassEP7_JNIEnvP8_jobject") >= 0)
{
            addrGetObjectClass = symbol.address;
            console.log("GetObjectClass is at ", symbol.address, symbol.name);
        } else if
(symbol.name.indexOf("_ZN3art3JNI21ReleaseStringUTFCharsEP7_JNIEnvP8_jstringPK
c") >= 0) {
            addrReleaseStringUTFChars = symbol.address;
            console.log("ReleaseStringUTFChars is at ", symbol.address,
symbol.name);
        }
    }

    if (addrRegisterNatives != null) {
        Interceptor.attach(addrRegisterNatives, {
            onEnter: function (args) {
                console.log("[RegisterNatives] method_count:", args[3]);
                var env = args[0];
                var java_class = args[1];

                var funcAllocObject = new NativeFunction(addrAllocObject,
"pointer", ["pointer", "pointer"]);
                var funcGetMethodID = new NativeFunction(addrGetMethodID,
"pointer", ["pointer", "pointer", "pointer", "pointer"]);
                var funcCallObjectMethod = new
NativeFunction(addrCallObjectMethod, "pointer", ["pointer", "pointer",
"pointer"]);
                var funcGetObjectClass = new
NativeFunction(addrGetObjectClass, "pointer", ["pointer", "pointer"]);
                var funcGetStringUTFChars = new
NativeFunction(addrGetStringUTFChars, "pointer", ["pointer", "pointer",
"pointer"]);
```

```
                var funcReleaseStringUTFChars = new
NativeFunction(addrReleaseStringUTFChars, "void", ["pointer", "pointer",
"pointer"]);

                var clz_obj = funcAllocObject(env, java_class);
                var mid_getClass = funcGetMethodID(env, java_class,
Memory.allocUtf8String("getClass"), Memory.allocUtf8String("
()Ljava/lang/Class;"));
                var clz_obj2 = funcCallObjectMethod(env, clz_obj,
mid_getClass);
                var cls = funcGetObjectClass(env, clz_obj2);
                var mid_getName = funcGetMethodID(env, cls,
Memory.allocUtf8String("getName"), Memory.allocUtf8String("
()Ljava/lang/String;"));
                var name_jstring = funcCallObjectMethod(env, clz_obj2,
mid_getName);
                var name_pchar = funcGetStringUTFChars(env, name_jstring,
ptr(0));
                var class_name = ptr(name_pchar).readCString();
                funcReleaseStringUTFChars(env, name_jstring, name_pchar);

                //console.log(class_name);

                var methods_ptr = ptr(args[2]);

                var method_count = parseInt(args[3]);
                for (var i = 0; i < method_count; i++) {
                    var name_ptr = Memory.readPointer(methods_ptr.add(i *
Process.pointerSize * 3));
                    var sig_ptr = Memory.readPointer(methods_ptr.add(i *
Process.pointerSize * 3 + Process.pointerSize));
                    var fnPtr_ptr = Memory.readPointer(methods_ptr.add(i *
Process.pointerSize * 3 + Process.pointerSize * 2));

                    var name = Memory.readCString(name_ptr);
                    var sig = Memory.readCString(sig_ptr);
                    var find_module = Process.findModuleByAddress(fnPtr_ptr);
                    console.log("[RegisterNatives] java_class:", class_name,
"name:", name, "sig:", sig, "fnPtr:", fnPtr_ptr, "module_name:",
find_module.name, "module_base:", find_module.base, "offset:",
ptr(fnPtr_ptr).sub(find_module.base));

                }
            },
            onLeave: function (retval) { }
        });
    }

    ishook_libart = true;
```

```
}

hook_libart();
```

结果很明显的打印了出来，包括动态注册的函数的名字，函数签名，加载地址和在so里的偏移量，

```
[RegisterNatives] java_class: myapplication.example.com.ndk_demo.MainActivity
name: stringFromJNI2 sig: ()Ljava/lang/String; fnPtr: 0x79f8698484
module_name: libnative-lib.so module_base: 0x79f8691000 offset: 0x7484
```

最后测试一下yang开源的一个hook art的脚本，很有意思，trace出了非常多的需要的信息。

```
frida -U --no-pause -f package_name -l hook_art.js
...
[FindClass] name:myapplication/example/com/ndk_demo/Test
[GetStaticFieldID] name:publicStaticField, sig:Ljava/lang/String;
[GetStringUTFChars] result:i am a publicStaticField
[NewStringUTF] bytes:Hello from C++ stringFromJNI2
[GetStringUTFChars] result:sakura
```

native层调用栈打印

直接使用frida提供的接口打印栈回溯。

```
Interceptor.attach(f, {
  onEnter: function (args) {
    console.log('RegisterNatives called from:\n' +
        Thread.backtrace(this.context, Backtracer.ACCURATE)
        .map(DebugSymbol.fromAddress).join('\n') + '\n');
  }
});
```

效果如下,我加到了hook registerNative的地方。

```
[Google Pixel::myapplication.example.com.ndk_demo]-> RegisterNatives called
from:
0x7a100be03c libart.so!0xe103c
0x7a100be038 libart.so!0xe1038
0x79f85699a0 libnative-
lib.so!_ZN7_JNIEnv15RegisterNativesEP7_jclassPK15JNINativeMethodi+0x44
0x79f85698e0 libnative-lib.so!JNI_OnLoad+0x90
0x7a102b9fd4
libart.so!_ZN3art9JavaVMExt17LoadNativeLibraryEP7_JNIEnvRKNSt3__112basic_strin
gIcNS3_11char_traitsIcEENS3_9allocatorIcEEEEP8_jobjectP8_jstringPS9_+0x638
0x7a08e3820c libopenjdkjvm.so!JVM_NativeLoad+0x110
0x70b921c4 boot.oat!oatexec+0xa81c4
```

主动调用去进行方法参数替换

使用`Interceptor.replace`，不赘述。主要目的还是为了改掉函数原本的执行行为，而不是仅仅打印一些信息。

inline hook

inline hook简单理解就是不是hook函数开始执行的地方，而是hook函数中间执行的指令 整体来说没什么区别，就是把找函数符号地址改成从so里找到偏移，然后加到so基地址上就行,注意一下它的attach的callback。

```
/**
 * Callback to invoke when an instruction is about to be executed.
 */
type InstructionProbeCallback = (this: InvocationContext, args:
InvocationArguments) => void;
type InvocationContext = PortableInvocationContext | WindowsInvocationContext
| UnixInvocationContext;

interface PortableInvocationContext {
    /**
     * Return address.
     */
    returnAddress: NativePointer;

    /**
     * CPU registers. You may also update register values by assigning to
these keys.
     */
    context: CpuContext;

    /**
     * OS thread ID.
     */
    threadId: ThreadId;

    /**
     * Call depth of relative to other invocations.
     */
    depth: number;

    /**
     * User-defined invocation data. Useful if you want to read an argument in
`onEnter` and act on it in `onLeave`.
     */
    [x: string]: any;
}
...
...
```

```
interface Arm64CpuContext extends PortableCpuContext {
    x0: NativePointer;
    x1: NativePointer;
    x2: NativePointer;
    x3: NativePointer;
    x4: NativePointer;
    x5: NativePointer;
    x6: NativePointer;
    x7: NativePointer;
    x8: NativePointer;
    x9: NativePointer;
    x10: NativePointer;
    x11: NativePointer;
    x12: NativePointer;
    x13: NativePointer;
    x14: NativePointer;
    x15: NativePointer;
    x16: NativePointer;
    x17: NativePointer;
    x18: NativePointer;
    x19: NativePointer;
    x20: NativePointer;
    x21: NativePointer;
    x22: NativePointer;
    x23: NativePointer;
    x24: NativePointer;
    x25: NativePointer;
    x26: NativePointer;
    x27: NativePointer;
    x28: NativePointer;

    fp: NativePointer;
    lr: NativePointer;
}
```

我的so是自己编译的，具体的汇编代码如下,总之这里很明显在775C时，x0里保存的是一个指向"sakura"这个字符串的指针。(其实我也不是很看得懂arm64了已经，就随便hook了一下) 所以hook这个指令，然后`Memory.readCString(this.context.x0);`打印出来，结果如下

```
.text:000000000000772C ; __unwind {
.text:000000000000772C                 SUB             SP, SP, #0x40
.text:0000000000007730                 STP             X29, X30,
[SP,#0x30+var_s0]
.text:0000000000007734                 ADD             X29, SP, #0x30
.text:0000000000007738 ; 6:   v6 = a1;
.text:0000000000007738                 MOV             X8, XZR
.text:000000000000773C                 STUR            X0, [X29,#var_8]
.text:0000000000007740 ; 7:   v5 = a3;
.text:0000000000007740                 STUR            X1, [X29,#var_10]
```

```
.text:0000000000007744                 STR             X2, [SP,#0x30+var_18]
.text:0000000000007748 ; 8:   v4 = (const char
*)_JNIEnv::GetStringUTFChars(a1, a3, 0LL);
.text:0000000000007748                 LDUR            X0, [X29,#var_8]
.text:000000000000774C                 LDR             X1, [SP,#0x30+var_18]
.text:0000000000007750                 MOV             X2, X8
.text:0000000000007754                 BL
._ZN7_JNIEnv17GetStringUTFCharsEP8_jstringPh ;
_JNIEnv::GetStringUTFChars(_jstring *,uchar *)
.text:0000000000007758                 STR             X0, [SP,#0x30+var_20]
.text:000000000000775C ; 9:   if ( (signed int)_JNIEnv::GetStringUTFLength(v6,
v5) > 0 )
.text:000000000000775C                 LDUR            X0, [X29,#var_8]
.text:0000000000007760                 LDR             X1, [SP,#0x30+var_18]
```

```
function inline_hook() {
    var libnative_lib_addr = Module.findBaseAddress("libnative-lib.so");
    if (libnative_lib_addr) {
        console.log("libnative_lib_addr:", libnative_lib_addr);
        var addr_775C = libnative_lib_addr.add(0x775C);
        console.log("addr_775C:", addr_775C);

        Java.perform(function () {
            Interceptor.attach(addr_775C, {
                onEnter: function (args) {
                    var name = this.context.x0.readCString()
                    console.log("addr_775C OnEnter :", this.returnAddress,
name);
                },
                onLeave: function (retval) {
                     console.log("retval is :", retval)
                }
            })
        })
    }
}
setImmediate(inline_hook())
```

```
Attaching...
libnative_lib_addr: 0x79fabe0000
addr_775C: 0x79fabe775c
TypeError: cannot read property 'apply' of undefined
    at [anon] (../../../frida-gum/bindings/gumjs/duktape.c:56618)
    at frida/runtime/core.js:55
[Google Pixel::myapplication.example.com.ndk_demo]-> addr_775C OnEnter :
0x79fabe7758 sakura
addr_775C OnEnter : 0x79fabe7758 sakura
```

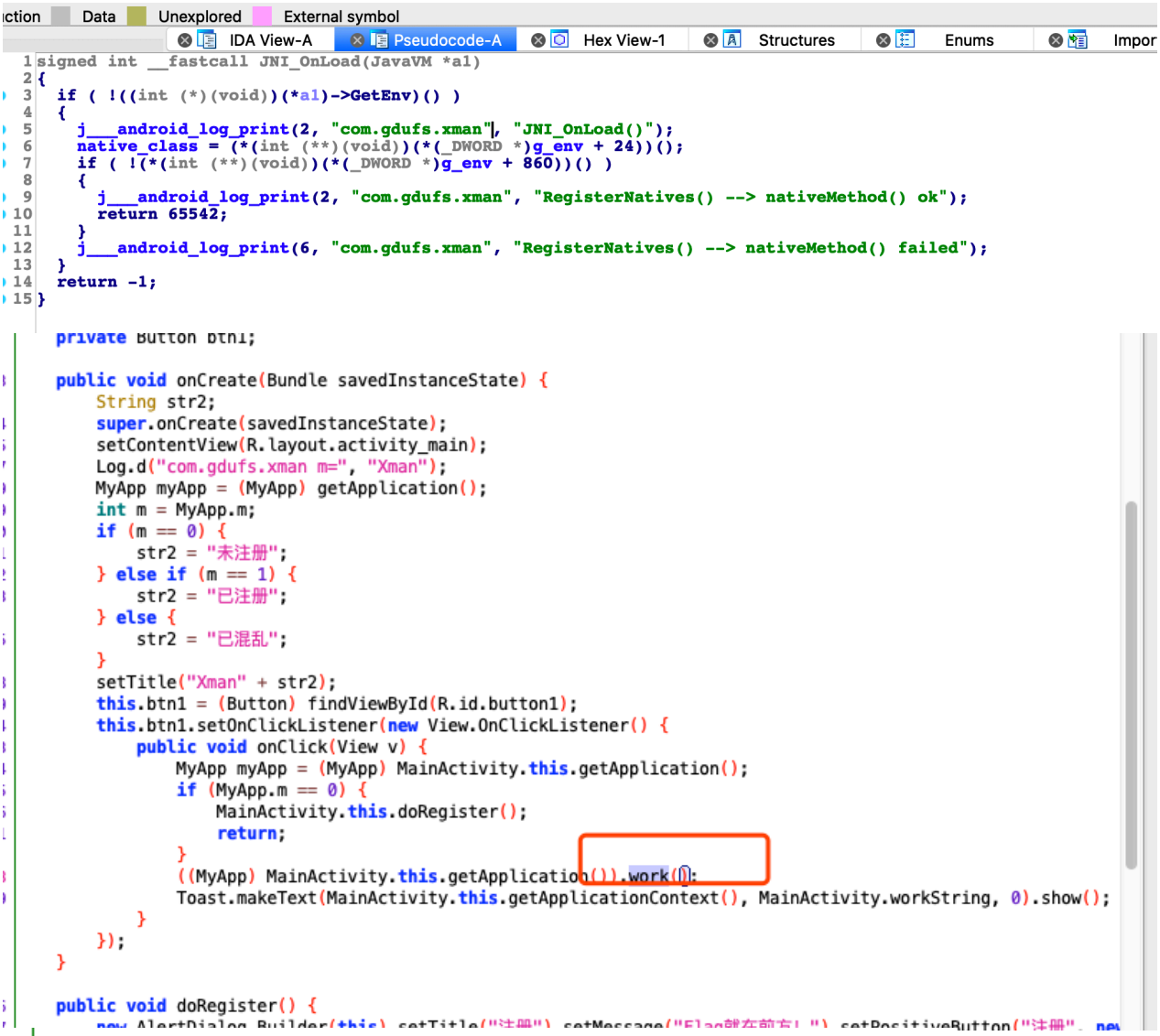
到这里已经可以总结一下我目前的学习了，需要补充一些frida api的学习，比如NativePointr里居然有个readCString，这些API是需要再看看的。

Frida native hook : Frida hook native app实战

- 破解Frida全端口检测的native层反调试
  - hook libc的pthread_create函数
- 破解TracePid的native反调试
  - target: https://gtoad.github.io/2017/06/25/Android-Anti-Debug/
  - solve : hook libc的fgets函数
- native层修改参数、返回值
- 静态分析`JNI_Onload`
- 动态trace主动注册 & IDA溯源
- 动态trace JNI、libc函数 & IDA溯源
- native层主动调用、打调用栈
- 主动调用libc读写文件

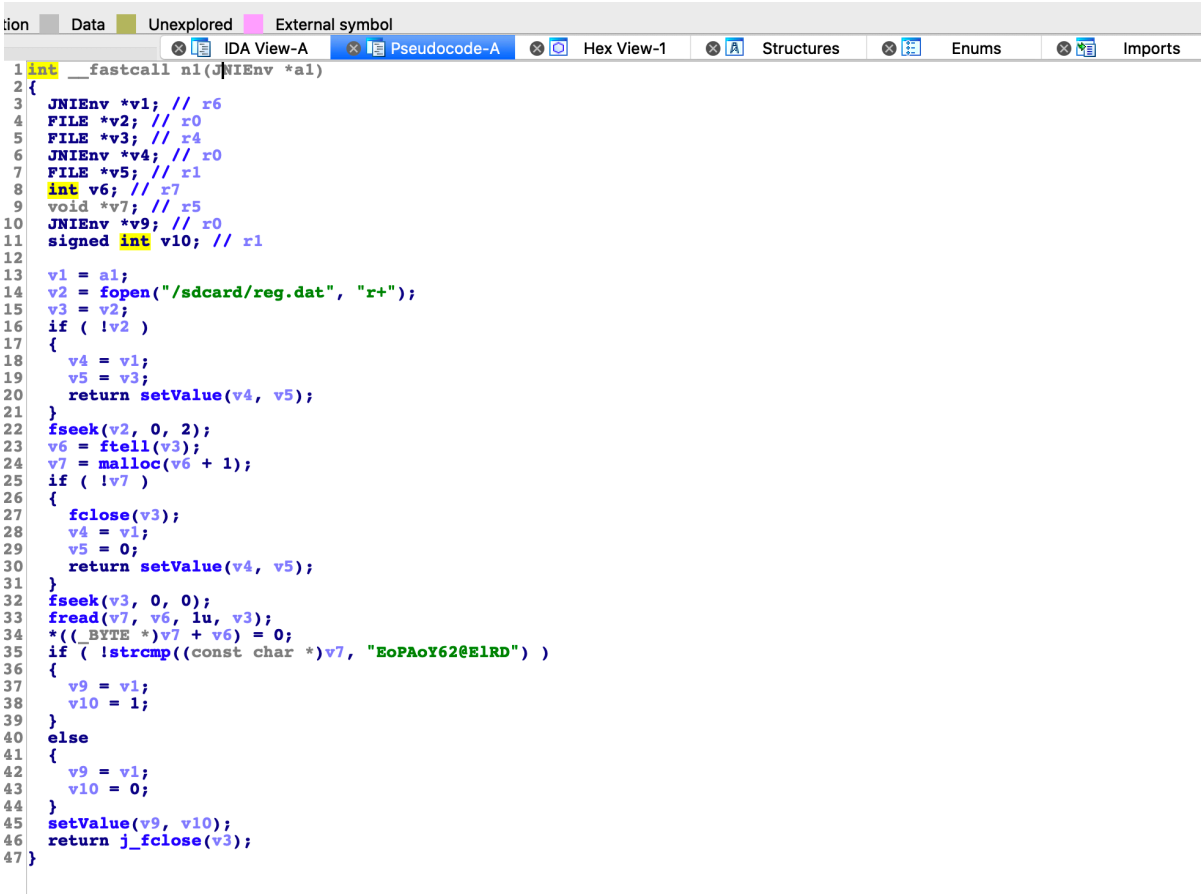
看下logcat

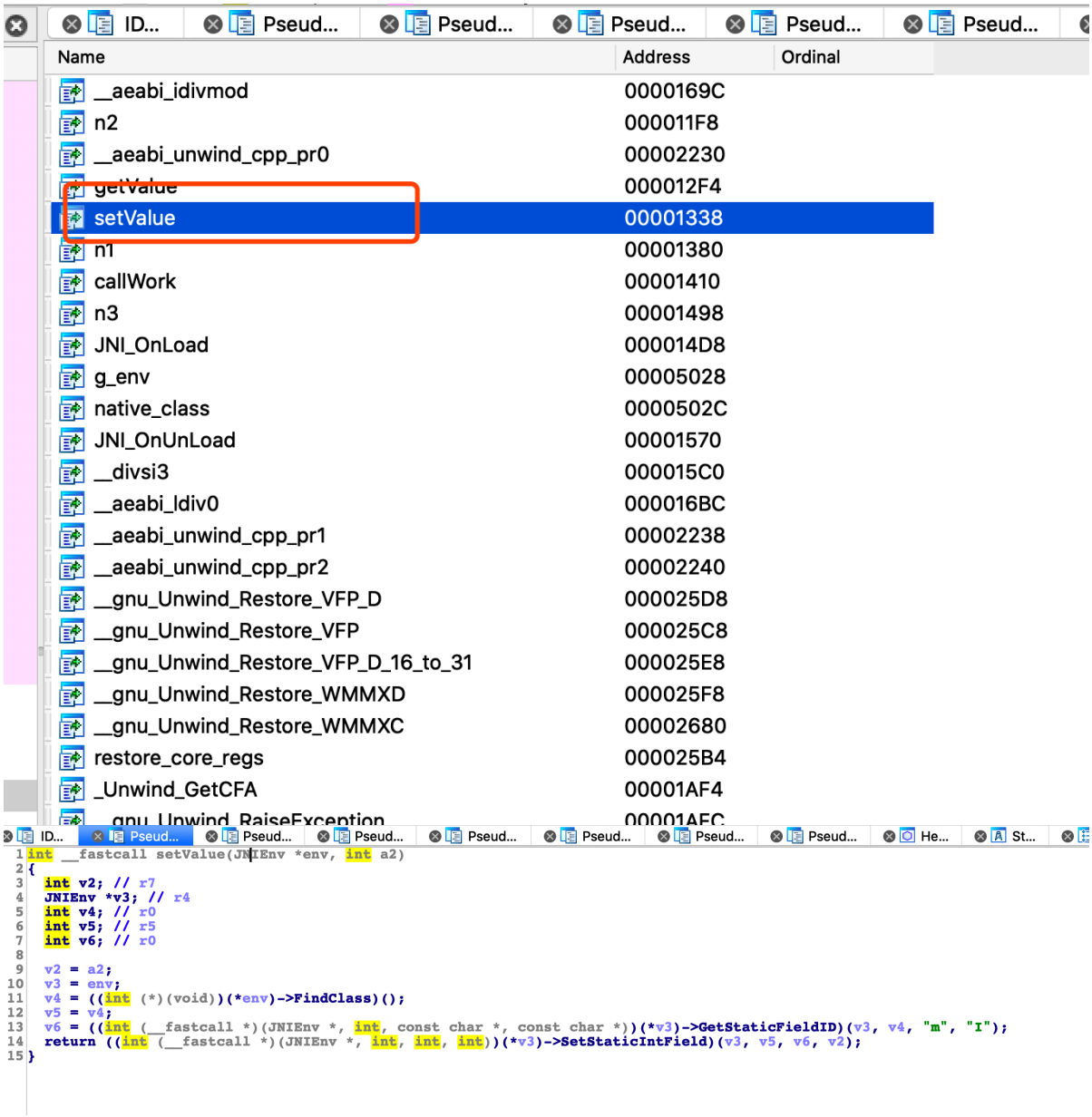
```
n/u0a128 for activity com.gdufs.xman/.MainActivity
12-28 05:53:26.898 26615 26615 V com.gdufs.xman: JNI_OnLoad()
12-28 05:53:26.898 26615 26615 V com.gdufs.xman: RegisterNatives() -->
nativeMethod() ok
12-28 05:53:26.898 26615 26615 D com.gdufs.xman m=: 0
12-28 05:53:26.980 26615 26615 D com.gdufs.xman m=: Xman
```


ction
Data
Unexplored
External symbol
IDA View-A
Pseudocode-A
Hex View-1
Structures
Enums
Impor
1 signed int __fastcall JNI_OnLoad(JavaVM *a1)
2 {
3 if (!((int (*)(void))(*a1)->GetEnv)())
4 {
5 j___android_log_print(2, "com.gdufs.xman", "JNI_OnLoad()");
6 native_class = (*(int (**)(void))(*(_DWORD *)g_env + 24))();
7 if (!(*(int (**)(void))(*(_DWORD *)g_env + 860))())
8 {
9 j___android_log_print(2, "com.gdufs.xman", "RegisterNatives() --> nativeMethod() ok");
10 return 65542;
11 }
12 j___android_log_print(6, "com.gdufs.xman", "RegisterNatives() --> nativeMethod() failed");
13 }
14 return -1;
15 }
private Button btn1;
public void onCreate(Bundle savedInstanceState) {
String str2;
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
Log.d("com.gdufs.xman m=", "Xman");
MyApp myApp = (MyApp) getApplication();
int m = MyApp.m;
if (m == 0) {
str2 = "未注册";
} else if (m == 1) {
str2 = "已注册";
} else {
str2 = "已混乱";
}
setTitle("Xman" + str2);
this.btn1 = (Button) findViewById(R.id.button1);
this.btn1.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
MyApp myApp = (MyApp) MainActivity.this.getApplication();
if (MyApp.m == 0) {
MainActivity.this.doRegister();
return;
}
((MyApp) MainActivity.this.getApplication()).work();
Toast.makeText(MainActivity.this.getApplicationContext(), MainActivity.workString, 0).show();
}
});
}
public void doRegister() {
6 public class MyApp extends Application {
public static int m = 0;
public native void initSN();
public native void saveSN(String str);
public native void work();
static {
13 System.loadLibrary("myjni");
}
16 public void onCreate() {
17 initSN();
18 Log.d("com.gdufs.xman m=", String.valueOf(m));
19 super.onCreate();
}
}

```
sakura@sakuradeMacBook-Pro:~/gitsource/frida-agent-example/agent$ frida -U --
no-pause -f com.gdufs.xman -l hook_reg.js
...
[Google Pixel::com.gdufs.xman]-> [RegisterNatives] method_count: 0x3
[RegisterNatives] java_class: com.gdufs.xman.MyApp name: initSN sig: ()V
fnPtr: 0xd4ddf3b1 module_name: libmyjni.so module_base: 0xd4dde000 offset:
0x13b1
[RegisterNatives] java_class: com.gdufs.xman.MyApp name: saveSN sig:
(Ljava/lang/String;)V fnPtr: 0xd4ddf1f9 module_name: libmyjni.so module_base:
0xd4dde000 offset: 0x11f9
[RegisterNatives] java_class: com.gdufs.xman.MyApp name: work sig: ()V fnPtr:
0xd4ddf4cd module_name: libmyjni.so module_base: 0xd4dde000 offset: 0x14cd
```

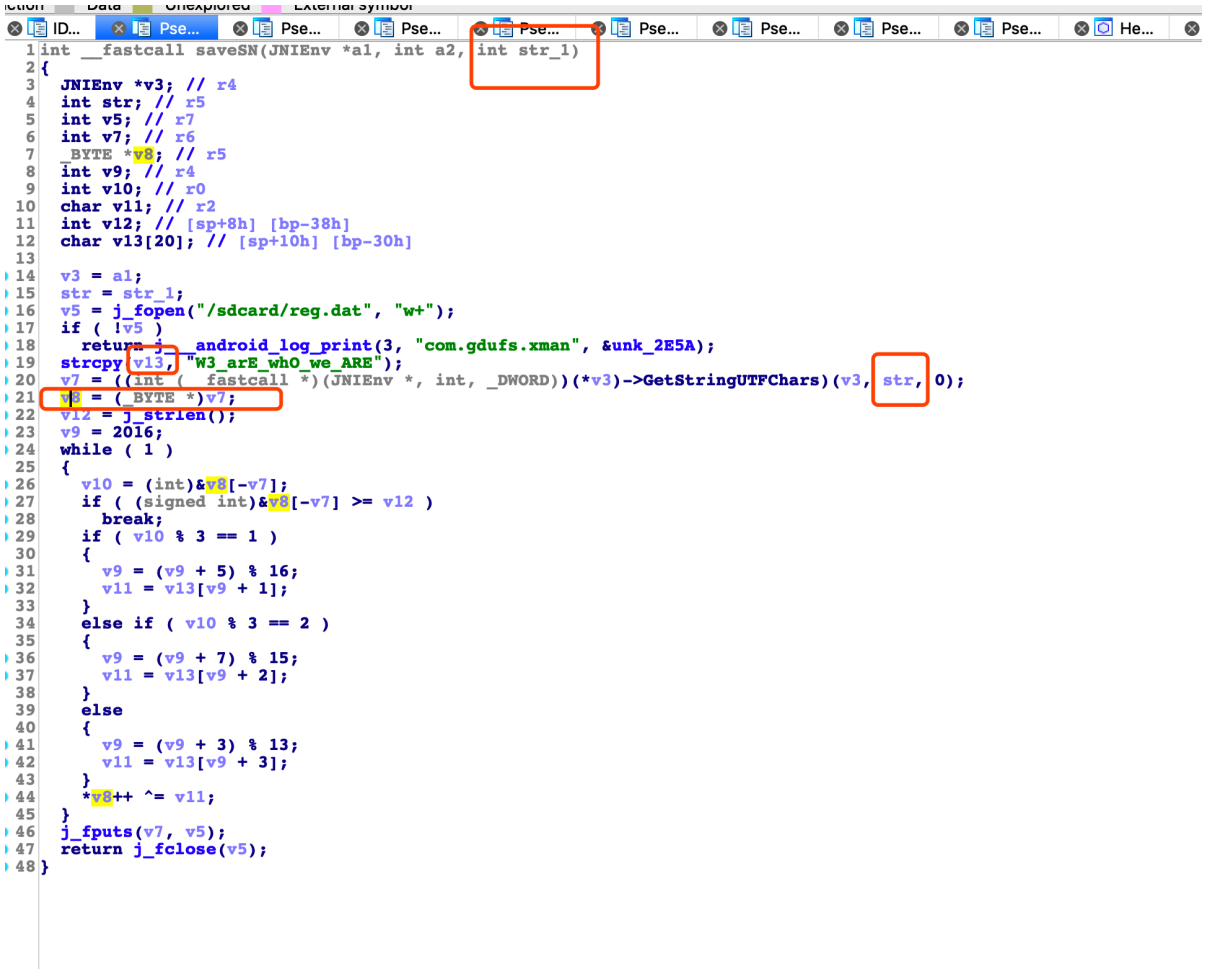
- initSN 感觉意思应该是从`/sdcard/reg.dat`里读一个值，然后和`EoPAoY62@ElRD`进行比较。 最后setValue，从导出函数看一下，最后推测第一个参数应该是JNIEnv *env，然后就看到了给字段m赋值。






```
int __fastcall setValue(JNIEnv *env, int a2)
{
  int v2; // r7
  JNIEnv *v3; // r4
  int v4; // r0
  int v5; // r5
  int v6; // r0

  v2 = a2;
  v3 = env;
  v4 = ((int (*)(void))(*env)->FindClass)();
  v5 = v4;
  v6 = ((int (__fastcall *)(JNIEnv *, int, const char *, const char *))(*v3)->GetStaticFieldID)(v3, v4, "m", "I");
  return ((int (__fastcall *)(JNIEnv *, int, int, int))(*v3)->SetStaticIntField)(v3, v5, v6, v2);
}
```

- saveSN 这个看上去就是根据str的值，去变换"W3_arE_whO_we_ARE"字符串，然后写入到`/sdcard/reg.dat`里



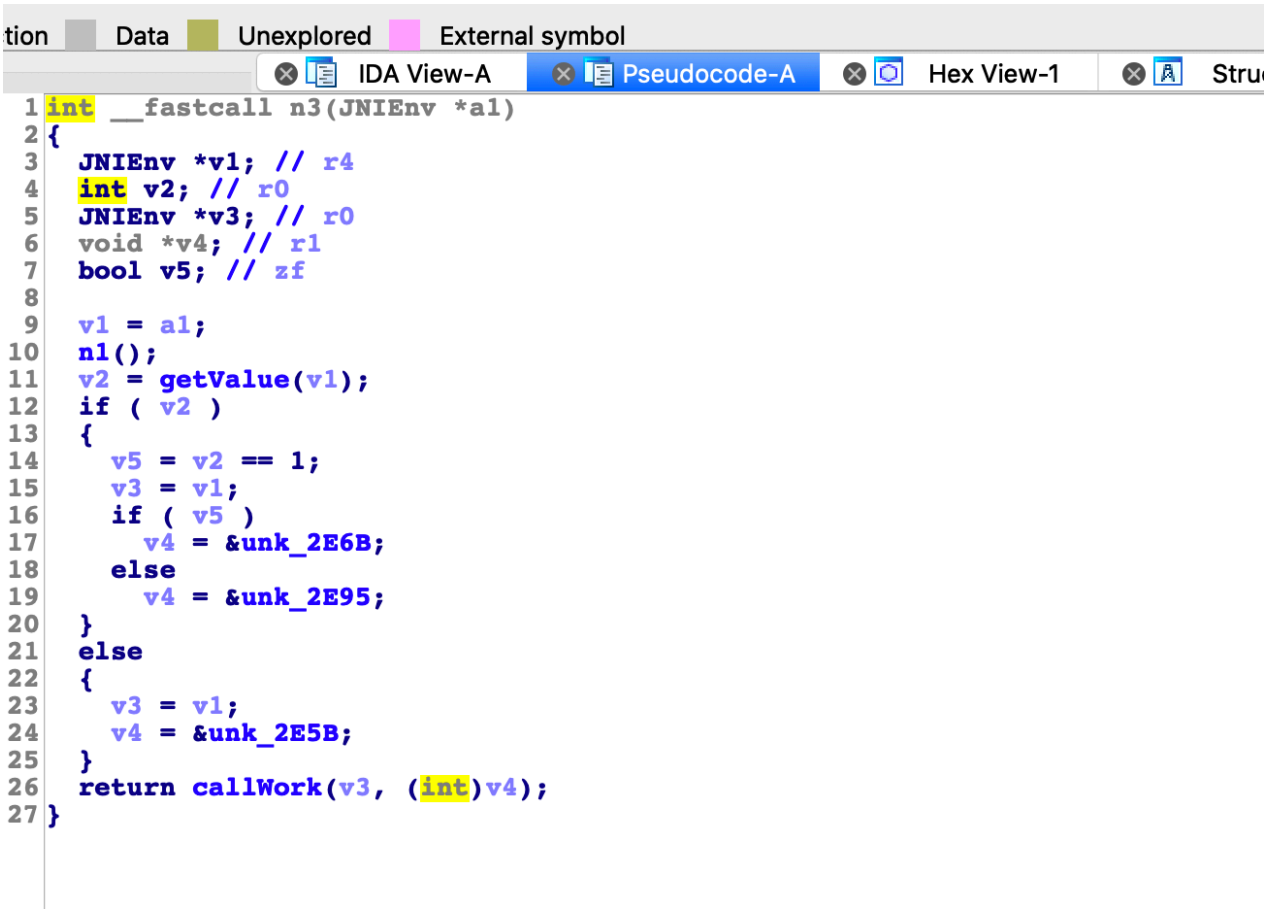
```
int __fastcall saveSN(JNIEnv *a1, int a2, int str_1)
{
  JNIEnv *v3; // r4
  int str; // r5
  int v5; // r7
  int v7; // r6
  _BYTE *v8; // r5
  int v9; // r4
  int v10; // r0
  char v11; // r2
  int v12; // [sp+8h] [bp-38h]
  char v13[20]; // [sp+10h] [bp-30h]

  v3 = a1;
  str = str_1;
  v5 = j_fopen("/sdcard/reg.dat", "w+");
  if ( !v5 )
    return j__android_log_print(3, "com.gdufs.xman", &unk_2E5A);
  strcpy(v13, "W3_arE_whO_we_ARE");
  v7 = ((int (__fastcall *)(JNIEnv *, int, _DWORD))(*v3)->GetStringUTFChars)(v3, str, 0);
  v8 = (_BYTE *)v7;
  v12 = j_strlen();
  v9 = 2016;
  while ( 1 )
  {
    v10 = (int)&v8[-v7];
    if ( (signed int)&v8[-v7] >= v12 )
      break;
    if ( v10 % 3 == 1 )
    {
      v9 = (v9 + 5) % 16;
      v11 = v13[v9 + 1];
    }
    else if ( v10 % 3 == 2 )
    {
      v9 = (v9 + 7) % 15;
      v11 = v13[v9 + 2];
    }
    else
    {
      v9 = (v9 + 3) % 13;
      v11 = v13[v9 + 3];
    }
    *v8++ ^= v11;
  }
  j_fputs(v7, v5);
  return j_fclose(v5);
}
```

结合一下看，只要initSN检查到`/sdcard/reg.dat`里是`EoPAoY62@ElRD`，应该就会给m设置成1。 只要m的值是1，就能走到work()函数的逻辑。

参考frida的file api

```
function main() {
    var file = new File("/sdcard/reg.dat",'w')
    file.write("EoPAoY62@ElRD")
    file.flush()
    file.close()
}
setImmediate(main())
```

这样我们继续看work的逻辑




```
1  int __fastcall n3(JNIEnv *a1)
2  {
3    JNIEnv *v1; // r4
4    int v2; // r0
5    JNIEnv *v3; // r0
6    void *v4; // r1
7    bool v5; // zf
8
9    v1 = a1;
10   n1();
11   v2 = getValue(v1);
12   if ( v2 )
13   {
14     v5 = v2 == 1;
15     v3 = v1;
16     if ( v5 )
17       v4 = &unk_2E6B;
18     else
19       v4 = &unk_2E95;
20   }
21   else
22   {
23     v3 = v1;
24     v4 = &unk_2E5B;
25   }
26   return callWork(v3, (int)v4);
27 }
```

v2是从getValue得到的，看上去就是m字段的值，此时应该是1，一会hook一下看看。


```
1  int __fastcall getValue(JNIEnv *a1)
2  {
3    JNIEnv *v1; // r4
4    int v2; // r0
5    int v3; // r5
6    int v4; // r0
7
8    v1 = a1;
9    v2 = ((int (*)(void))(*a1)->FindClass)();
10   v3 = v2;
11   v4 = ((int (__fastcall *)(JNIEnv *, int, const char *, const char *))(*v1)->GetStaticFieldID)(v1, v2, "m", "I");
12   return ((int (__fastcall *)(JNIEnv *, int, int))(*v1)->GetStaticIntField)(v1, v3, v4);
13 }
```

```
[NewStringUTF] bytes:输入即是flag,格式为xman{……}!
```

callWork里又调用了work函数，死循环了。

那看来看去最后还是回到了initSN，那其实我们看的顺序似乎错了。 理一下逻辑，n2执行完保存到文件，然后n1 check一下，所以最后还是要逆n2的算法，pass。

Frida trace四件套

jni trace : trace jni

https://github.com/chame1eon/jnitrace

```
pip install jnitrace

Requirement already satisfied: frida>=12.5.0 in
/Users/sakura/.pyenv/versions/3.7.7/lib/python3.7/site-packages (from
jnitrace) (12.8.0)
Requirement already satisfied: colorama in
/Users/sakura/.pyenv/versions/3.7.7/lib/python3.7/site-packages (from
jnitrace) (0.4.3)
Collecting hexdump (from jnitrace)
  Downloading
https://files.pythonhosted.org/packages/55/b3/279b1d57fa3681725d0db8820405cdcb
4e62a9239c205e4ceac4391c78e4/hexdump-3.3.zip
Installing collected packages: hexdump, jnitrace
  Running setup.py install for hexdump ... done
  Running setup.py install for jnitrace ... done
Successfully installed hexdump-3.3 jnitrace-3.0.8
```

usage: `jnitrace [options] -l libname target` 默认应该是spawn运行的，

- `-m`来指定是`spawn`还是`attach`
- `-b`指定是`fuzzy`还是`accurate`
- `-i <regex>`指定一个正则表达式来过滤出方法名，例如`-i Get -i RegisterNatives`就会只打印出名字里包含Get或者RegisterNatives的JNI methods。

- `-e <regex>`和`-i`相反，同样通过正则表达式来过滤，但这次会将指定的内容忽略掉。
- `-I <string>`trace导出的方法，jnitrace认为导出的函数应该是从Java端能够直接调用的函数，所以可以包括使用RegisterNatives来注册的函数，例如`-I stringFromJNI -I nativeMethod([B)V`，就包括导出名里有stringFromJNI，以及使用RegisterNames来注册，并带有nativeMethod([B)V签名的函数。
- `-o path/output.json`，导出输出到文件里。
- `-p path/to/script.js`，用于在加载jnitrace脚本之前将指定路径的Frida脚本加载到目标进程中，这可以用于在jnitrace启动之前对抗反调试。
- `-a path/to/script.js`，用于在加载jnitrace脚本之后将指定路径的Frida脚本加载到目标进程中
- `--ignore-env`，不打印所有的JNIEnv函数
- `--ignore-vm`，不打印所有的JavaVM函数

```
sakura@sakuradeMacBook-
Pro:~/Desktop/lab/alpha/tools/android/frida_learn/0620/0620/xman/resources/lib
/armeabi-v7a$ jnitrace -l libmyjni.so com.gdufs.xman
Tracing. Press any key to quit...
Traced library "libmyjni.so" loaded from path "/data/app/com.gdufs.xman-
X0HkzLhbptSc0tjGZ3yQ2g==/lib/arm".

           /* TID 28890 */
    355 ms [+] JavaVM->GetEnv
    355 ms |- JavaVM*          : 0xefe99140
    355 ms |- void**           : 0xda13e028
    355 ms |:     0xeff312a0
    355 ms |- jint             : 65542
    355 ms |= jint             : 0

    355 ms ------------------------Backtrace------------------------
    355 ms |-> 0xda13a51b: JNI_OnLoad+0x12 (libmyjni.so:0xda139000)


           /* TID 28890 */
    529 ms [+] JNIEnv->FindClass
    529 ms |- JNIEnv*          : 0xeff312a0
    529 ms |- char*            : 0xda13bdef
    529 ms |:     com/gdufs/xman/MyApp
    529 ms |= jclass           : 0x81    { com/gdufs/xman/MyApp }

    529 ms ------------------------Backtrace------------------------
    529 ms |-> 0xda13a539: JNI_OnLoad+0x30 (libmyjni.so:0xda139000)


           /* TID 28890 */
    584 ms [+] JNIEnv->RegisterNatives
    584 ms |- JNIEnv*          : 0xeff312a0
    584 ms |- jclass           : 0x81    { com/gdufs/xman/MyApp }
    584 ms |- JNINativeMethod* : 0xda13e004
```

```
584 ms |:     0xda13a3b1 - initSN()V
584 ms |:     0xda13a1f9 - saveSN(Ljava/lang/String;)V
584 ms |:     0xda13a4cd - work()V
584 ms |- jint             : 3
584 ms |= jint             : 0

584 ms ------------------------Backtrace------------------------
584 ms |-> 0xda13a553: JNI_OnLoad+0x4a (libmyjni.so:0xda139000)


       /* TID 28890 */
638 ms [+] JNIEnv->FindClass
638 ms |- JNIEnv*          : 0xeff312a0
638 ms |- char*            : 0xda13bdef
638 ms |:     com/gdufs/xman/MyApp
638 ms |= jclass           : 0x71    { com/gdufs/xman/MyApp }

638 ms ------------------------Backtrace------------------------
638 ms |-> 0xda13a377: setValue+0x12 (libmyjni.so:0xda139000)


       /* TID 28890 */
688 ms [+] JNIEnv->GetStaticFieldID
688 ms |- JNIEnv*          : 0xeff312a0
688 ms |- jclass           : 0x71    { com/gdufs/xman/MyApp }
688 ms |- char*            : 0xda13be04
688 ms |:     m
688 ms |- char*            : 0xda13be06
688 ms |:     I
688 ms |= jfieldID         : 0xf1165004    { m:I }

688 ms ------------------------Backtrace------------------------
688 ms |-> 0xda13a38d: setValue+0x28 (libmyjni.so:0xda139000)
```

strace : trace syscall

https://linuxtools-rst.readthedocs.io/zh_CN/latest/tool/strace.html

frida-trace : trace libc(or more)

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

Usage:`frida-trace [options] target`

```
frida-trace -U -i "strcmp" -f com.gdufs.xman
...
  5634 ms  strcmp(s1="fi", s2="es-US")
  5635 ms  strcmp(s1="da", s2="es-US")
  5635 ms  strcmp(s1="es", s2="es-US")
  5635 ms  strcmp(s1="eu-ES", s2="es-US")
  5635 ms  strcmp(s1="et-EE", s2="es-US")
  5635 ms  strcmp(s1="et-EE", s2="es-US")
```

- art trace: hook artmethod

hook_artmethod : trace java函数调用

https://github.com/lasting-yang/frida_hook_libart/blob/master/hook_artmethod.js

修改AOSP源码打印

改aosp源码trace信息

Frida native hook : init_array开发和自动化逆向

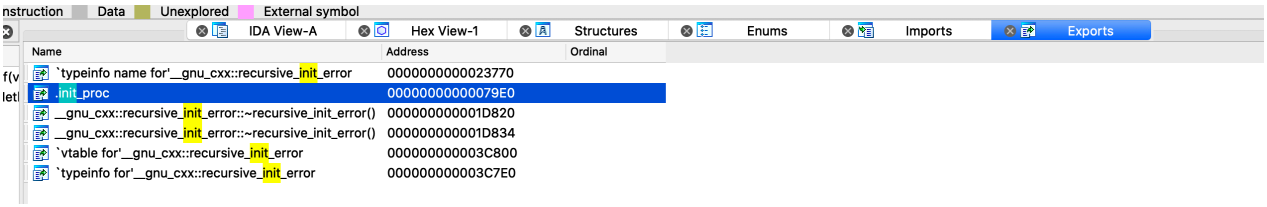
init_array原理

常见的保护都会在init_array里面做，关于其原理，主要阅读以下文章即可。

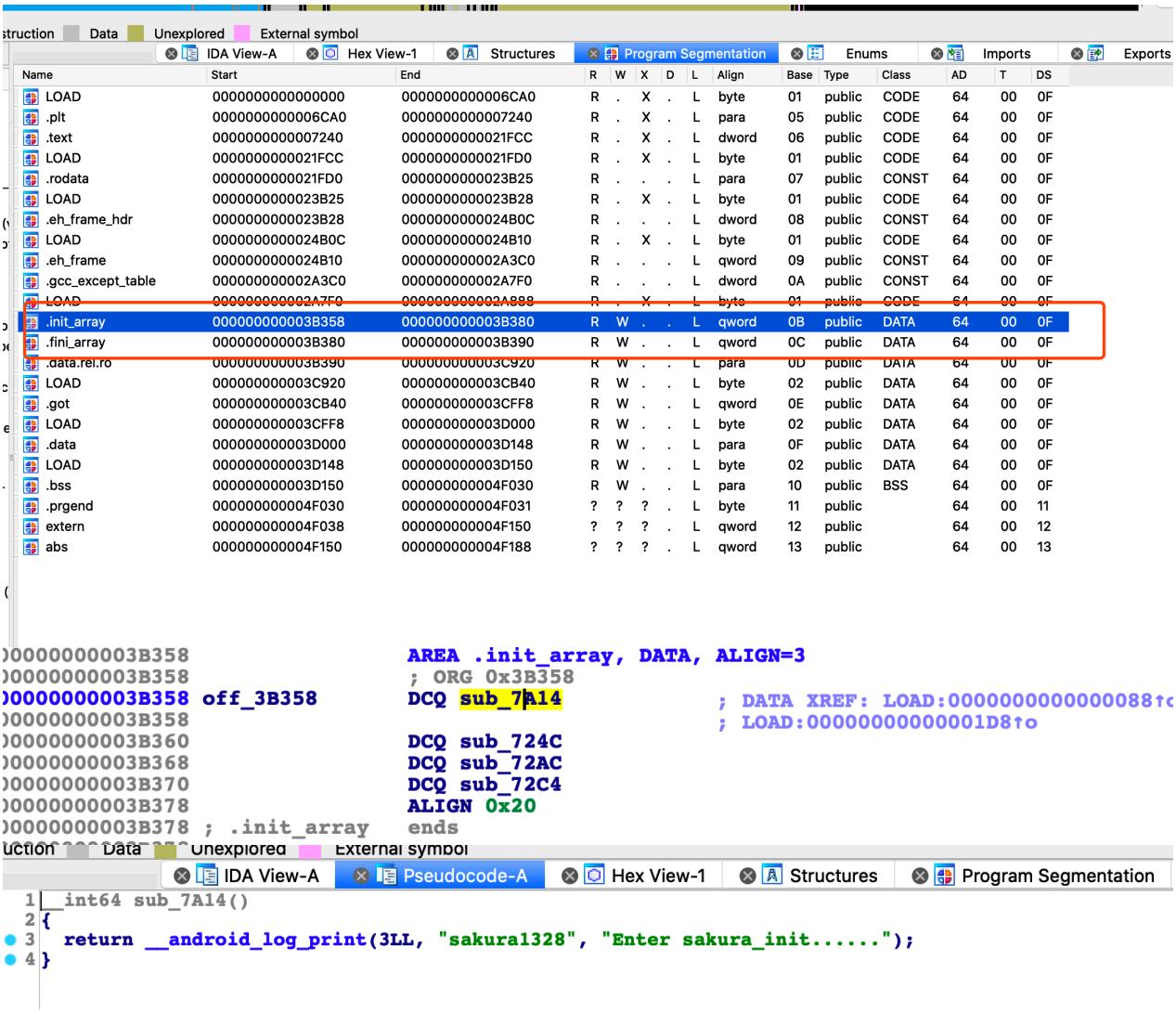
- IDA调试android so的.init_array数组
- Android NDK中.init段和.init_array段函数的定义方式
- Linker学习笔记

IDA静态分析init_array

```
// 编译生成后在.init段 [名字不可更改]
extern "C" void _init(void) {
    LOGD("Enter init......");
}

// 编译生成后在.init_array段 [名字可以更改]
__attribute__((__constructor__)) static void sakura_init() {
    LOGD("Enter sakura_init......");
}
...
...
2016-12-29 16:51:23.017 5160-5160/com.example.ndk_demo D/sakura1328: Enter
init......
2016-12-29 16:51:23.017 5160-5160/com.example.ndk_demo D/sakura1328: Enter
sakura_init......
```



IDA快捷键`shift+F7`找到segment，然后就可以找到`.init_array`段，然后就可以找到里面保存的函数地址。



IDA动态调试so

- 打开要调试的apk，找到入口

```
sakura@sakuradeMacBook-Pro:~/.gradle/caches$ adb shell dumpsys activity top |
grep TASK
TASK com.android.systemui id=29 userId=0
TASK null id=26 userId=0
TASK com.example.ndk_demo id=161 userId=0
```

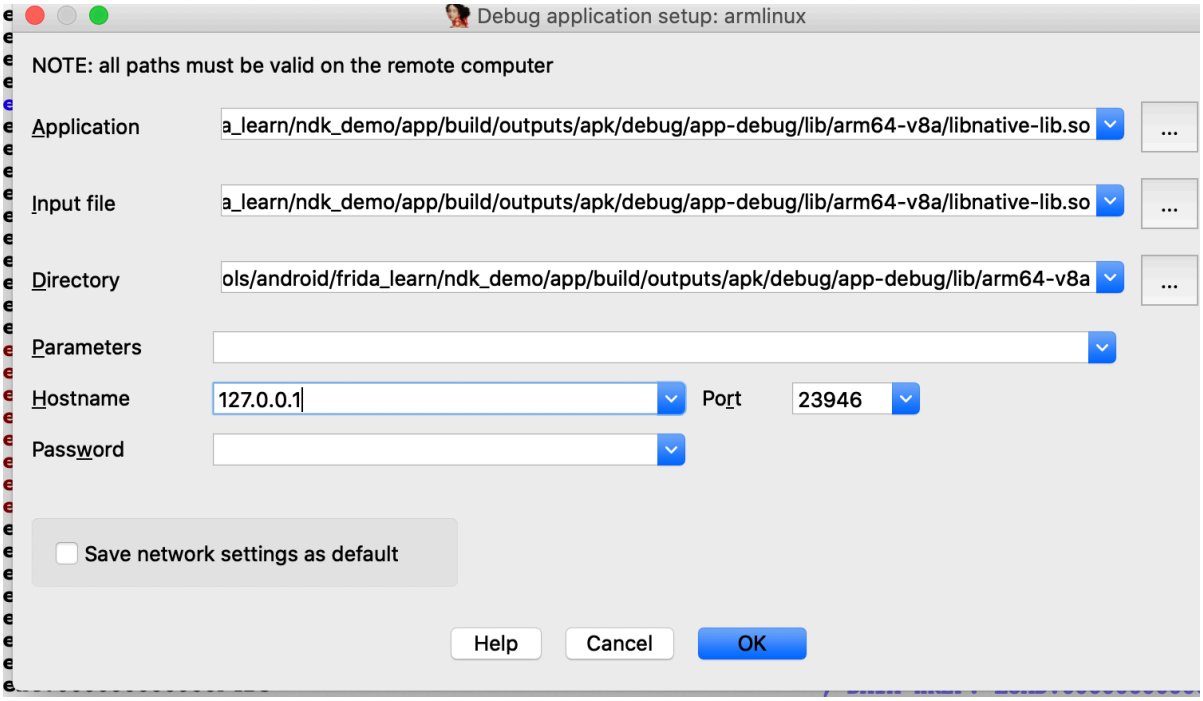
- 启动apk,并让设备将处于一个Waiting For Debugger的状态 `adb shell am start -D -n com.example.ndk_demo/.MainActivity`
- 执行android_server64

```
sailfish:/data/local/tmp # ./android_server64
IDA Android 64-bit remote debug server(ST) v1.22. Hex-Rays (c) 2004-2017
Listening on 0.0.0.0:23946...
```

- 新开一个窗口使用forward程序进行端口转发：`adb forward tcp:23946 tcp:23946`

`adb forward tcp:<本地机器的网络端口号> tcp:<模拟器或是真机的网络端口号>` 例:adb [-d|-e|-s] forward tcp:6100 tcp:7100 表示把本机的6100端口号与模拟器的7100端口建立起相关，当模拟器或真机向自己的7100端口发送了数据，那们我们可以在本机的6100端口读取其发送的内容，这是一个很关键的命令，以后我们使用jdb调试apk之前，就要用它先把目标进程和本地端口建立起关联

- 打开IDA，选择菜单Debugger -> Attach -> Remote ARM Linux/Android debugger
- 打开IDA，选择菜单Debugger -> Process options, 填好，然后选择进程去attach。

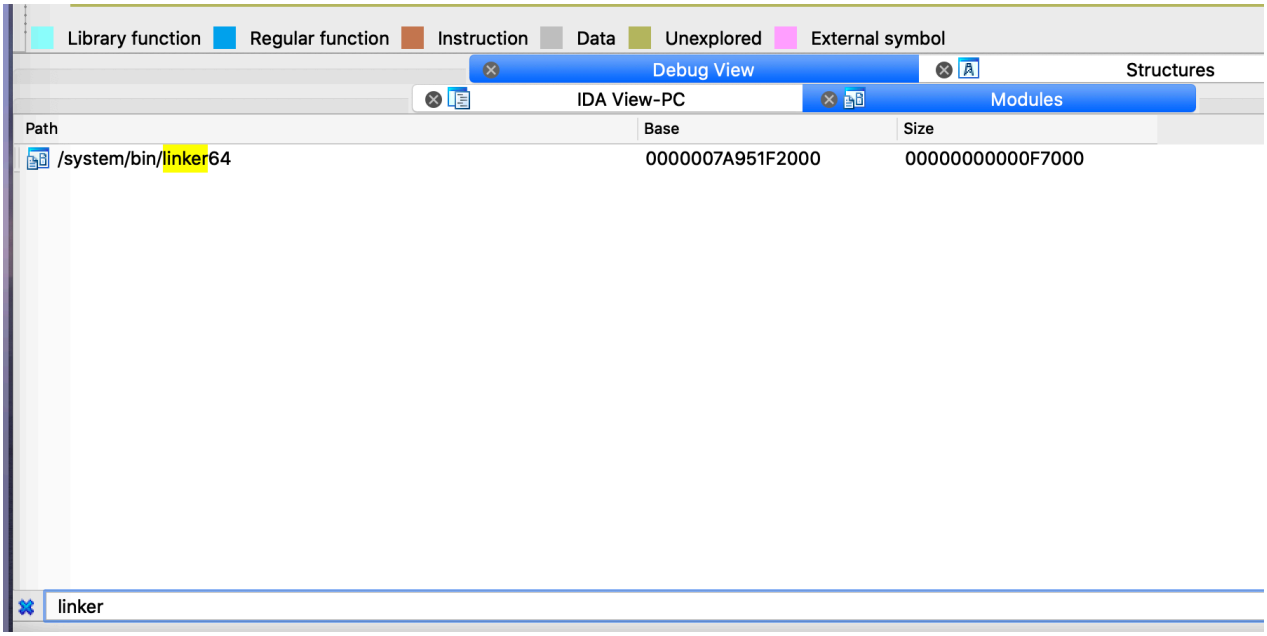


- 查看待调试的进程`adb jdwp`

```
sakura@sakuradeMacBook-Pro:~$ adb jdwp
10436
```

- 转发端口`adb forward tcp:8700 jdwp:10436`，将该进程的调试端口和本机的8700绑定。
- jdb连接调试端口，从而让程序继续运行 `jdb -connect com.sun.jdi.SocketAttach:hostname=127.0.0.1,port=8700`
- 找到断点并断下。

打开module 找到linker64



找到call array函数 下断并按F9断下

最终我确实可以调试到`.init_array`的初始化，具体的代码分析见Linker学习笔记这里。

init_array && JNI_Onload “自吐”

JNI_Onload

目标是找到动态注册的函数的地址，因为这种函数没有导出。

```
JNINativeMethod methods[] = {
            {"stringFromJNI2", "()Ljava/lang/String;", (void *)
stringFromJNI2},
    };
    env->RegisterNatives(env->FindClass("com/example/ndk_demo/MainActivity"),
methods,
                         1);
```

首先`jnitrace -m spawn -i "RegisterNatives" -l libnative-lib.so com.example.ndk_demo`

```
    525 ms [+] JNIEnv->RegisterNatives
    525 ms |- JNIEnv*          : 0x7a106cc1c0
    525 ms |- jclass           : 0x89    { com/example/ndk_demo/MainActivity }
    525 ms |- JNINativeMethod* : 0x7ff0b71120
    525 ms |:     0x79f00d36b0 - stringFromJNI2()Ljava/lang/String;
```

然后`objection -d -g com.example.ndk_demo run memory list modules explore | grep demo`

```
sakura@sakuradeMacBook-Pro:~$ objection -d -g com.example.ndk_demo run memory
list modules explore | grep demo
[debug] Attempting to attach to process: `com.example.ndk_demo`
Warning: Output is not to a terminal (fd=1).
base.odex                                          0x79f0249000  106496 (104.0
KiB)    /data/app/com.example.ndk_demo-
HGAFhnKyKCSIpzn227pwXw==/oat/arm64/base.odex
libnative-lib.so                                   0x79f00c4000  221184 (216.0
KiB)    /data/app/com.example.ndk_demo-
HGAFhnKyKCSIpzn227pwXw==/lib/arm64/libnative...
```

offset = 0x79f00d36b0 - 0x79f00c4000 = 0xf6b0

这样就找到了

init_array

没有支持arm64，可以在安装app的时候`adb install --abi armeabi-v7a`强制让app运行在32位模式

这个脚本整体来说就是hook callfunction，然后打印出init_array里面的函数地址和参数等。

从源码看，关键就是call_array这里调用的call_function，第一个参数代表这是注册的init_array里面的function，第二个参数则是init_array里存储的函数的地址。

```
template <typename F>
static void call_array(const char* array_name __unused,
                       F* functions,
                       size_t count,
                       bool reverse,
                       const char* realpath) {
  if (functions == nullptr) {
```

```
    return;
  }

  TRACE("[ Calling %s (size %zd) @ %p for '%s' ]", array_name, count,
functions, realpath);

  int begin = reverse ? (count - 1) : 0;
  int end = reverse ? -1 : count;
  int step = reverse ? -1 : 1;

  for (int i = begin; i != end; i += step) {
    TRACE("[ %s[%d] == %p ]", array_name, i, functions[i]);
    call_function("function", functions[i], realpath);
  }

  TRACE("[ Done calling %s for '%s' ]", array_name, realpath);
}
```

```
function LogPrint(log) {
    var theDate = new Date();
    var hour = theDate.getHours();
    var minute = theDate.getMinutes();
    var second = theDate.getSeconds();
    var mSecond = theDate.getMilliseconds()

    hour < 10 ? hour = "0" + hour : hour;
    minute < 10 ? minute = "0" + minute : minute;
    second < 10 ? second = "0" + second : second;
    mSecond < 10 ? mSecond = "00" + mSecond : mSecond < 100 ? mSecond = "0" +
mSecond : mSecond;

    var time = hour + ":" + minute + ":" + second + ":" + mSecond;
    var threadid = Process.getCurrentThreadId();
    console.log("[" + time + "]" + "->threadid:" + threadid + "--" + log);

}

function hooklinker() {
    var linkername = "linker";
    var call_function_addr = null;
    var arch = Process.arch;
    LogPrint("Process run in:" + arch);
    if (arch.endsWith("arm")) {
        linkername = "linker";
    } else {
        linkername = "linker64";
        LogPrint("arm64 is not supported yet!");
    }
```

```
    var symbols = Module.enumerateSymbolsSync(linkername);
    for (var i = 0; i < symbols.length; i++) {
        var symbol = symbols[i];
        //LogPrint(linkername + "->" + symbol.name + "---" + symbol.address);
        if (symbol.name.indexOf("__dl__ZL13call_functionPKcPFviPPcS2_ES0_") !=
-1) {
            call_function_addr = symbol.address;
            LogPrint("linker->" + symbol.name + "---" + symbol.address)

        }
    }

    if (call_function_addr != null) {
        var func_call_function = new NativeFunction(call_function_addr,
'void', ['pointer', 'pointer', 'pointer']);
        Interceptor.replace(new NativeFunction(call_function_addr,
            'void', ['pointer', 'pointer', 'pointer']), new
NativeCallback(function (arg0, arg1, arg2) {
            var functiontype = null;
            var functionaddr = null;
            var sopath = null;
            if (arg0 != null) {
                functiontype = Memory.readCString(arg0);
            }
            if (arg1 != null) {
                functionaddr = arg1;

            }
            if (arg2 != null) {
                sopath = Memory.readCString(arg2);
            }
            var modulebaseaddr = Module.findBaseAddress(sopath);
            LogPrint("after load:" + sopath + "--start call_function,type:" +
functiontype + "--addr:" + functionaddr + "---baseaddr:" + modulebaseaddr);
            if (sopath.indexOf('libnative-lib.so') >= 0 && functiontype ==
"DT_INIT") {
                LogPrint("after load:" + sopath + "--ignore
call_function,type:" + functiontype + "--addr:" + functionaddr + "---
baseaddr:" + modulebaseaddr);

            } else {
                func_call_function(arg0, arg1, arg2);
                LogPrint("after load:" + sopath + "--end call_function,type:"
+ functiontype + "--addr:" + functionaddr + "---baseaddr:" + modulebaseaddr);

            }

        }, 'void', ['pointer', 'pointer', 'pointer']));
    }
```

```
}

setImmediate(hooklinker)
```

我调试了一下linker64，因为没有导出call_function的地址，所以不能直接hook符号名，而是要根据偏移去hook，以后再说。 其实要看`init_array`，直接shift+F7去segment里面找`.init_array`段就可以了，这里主要是为了反反调试，因为可能反调试会加在init_array里，hook call_function就可以让它不加载反调试程序。

native层未导出函数主动调用（任意符号和地址）

现在我想要主动调用sakura_add来打印值,可以ida打开找符号，或者根据偏移，总之最终用这个NativePointer指针来初始化一个NativeFunction来调用。

```
extern "C"
JNIEXPORT jint JNICALL
Java_com_example_ndk_1demo_MainActivity_sakuraWithInt(JNIEnv *env, jobject
thiz, jint a, jint b) {
    // TODO: implement sakuraWithInt()
    return sakura_add(a,b);
}
...
int sakura_add(int a, int b){
    int sum = a+b;
    LOGD("sakura add a+b:",sum);
    return sum;
}
```

```
function main() {
    var libnative_lib_addr = Module.findBaseAddress("libnative-lib.so");
    console.log("libnative_lib_addr is :", libnative_lib_addr);
```

```
    if (libnative_lib_addr) {
        var sakura_add_addr1 = Module.findExportByName("libnative-lib.so",
"_Z10sakura_addii");
        var sakura_add_addr2 = libnative_lib_addr.add(0x0F56C) ;
        console.log("sakura_add_addr1 ", sakura_add_addr1);
        console.log("sakura_add_addr2 ", sakura_add_addr2)
    }

    var sakura_add1 = new NativeFunction(sakura_add_addr1, "int", ["int",
"int"]);
    var sakura_add2 = new NativeFunction(sakura_add_addr2, "int", ["int",
"int"]);

    console.log("sakura_add1 result is :", sakura_add1(200, 33));
    console.log("sakura_add2 result is :", sakura_add2(100, 133));
}
setImmediate(main())
...
...
libnative_lib_addr is : 0x79fa1c5000
sakura_add_addr1  0x79fa1d456c
sakura_add_addr2  0x79fa1d456c
sakura_add1 result is : 233
sakura_add2 result is : 233
```

C/C++ hook

Native/JNI层参数打印和主动调用参数构造

jni的基本类型要通过调用jni相关的api转化成c++对象，才能打印和调用。 jni主动调用的时候，参数构造有两种方式，一种是`Java.vm.getenv`，另一种是hook获取env之后来调用jni相关的api构造参数。

C/C++编成so并引入Frida调用其中的函数

// todo