

Android Lab 2

Fiddler

下载网址: <https://www.telerik.com/download/fiddler> 参考文档:
<https://blog.csdn.net/u012206617/article/details/108714615>

PC: 安装证书以解密 HTTPS

安装流程: Tools -> Options -> HTTPS -> Decrypt HTTPS traffic option

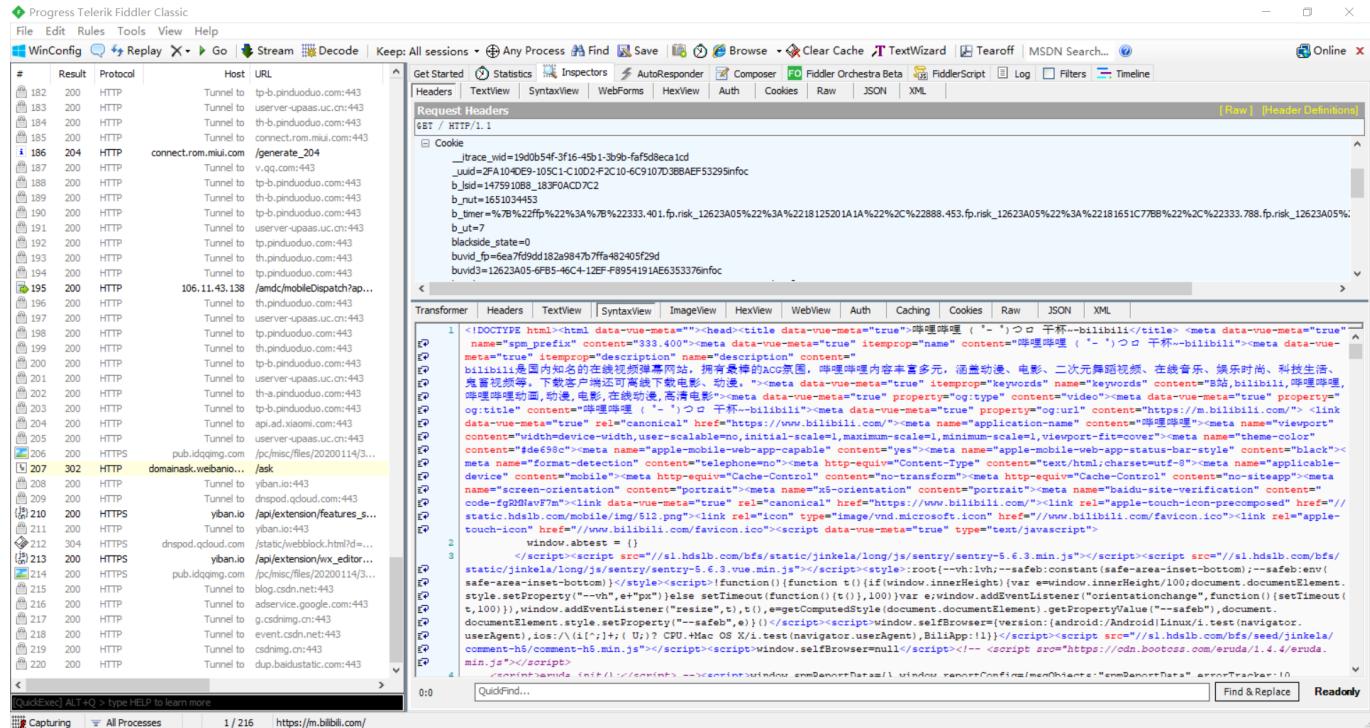
- 选择 DO_NOT_TRUST_FIDDLER 证书安装即可。

Android: 安装证书以抓取安卓手机包

前提: 同一局域网

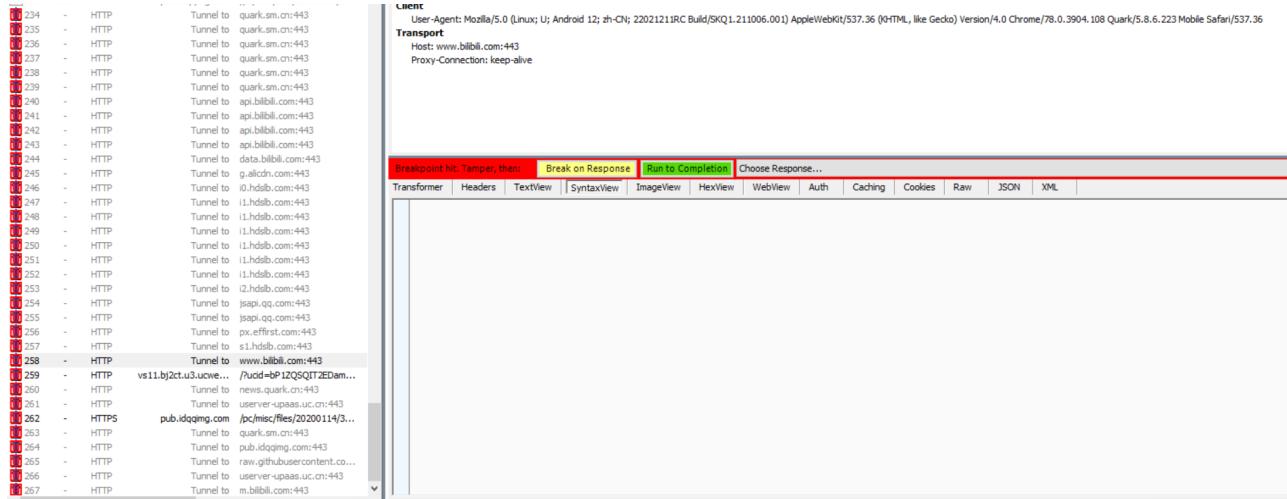
安装流程:

- PC -> Tools -> Options -> Connections -> Allow remote computers to connect
- Android -> WLAN -> 代理 -> 输入 PC 端IP和端口(默认8888) -> 网页登录该地址` ` IP:port -> 下载证书 -> 安装证书。



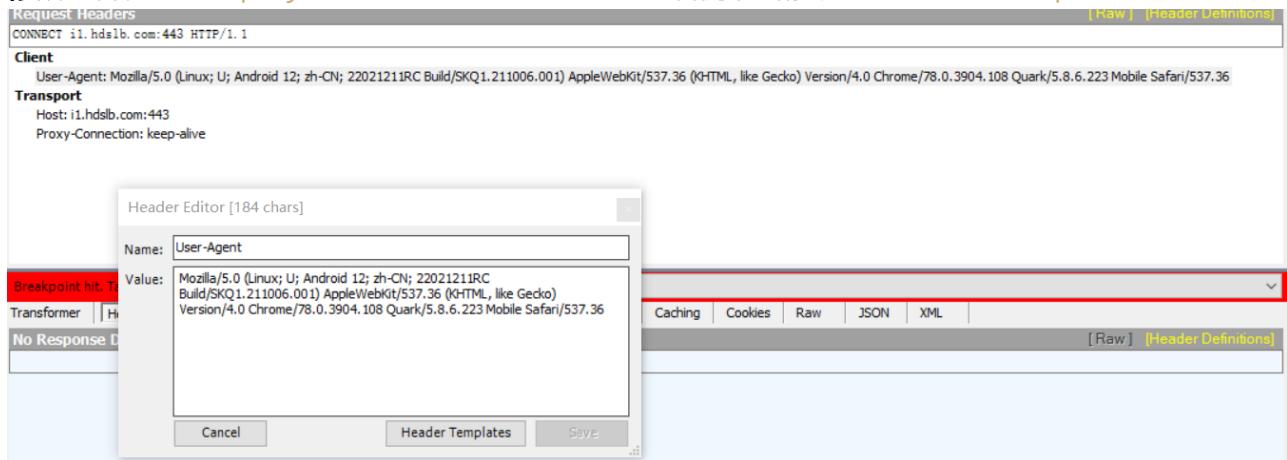
拦截/修改包

- 拦截: fiddle -> rules -> Automatic breakpoints -> Before Requests



- 还原: fiddle -> rules -> Automatic breakpoints -> Disabled

- 修改: 右键 -> replay -> reissue and edit -> 右侧双击修改 -> run to completion



Frida

插桩

插桩指的是目标程序代码中某些位置插入或修改一些代码，从而在目标程序运行过程中获取某些程序状态并加以分析。简单来说就是在代码中插入代码。

- 函数插桩
- 字节码插桩

安装

```
// 前置环境 python3.7, pip
pip install frida==14.2.17
pip install frida-tools(==9.2.4)
```

下载对应架构的frida-server，使用adb传入手机data/local/tmp中。

```
// 以 MuMu模拟器 为例  
adb connect 127.0.0.1:7555  
adb push [frida-server] /data/local/tmp
```

测试

```
// 终端1  
adb shell  
cd /data/local/tmp  
chmod 777 frida-server  
../frida-server  
  
// 终端2  
frida-ps -U
```

```
PS D:\Master Life> frida-ps -U  
PID  Name  
---  
284  abd  
943  android.process.media  
1658 com.android.defcontainer  
1740 com.android.gallery3d  
1415 com.android.keychain  
1792 com.android.phone  
1444 com.android.providers.calendar  
1830 com.android.settings  
1724 com.android.settings:superuser  
760  com.android.systemui  
835  com.tencent.mm  
1322 com.tencent.mm:push
```

使用

参考1: [blog](#) 参考2: [官方文档](#), 有点老了 参考3: 书籍: 安卓 Frida 逆向与抓包实战, 陈桂林著。

以下都以 hook App 的 Java 层为例。

方法1: js + py 混用

```
import frida #导入frida模块  
import sys #导入sys模块  
  
jscode = """ #从此处开始定义用来Hook的javascript代码
```

```

"""
#####
##### 以下除包名外固定 #####
#js中执行send函数后要回调的函数
def on_message(message,data):
    print(message)

#得到设备并劫持进程[xxx.xxx.xxx]
process = frida.get_remote_device().attach('xxx.xxx.xxx')

#创建js脚本
script = process.create_script(jscode)

#加载回调函数，也就是js中执行send函数规定要执行的python函数
script.on('message',on_message)

#加载脚本
script.load()
sys.stdin.read()

```

jscode 使用方法：

1. 修改返回值

```

jscode = """
// App 中 Java 层 hook 函数 Java.perform
Java.perform(function () {
    // Java.use 新建一个对象
    var Myclass= Java.use('这里填写要Hook的类名');

    // Hook的类下的方法名Mymethod
    Myclass.Mymethod.implementation = function ([arg]) {

        // 输出打印相关的提示语结果
        send('Hook success');

        // 对参数进行一些操作
        var ret = this.OutClass(arg);

        return [希望hook的返回值];
    };
});

"""

```

2. 重载函数

```

//如果一个类的两个方法具有相同的名称，需要使用"重载"，若不知具体参数，出错会有提示。
myClass.myMethod.overload().implementation = function(){

```

```
// do sth
}

myClass.myMethod.overload("[B", "[B").implementation = function(param1, param2) {
    // do sth
}

myClass.myMethod.overload("android.context.Context", "boolean").implementation =
function(param1, param2){
    // do sth
}
```

3. 主动调用

```
Java.perform(function(){
    console.log("Inside java perform function")

    // 静态函数主动调用
    var MainActivity = Java.use('xxx.xxx.xxx')
    MainActivity.staticfunc() // 这里 staticfunc 为函数名

    // 动态函数主动调用
    Java.choose('xxx.xxx.xxx',{
        onMatch : function(instance){
            console.log('instance found',instance)
            instance.func() // 这里 func 为函数名
        },
        onComplete: function(){
            console.log('Search Complete')
        }
    })
})
```

方法2: js 与 py 分开自动化调用

只需要编写好 `hooks.js` 文件后，开两个终端分别运行以下代码即可。

```
// 终端1
python autostart.py
// 终端2
pyython autojs.py com.xxx.xxx
```

```
# autostart.py
# 启动 frida 服务

import sys
import subprocess
```

```
from turtle import forward

# MuMu 模拟器
forward0 = "adb connect 127.0.0.1:7555"
# Frida
forward1 = "adb forward tcp:27042 tcp:27042"
forward2 = "adb forward tcp:27043 tcp:27043"
# 运行 frida-server
cmd = ["adb shell","cd /data/local/tmp","./frida-server"]

def Forward0():
    s = subprocess.Popen(str(forward0+"\r\n"), stderr=subprocess.PIPE,
        stdin=subprocess.PIPE, stdout=subprocess.PIPE, shell=True)
    stderrinfo, stdoutinfo = s.communicate()
    return s.returncode

def Forward1():
    s = subprocess.Popen(str(forward1+"\r\n"), stderr=subprocess.PIPE,
        stdin=subprocess.PIPE, stdout=subprocess.PIPE, shell=True)
    stderrinfo, stdoutinfo = s.communicate()
    return s.returncode

def Forward2():
    s = subprocess.Popen(str(forward2+"\r\n"), stderr=subprocess.PIPE,
        stdin=subprocess.PIPE, stdout=subprocess.PIPE, shell=True)
    stderrinfo, stdoutinfo = s.communicate()
    return s.returncode

def Run():
    s = subprocess.Popen(str(cmd[0])+"\r\n", stderr=subprocess.PIPE,
        stdin=subprocess.PIPE, stdout=subprocess.PIPE, shell=True)

    for i in range(1,len(cmd)):
        s.stdin.write(bytes(str(cmd[i])+"\r\n"),encoding='utf-8')
        s.stdin.flush()

    stderrinfo, stdoutinfo = s.communicate()
    return s.returncode

if __name__ == "__main__":
    Forward0()
    print("adb connect 127.0.0.1:7555")
    Forward1()
    print("adb forward tcp:27042 tcp:27042")
    Forward2()
    print("adb forward tcp:27043 tcp:27043")
    print("Android server--->./frida-server")
    print("success-->please to check `frida-ps -U`")
    Run()
```

```

# autojs.py
# 自动启动 js 脚本
import frida, sys
import io

def on_message(message, data):
    if message['type'] == 'send':
        print("[*] {0}".format(message['payload']))
    else:
        print(message)

def run(pkg):
    jscode = io.open('hooks.js', 'r', encoding='utf8').read()
    device = frida.get_usb_device(timeout=5)
    pid = device.spawn(pkg)
    session = device.attach(pid)
    script = session.create_script(jscode)
    script.on('message', on_message)
    script.load()
    sys.stdin.read()

def main(argv):
    if len(argv) != 2:
        print("must input two arg")
        print("For example: python demo.py packName")
    else:
        run(argv[1])

if __name__ == "__main__":
    main(sys.argv)

```

Xposed

Xposed框架是一款可以在不修改APK的情况下影响程序运行(修改系统)的框架服务，于2017年停止更新。其Hook流程比较Frida略显臃肿。Xposed集成了App，直接安装在手机即可。在创建一个Empty Activity后，Hook流程为：

1. 依赖

```

// build.gradle
provided 'de.robv.android.xposed:api:82'
provided 'de.robv.android.xposed:api:82:sources'

// AndroidManifest.xml
<meta-data
    android:name="xposedmodule"
    android:value="true"/>
<meta-data
    android:name="xposeddescription"
    android:value="这是一个Xposed"/>

```

```
<meta-data
    android:name="xposedminversion"
    android:value="82"/>
```

2. 新建 assets 文件夹，文件夹下新建 xposed_init 文件，文件中填写 XposedInit 的完整包名：

`com.xxx.xxx.XposedInit` (xxx为创建时名)

3. 开始 Hook

```
public class XposedInit implements IXposedHookLoadPackage {
    @Override
    public void handleLoadPackage(final XC_LoadPackage.LoadPackageParam lpparam) {
        if (lpparam.packageName.equals("[要Hook的包名]")) {
            // 开始操作
            // ...
        }
    }
}
```

Task1:Jarvis OJ 题目

由于 Jarvis OJ 登陆不上(502服务器问题)，所以只能找副本进行参考。

[61dctf]androideeasy

没有加壳没有混淆的一道简单逆向题，通过工具反汇编可以发现就是一个异或运算：

```
a = [113, 123, 118, 112, 108, 94, 99, 72, 38, 68, 72, 87, 89, 72, 36, 118, 100,
78, 72, 87, 121, 83, 101, 39, 62, 94, 62, 38, 107, 115, 106]
flag = ''
for i in range(0,31):
    flag += chr(a[i] ^ 0x17)
print(flag)
# 结果: flag{It_1S_@N_3asY_@nDr0)I)1|d}
```

DD - Android Easy

和上题类似，通过一个数组操作函数 i() 计算 flag 并与输入比较，则直接将 i() 函数提取出来运行即可：

```
public class solution{
    private static final byte[] p = {-40, -62, 107, 66, -126, 103, -56, 77,
    122, -107, -24, -127, 72, -63, -98, 64, -24, -5, -49, -26, 79, -70, -26,
    -81, 120, 25, 111, -100, -23, -9, 122, -35, 66, -50, -116, 3, -72, 102,
    -45, -85, 0, 126, -34, 62, 83, -34, 48, -111, 61, -9, -51, 114, 20, 81,
    -126, -18, 27, -115, -76, -116, -48, -118, -10, -102, -106, 113, -104,
    98, -109, 74, 48, 47, -100, -88, 121, 22, -63, -32, -20, -41, -27, -20,
    -118, 100, -76, 70, -49, -39, -27, -106, -13, -108, 115, -87, -1, -22,
```

```

-53, 21, -100, 124, -95, -40, 62, -69, 29, 56, -53, 85, -48, 25, 37, -78,
11, -110, -24, -120, -82, 6, -94, -101};
private static final byte[] q = {-57, -90, 53, -71, -117, 98, 62, 98,
101, -96, 36, 110, 77, -83, -121, 2, -48, 94, -106, -56, -49, -80, -1,
83, 75, 66, -44, 74, 2, -36, -42, -103, 6, -115, -40, 69, -107, 85, -78,
-49, 54, 78, -26, 15, 98, -70, 8, -90, 94, -61, -84, 64, 112, 51, -29,
-34, 126, -21, -126, -71, -31, -24, -60, -2, -81, 66, -84, 85, -91, 10,
84, 70, -8, -63, 26, 126, -76, -104, -123, -71, -126, -62, -23, 11, -39,
70, 14, 59, -101, -39, -124, 91, -109, 102, -49, 21, 105, 0, 37, Byte.
MIN_VALUE, -57, 117, 110, -115, -86, 56, 25, -46, -55, 7, -125, 109, 76,
104, -15, 82, -53, 18, -28, -24};

public static String i() {
    byte[] bArr = new byte[p.length];
    for (int i = 0; i < bArr.length; i++) {
        bArr[i] = (byte) (p[i] ^ q[i]);
    }
    byte b = bArr[0];
    int i2 = 0;
    while (bArr[b + i2] != 0) {
        i2++;
    }
    byte[] bArr2 = new byte[i2];
    for (int i3 = 0; i3 < i2; i3++) {
        bArr2[i3] = bArr[b + i3];
    }
    return new String(bArr2);
}
public static void main(String[] args){
    System.out.println(i());
}
}

// 结果: DDCTF-3ad60811d87c4a2dba0ef651b2d93476@didichuxing.com

```

FindPass

由于找不到源文件，所以做不了这道题。但是其新意也就是利用图片存储的矩阵值进行运算获取 flag，并且要注意矩阵值和 char 值之间的越界问题。

参考文档: <https://blog.csdn.net/getsum/article/details/85276512>

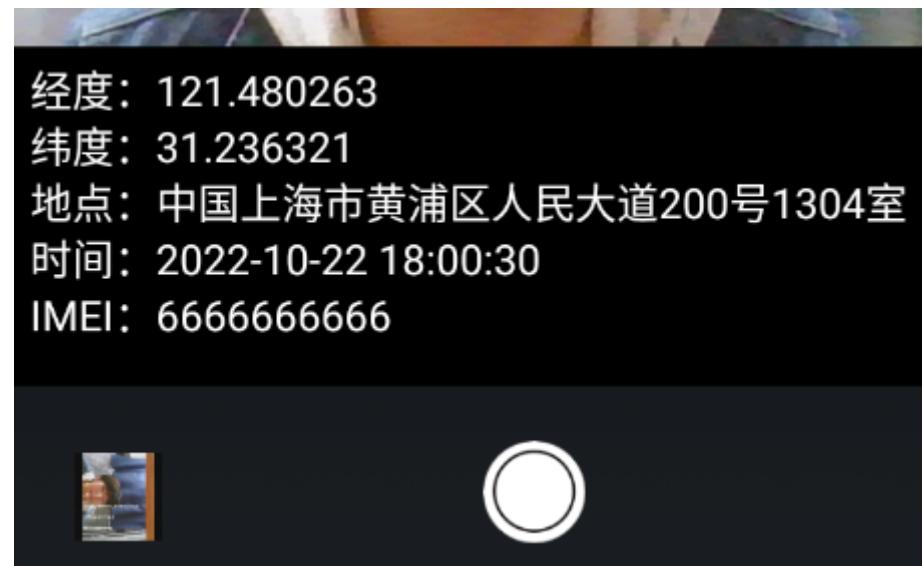
Task2：元道经纬相机 IMEI 数据流分析

首先反汇编程序，发现混淆又加壳，难度比较大，我决定试着从一个 demo 下手。

远古版本的元道经纬相机2.2.4：轻量混淆

根据 Android 编程的相关知识，获取手机 IMEI 大多要调用 getDeviceId 函数，全局搜索后能找到：

```
private String w() {
    String str = "";
    if (ActivityCompat.checkSelfPermission(this, "android.permission.READ_PHONE_STATE") == PackageManager.PERMISSION_GRANTED) {
        str = ((TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE)).getDeviceId();
    }
    return (str == null || str.equals("")) ? "123456" : str;
}
```



通过 frida hook，成功得比较轻松：

frida 脚本：

```
import frida #导入frida模块
import sys #导入sys模块

jscode = """
Java.perform(function () {
    send('Here we go!');
    var Myclass = Java.use("com.ydtx.camera.MainActivity");
    Myclass.w.overload().implementation = function () {
        send('Hook success');
        var type = arguments[0];
        send('arg:' + type);
        return "6666666666";
    };
});
"""

#####
##### 以下除包名外固定 #####
#js中执行send函数后要回调的函数
def on_message(message,data):
    print(message)

#得到设备并劫持进程[xxx.xxx.xxx]
process = frida.get_remote_device().attach('com.ydtx.camera')

#创建js脚本
```

```
script = process.create_script(jscode)

#加载回调函数，也就是js中执行send函数规定要执行的python函数
script.on('message',on_message)

#加载脚本
script.load()
sys.stdin.read()
```

最新版元道经纬相机5.5.3：去壳+高度混淆

参考去壳工具: <https://github.com/hluwa/FRIDA-DEXDump>

1. 去壳，通过 FRIDA_DEXDump 工具（工具使用非常简单，pip 安装后，用之前写好的 `autostart.py` 启动后，在安卓端运行要去壳的 app 然后终端输入 `frida-dexdump -FU`），可以发现有八个包。通过 jadx 查看，前几个包信息量比较大，可以进行静态分析。

2. 开始寻找，仍然是通过全局搜索 `getDeviceId` 的方式，在第三个 dex 包中找到了 40 多个，最终锁定到了几个函数中。首先是 `DeviceConfig` 类中的 `getImeiNew` 函数，看起来就非常地像，尝试 hook，发现启动后确实有触发该函数。

```

4 jscode = """
5     Java.perform(function () {
6         send('Here we go!');
7         var Myclass = Java.use("com.umeng.commonsdk.statistics.common.DeviceConfig");
8         Myclass.getImeiNew.implementation = function (context) {
9             send('Hook success');
10            var type = arguments[0];
11            send('arg:' + type);
12            return "6666666666";
13        };
14    });

```

问题 输出 调试控制台 终端 JUPYTER

```

Traceback (most recent call last):
  File ".\tmp.py", line 34, in <module>
    sys.stdin.read()
KeyboardInterrupt
PS D:\Master Life> python .\tmp.py
{'type': 'send', 'payload': 'Here we go!'}
Traceback (most recent call last):
  File ".\tmp.py", line 34, in <module>
    sys.stdin.read()
KeyboardInterrupt
PS D:\Master Life> python .\tmp.py
{'type': 'send', 'payload': 'Here we go!'}
{'type': 'send', 'payload': 'Hook success'}
{'type': 'send', 'payload': 'arg:com.ydtx.camera.App@c7a40ca'}

```

这里尝试 hook 全局变量 sImei 但是应用里面仍然显示正常的 IMEI，说明这个 `DeviceConfig` 类并不是我们的目标。

```

4 jscode = """
5     Java.perform(function () {
6         send('Here we go!');
7         var Myclass = Java.use("com.umeng.commonsdk.statistics.common.DeviceConfig");
8         Myclass.getImeiNew.implementation = function (context) {
9             send('Hook success');
10            Myclass.sImei.value = "6666666666";
11            send('imei:' + Myclass.sImei.value);
12            return "6666666666";
13        };
14    });

```

问题 输出 调试控制台 终端 JUPYTER

```

frida.ProcessNotFoundError: unable to find process with name 'com.ydtx.camera'
PS D:\Master Life> python .\tmp.py
{'type': 'send', 'payload': 'Here we go!'}
Traceback (most recent call last):
  File ".\tmp.py", line 34, in <module>
    sys.stdin.read()
KeyboardInterrupt
PS D:\Master Life> python .\tmp.py
{'type': 'send', 'payload': 'Here we go!'}
Traceback (most recent call last):
  File ".\tmp.py", line 34, in <module>
    sys.stdin.read()
KeyboardInterrupt
PS D:\Master Life> python .\tmp.py
{'type': 'send', 'payload': 'Here we go!'}
{'type': 'send', 'payload': 'Hook success'}
{'type': 'send', 'payload': 'imei:6666666666'}
{'type': 'send', 'payload': 'Hook success'}
{'type': 'send', 'payload': 'imei:6666666666'}

```

3. 根据经验，一般参数放在 `utils` 类中，但是惨不忍睹地，该类基本没有被 jadx 反汇编成功。不过通过全局搜索倒是找到了 c 函数中有获取 IMEI 的操作，并且返回的也是 String 类，尝试混淆一下，发现成功。

```
4         return AspectRatio.g(height, width);
    }

/* JADY WARNING: Removed duplicated region for block: B:15:0x0036 */
/* JADY WARNING: Removed duplicated region for block: B:17:? A[RETURN, SYNTHETIC] */
/* Code decompiled incorrectly, please refer to instructions dump. */
public static java.lang.String c(android.content.Context r4) {
/*
    java.lang.String r0 = "phone"
    java.lang.Object r0 = r4.getSystemService(r0)
    android.telephony.TelephonyManager r0 = (android.telephony.TelephonyManager) r0
    int r1 = android.os.Build.VERSION.SDK_INT
    java.lang.String r2 = ""
    r3 = 29
    if (r1 < r3) goto L_0x0012
L_0x0010:
    r4 = r2
    goto L_0x0034
L_0x0012:
    r3 = 26
    if (r1 < r3) goto L_0x0021
    boolean r4 = o(r4)
    if (r4 == 0) goto L_0x0010
    java.lang.String r4 = r0.getImei()
    goto L_0x0034
L_0x0021:
    r3 = 23
    if (r1 < r3) goto L_0x0030
    boolean r4 = o(r4)
    if (r4 == 0) goto L_0x0010
    java.lang.String r4 = r0.getDeviceId()
    goto L_0x0034
L_0x0030:
    java.lang.String r4 = r0.getDeviceId()
L_0x0034:
    if (r4 == 0) goto L_0x0037
    r2 = r4
L_0x0037:
    return r2
*/
```

```

4 jscode = """
5     Java.perform(function () {
6         send('Here we go!');
7         var Myclass = Java.use("com.umeng.commonsdk.statistics.common.DeviceConfig");
8             Myclass.getImeiNew.implementation = function (context) {
9                 send('Hook success');
10                Myclass.sImei.value = "6666666666";
11                send('imei:' + Myclass.sImei.value);
12                return "6666666666";
13            };
14
15            var Myclass2 = Java.use("com.ydtx.camera.utils.z");
16            Myclass2.c.implementation = function (context) {
17                send('Hook success2');
18                return "6666666666";
19            };
20        });
21    """
```

问题 输出 调试控制台 终端 JUPYTER

```

Traceback (most recent call last):
  File ".\tmp.py", line 40, in <module>
    sys.stdin.read()
KeyboardInterrupt
PS D:\Master Life> python .\tmp.py
[{'type': 'send', 'payload': 'Here we go!'}
{'type': 'send', 'payload': 'Hook success2'}
{'type': 'send', 'payload': 'Hook success2'}
{'type': 'send', 'payload': 'Hook success2'}
{'type': 'send', 'payload': 'Hook success'}
{'type': 'send', 'payload': 'imei:6666666666'}
{'type': 'send', 'payload': 'Hook success'}
{'type': 'send', 'payload': 'imei:6666666666'}
```



成功截图：

frida 代码：

```

import frida #导入frida模块
import sys #导入sys模块

jscode = """
    Java.perform(function () {
        send('Here we go!');
        var Myclass =
Java.use("com.umeng.commonsdk.statistics.common.DeviceConfig");
        Myclass.getImeiNew.implementation = function (context) {
            send('Hook success');
            Myclass.sImei.value = "6666666666";
```

```
        send('imei:' + Myclass.sImei.value);
        return "6666666666";
    };

    var Myclass2 = Java.use("com.ydtx.camera.utils.z");
    Myclass2.c.implementation = function (context) {
        send('Hook success2');
        return "6666666666";
    };
};

"""
#####
##### 以下除包名外固定 #####
#js中执行send函数后要回调的函数
def on_message(message,data):
    print(message)

#得到设备并劫持进程[xxx.xxx.xxx]
process = frida.get_remote_device().attach('com.ydtx.camera')

#创建js脚本
script = process.create_script(jscode)

#加载回调函数，也就是js中执行send函数规定要执行的python函数
script.on('message',on_message)

#加载脚本
script.load()
sys.stdin.read()
```

Task3 : 流量追踪分析

实验App：超星学习通。 使用手机点击超星学习通并模拟使用部分功能，有以下抓包：

1. 初始界面，可以看到是一个 Get 请求

The screenshot shows a Fiddler session with the following details:

- Session Headers:** GET /api/apkInfos.aspx?apkId=con_chaoxing.mobile&spid=103323031 HTTP/1.1
- Client Headers:** Accept-Encoding: gzip, Accept-Language: zh-CN, User-Agent: Dalvik/2.1.0 (Linux; U; Android 12; 22021211RC Build/SKQ1.2111006.001) (chldid:c229fb7b203b999e19c22dbd3fb71d) (device:2201)
- Cookies:** _d=1664117134875, _industry=9, _bd=72150604, _uid=103323031, DSTASH_LOG_C=38-UN_15700-US_103323031-T_1664117134877, fdcCount=1, K1450_SERVER_ECR=REPRFSWrdQRWdcdNwdwM2pxEfFSYmgORDNHM29LMFM4ZGR4bX3M3VPNOfGQ02zIMDjXezXz0oASGWhn1dErU7U, lv=1, sso_id=103323031, ufb=02d393beef7a0d92a9ed2f5845e6fb197d74ebfcbe1d6cfdf301d66b1ad8a61529dc093e8d4b5e0b307076524285a98cb27fe88bb63, UBD=103323031, vC=918246153004-10f650824-F5CB40876C3, vC2=AD626533DC-4894-3A4E620B8895-4d4, vC3=E_d-2%bd2e0%2By0zvTolkWtRzWgRz%2B41VhYNDyWFnNzh%2B0tgFx4%2FmDcok7JnLfWzFLtf.%2BgG5AUxD7k0C67ubq, vC4=918246153004-93a8839e-27731bd, vC5=918246153004-93a8839e-27731bd.
- Transport:** Connection: Keep-Alive, Host: apps.chaoxing.com
- Body Content:** (Redacted)

The main pane displays a list of requests and responses, with several entries highlighted in red. A large red arrow points from the "Host" header in the request to the "Host" entry in the transport section. The bottom right corner of the interface has a "Readonly" button.

2. 功能界面，可以看见是一个 POST 请求，并且其后还伴随着一些 Get 请求加载图片

① Host: yhhd.chaoxing.com

② Get 请求

③ Client

④ Cookies

⑤ Miscellaneous & Security: 从这里可以看出文件是 image

⑥ Transport

⑦ Body: 图片内容

公告板

3. 同时还可以发现，超星打包时 js 文件甚至注释都没有处理。

The screenshot shows the Fiddler interface with a request for a JavaScript file. The file content includes several comments and some code. A red box highlights the code section, which starts with:

```

870 B postNotification(cmd, $.extend({
871   show: 0,
872   shade: 1, //是否显示 1显示, 0不显示
873   width: '',
874   height: '',
875   icon: 'http://static.basicedu.chaoxing.com/app/share.png', //
876   //菜单图标, 为空或没有此属性, 则不显示
877   // option: 'var preHandler = function (e) {e.preventDefault();};$( "#' +
878   // .share_pop" ).show():document.querySelector("body>div").addEventListener("touchmove",
879   , preHandler, false);$( "#'.share_pop" ).click(function () {$(this).hide():document.
880   querySelector("body>div").removeEventListener("touchmove", preHandler, false)});',
881   //操作, 实际类型为js方法, 在客户端上调用用app内的js方法
882   opt: ''
883 }, children));
884 );
885 
```

A red annotation at the bottom right of the highlighted code area says "打包时注释没有处理" (Comments are not handled during packaging).

4. 再往前找，可以找到 HTTPS 的连接环节，使用的是 TLS12 协议。

The screenshot shows the Fiddler interface with the "SyntaxView" tab selected. It displays the TLS handshake details. A red box highlights the certificate information, which includes:

TLS 的密钥交换环节

```

1 Encrypted HTTPS traffic flows through this CONNECT tunnel. HTTPS Decryption is
2 enabled in Fiddler, so decrypted sessions running in this tunnel will be shown in
3 the Web Sessions list.
4
5 Secure Protocol: Tls12
6 Cipher: Aes128 128bits
7 Hash Algorithm: Sha256 256bits
8 Key Exchange: ECDHE_RSA (0xae06) 255bits
9
10 == Server Certificate =====
11 [Subject]
12   CN=*.chaoxing.com, O=北京世纪超星信息技术发展有限责任公司, S=北京市, C=CN
13 [Issuer]
14   CN=WoTrus OV Server CA [Run by the Issuer], O=WoTrus CA Limited, C=CN
15 [Serial Number]
16   6C3B8EA231A890DBEC4862D0E2D93128
17
18 [Not Before]
19   2022/1/19 8:00:00
20
21 [Not After]

```

证书

A red box also highlights the certificate subject information.

5. 在抓包过程中，退出登录后无法登录，是由于 Fiddler 抓 HTTPS 包导致的问题（开了代理后证书校验/Cookies可能不通过），option 取消勾选 Captre HTTPS CONNECTS 即可登录。