# Frida Hook 常用函数、java 层 hook、so 层 hook、RPC、群控

转载

擒贼先擒王 已于 2022-03-29 17:21:28 修改 5305 收藏 42
分类专栏： Hook 框架 之 Frida 文章标签： java android
于 2021-01-29 00:51:59 首次发布
原文链接：https://www.52pojie.cn/thread-1196917-1-1.html
版权

Hook 框架 之 Frida 专栏收录该内容
9 篇文章 12 订阅
订阅专栏

From：Frida hook 常用函数分享：https://www.52pojie.cn/thread-1196917-1-1.html
From：Frida Hook Android 常用方法：https://blog.csdn.net/zhy025907/article/details/89512096

> Frida 使用：https://zhuanlan.zhihu.com/p/339504595

Frida 官方手册 - 函数 Hook：https://blog.csdn.net/freakishfox/article/details/78289293
Frida Java Hook 详解：代码及示例 (上)：https://www.secpulse.com/archives/132082.html

**写 APP爬虫会需要用到哪些工具呢？：https://zhuanlan.zhihu.com/p/63899519**

> 《风控要略——互联网业务反欺诈之路》主要分为洞察黑产、体系构建、实战教程和新的战场4个部分

## 使用 IDE 编写 frida js 时智能提示

工欲善其事，必先利其器。编写 frida js 时让 IDE 智能提示

### 方法 1：

- git clone https://github.com/oleavr/frida-agent-example.git
- cd frida-agent-example/，执行命令 npm install
- 然后使用 VSCode、pycharm、idea 等 IDE 打开此工程，在 agent 目录下编写 JavaScript 代码时就会有智能提示。

**JS单步调试**

：https://bbs.pediy.com/thread-265160.htm

能愉快的单步调试 frida 的 js 脚本，可以方便不少。首先运行 frida 脚本

```
frida -l </Users/name/path/test.js> --debug --runtime=v8 <port/name>
```

或者

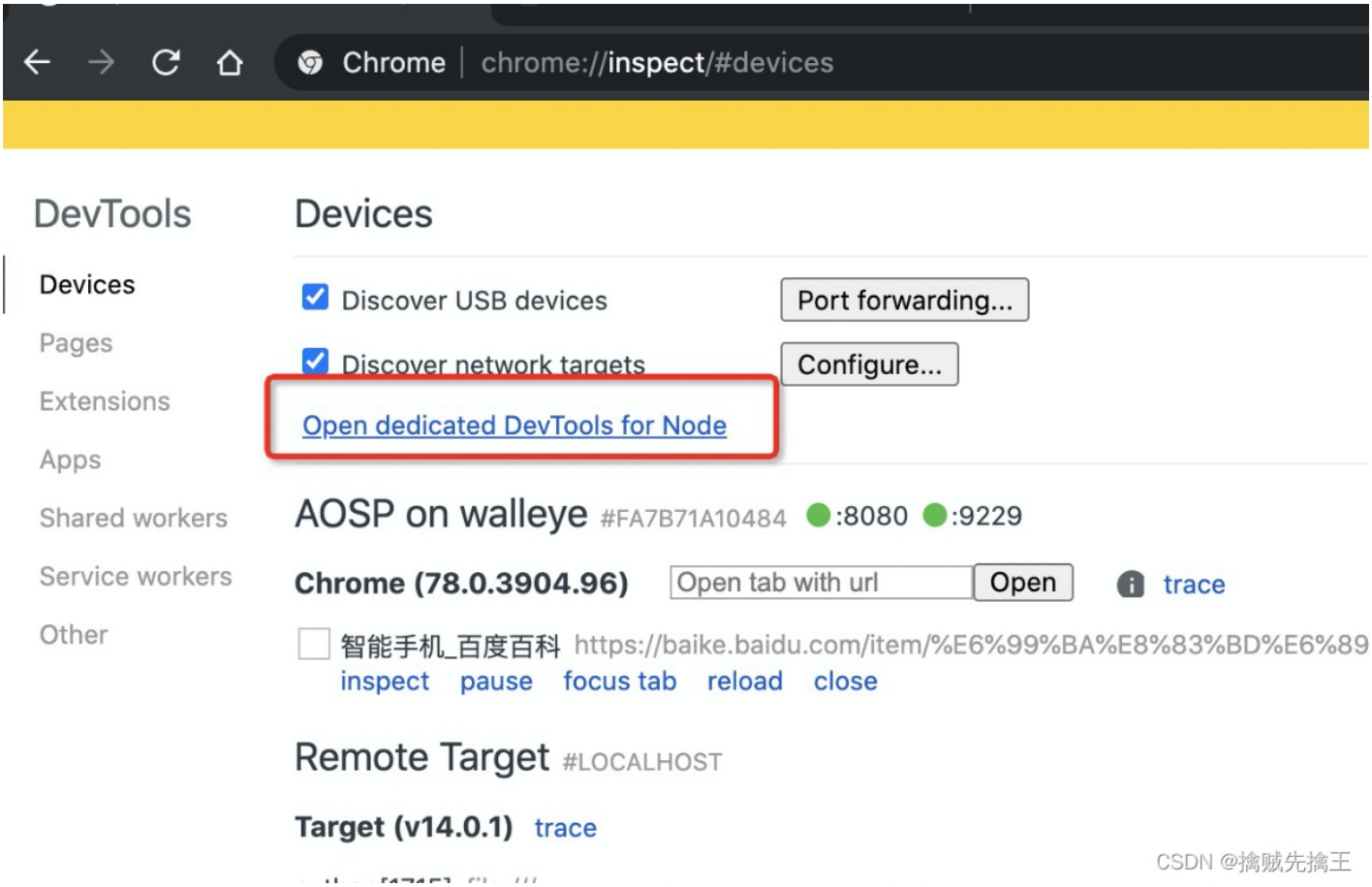
```
session = dev.attach(app.pid)
script = session.create_script(jscode, runtime="v8")
session.enable_debugger()
```

启动后会回显 Inspector 正在监听 9229 默认端口

```
Chrome Inspector server listening on port 9229
```

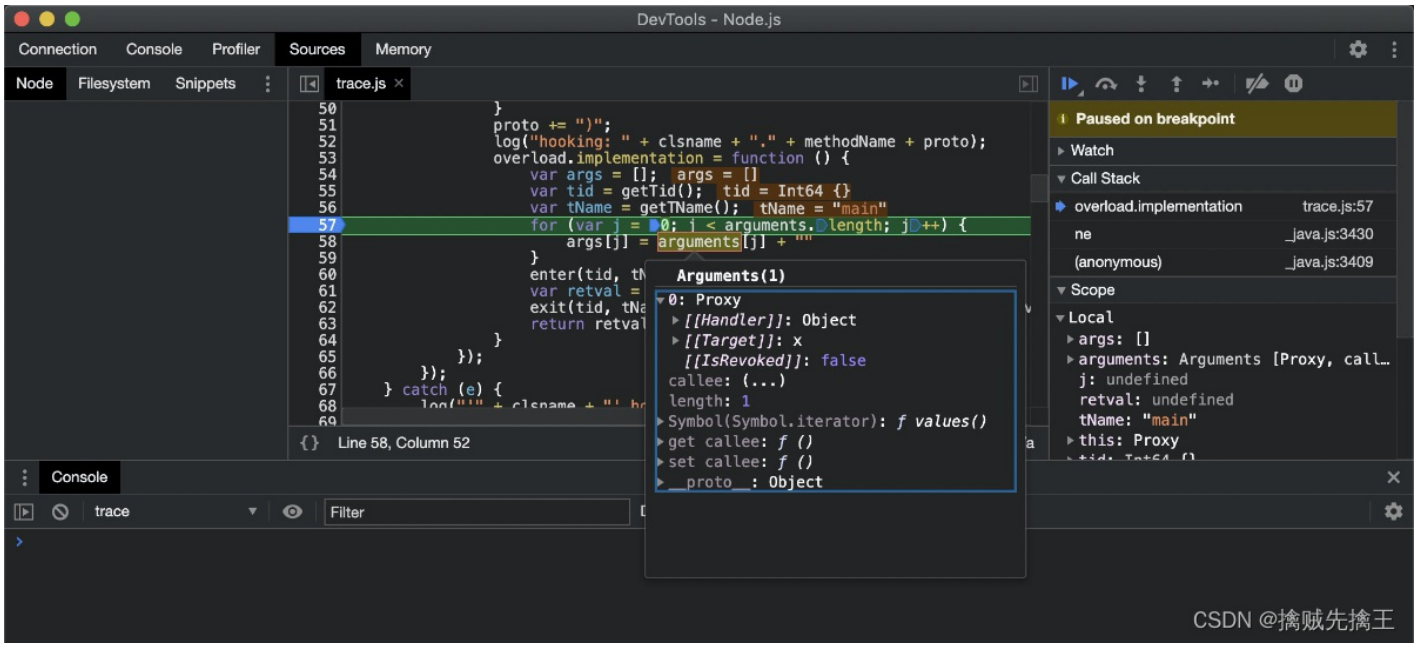
### chrome

打开 chrome://inspect, 点击 `Open dedicated DevTools for Node`。



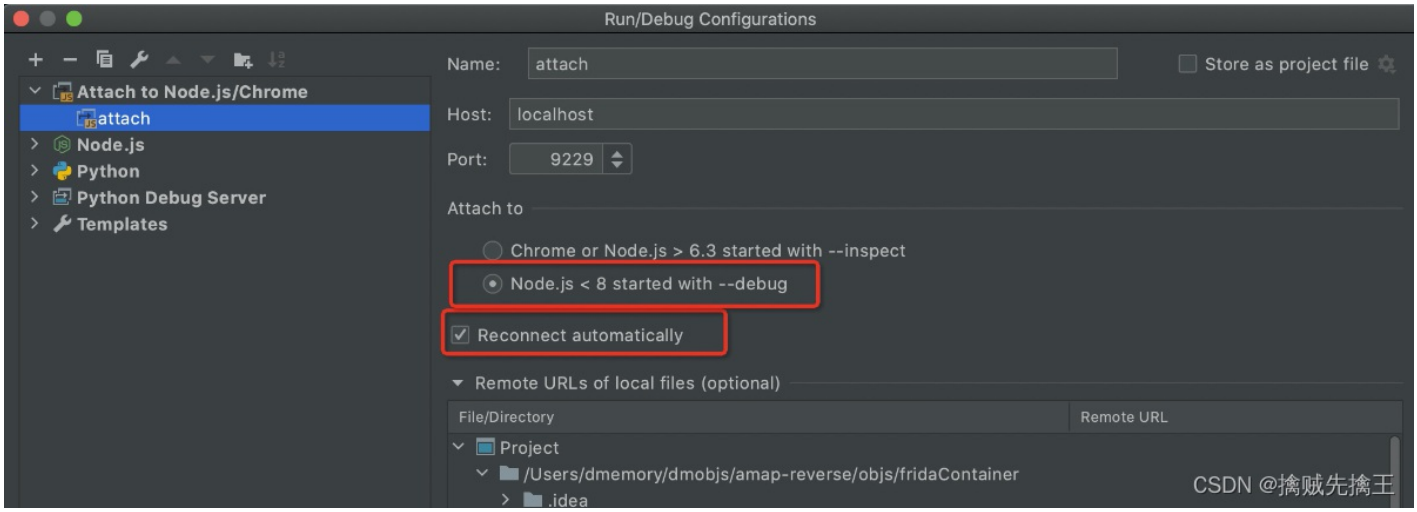
CSDN @擒贼先擒王

此时 debug 已经连接，切换至 `Sources`，按 `Command + P` 加载要调试的脚本，即可下断调试了。

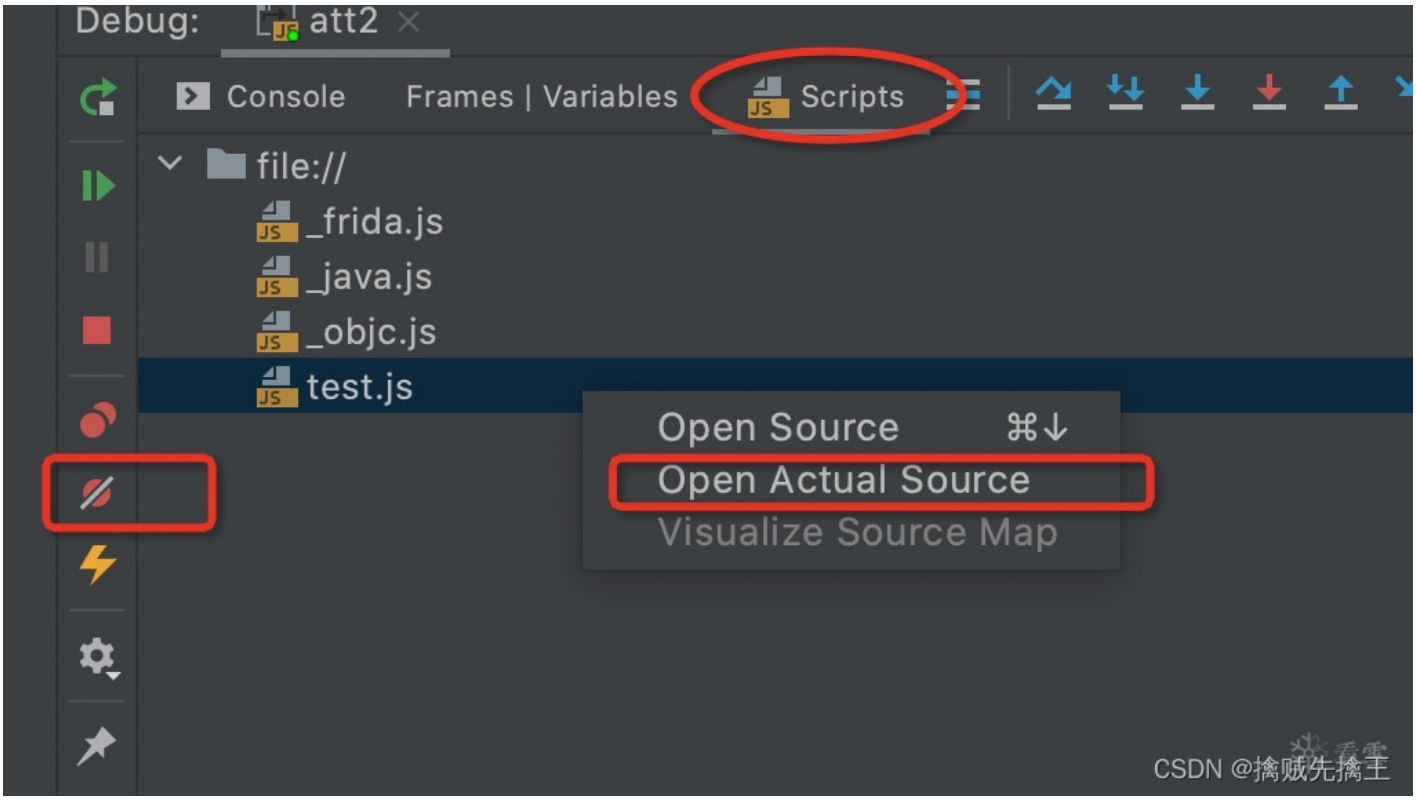



### pycharm

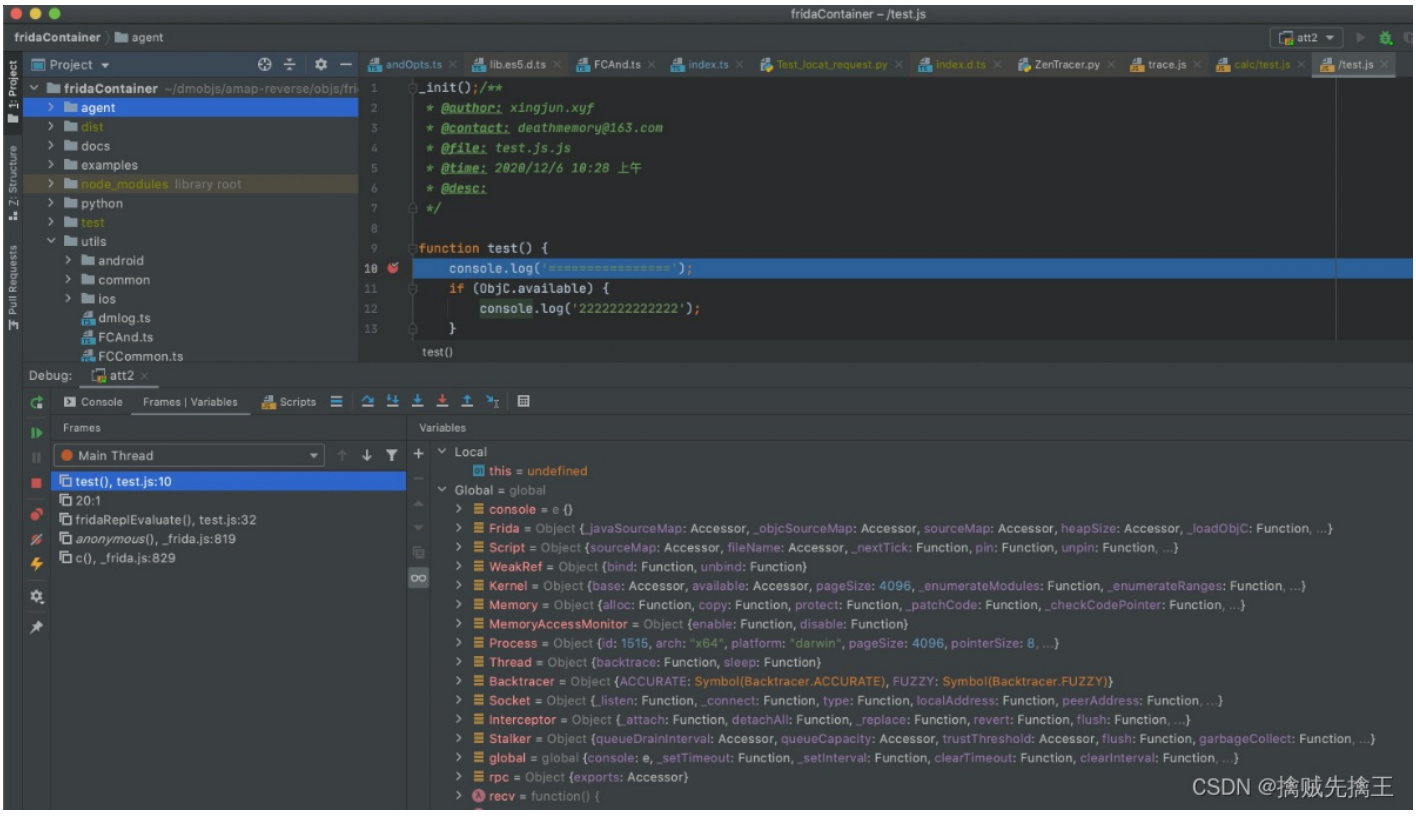
首先安装 Node.js 插件，重启。添加调试器 `Attaching to Node.js/Chrome`，端口默认即可。`Attach to` 应选择 `Node.js < 8 started with --debug`， 下面的自动重连选项可选可不选。




触发断点需要在 debug 窗口切换到 script 选项卡，右键要调试的脚本，选择 Open Actual Source，在新打开的 Actual Source 窗口设置好断点后，需要再取消/启用一次所有断点作为激活，发现断点上打上对勾才真正可用了。




接下来就可以愉快的调试了




优缺点

1. 用 Chrome 调试支持的更为顺滑，调试脚本自动重加载，断点也能正确响应。
2. 用 PyCharm 调试断点有时需要手动激活有点麻烦，但可以使用PyCharm 的Debug 窗口和快捷键。
3. PyCharm 使用 ts 环境调试时，可以直接在ts文件上下断，也不需要手动激活断点。

## 方法 2:

：https://bbs.pediy.com/thread-258513.htm

**基础 frida 代码完成**

frida 代码提示插件。如果你安装了 Frida, 不管你熟不熟悉 nodejs 的生态, 肯定已经安装好了

```
npm install @types/frida-gum
```

你需要在你编写注入 js 文件的目录下运行 ( 可以不事先创建 package.json，只是会出现一条警告 )

然后使用可以TypeScript代码完成功能的编辑器 ( 比如 vscode、pycharm、idea ) 打开js文件即刻。

**"基础 frida 代码完成"** 就可以补全 frida js 代码，如果想要 类成员函数及成员变量的类型等功能，可以安装插件 **frida-tsplugin**

**下载并安装插件 frida-tsplugin**

在任意目录下：**git clone https://github.com/tacesrever/frida-tsplugin** ，然后在 frida-tsplugin 目录下运行：

```
npm install
npm run compile
```

**frida-tsplugin 特性**

1. 可以识别 Java.use 和 Java.cast
2. 可以追踪变量赋值传递
3. 可以识别并追踪类成员函数及成员变量的类型
4. 可以根据重载函数的参数类型识别对应的重载函数
5. 可以识别 `someJavaFunction[.overload(...)].implementation = function(...) {...}` 函数块中的参数类型和this类型

ps. 对于未能追踪到的类型, 可以使用 Java.cast 来为其做一个声明

了解更多：http://www.yxfzedu.com/rs_show/115

## FridaContainer 脚本集分享

From：https://bbs.pediy.com/thread-265160.htm

FridaContainer 整合了网上流行的和平时常用的 frida 脚本，为逆向工作提效之用。

地址：https://github.com/deathmemory/FridaContainer

## 反反调试 anti-anti-debug

整理了一些普通反调试的绕过或定位的方法。因为很多反调试的手段是通过读取各个状态文件，查找特征字符串来判断是否被调试，而读取文件内容又通常都用到了 `fgets` 函数，如此我们就直接 hook 此函数加入过滤规则就能过滤掉许多反调试检测。

例如：
`/proc/<pid>/status` 检测 `TracerPid: 0` 、 `State: S (sleeping)` 、 `SigBlk: 0000000000001204` ，
`/proc/<pid>/stat` 检测 `t (tracing stop)`
`/proc/<pid>/wchan` 检测 `SyS_epoll_wait` 等

Hook fgets 后，触发时首先获取一下 LR 寄存器的值，保存一下现场，之后返回信息时可以把 LR 带出来，方便定位。然后调用原函数，对赋值的 buffer 进行过滤就可以了。

**FridaContainer 调用：FCAnd.anti.anti_debug();**

**小结：**当上面的方法无法绕过反调试时，可以再 Hook 一些常用的退出函数来定位反调点，比如 hook `exit`, `kill` ，再总结出一些其他过反调的方法，思路类似。

## anti-ssl-pinning

我们也在 FridaContainer 里面集成了 Frida 版的 JustTrustMe 来过 SSL Pinning 检测。

此部分代码主要借鉴了：https://codeshare.frida.re/@akabe1/frida-multiple-unpinning/

支持 20 种类库的SSL 验证绕过：

- TrustManager (Android < 7)
- TrustManagerImpl (Android > 7)
- OkHTTPv3 (quadruple bypass)
- Trustkit (triple bypass)
- Appcelerator Titanium
- OpenSSLSocketImpl Conscrypt
- OpenSSLEngineSocketImpl Conscrypt
- OpenSSLSocketImpl Apache Harmony
- PhoneGap sslCertificateChecker
- IBM MobileFirst pinTrustedCertificatePublicKey (double bypass)
- IBM WorkLight (ancestor of MobileFirst) HostNameVerifierWithCertificatePinning (quadruple bypass)
- Conscrypt CertPinManager
- CWAC-Netsecurity (unofficial back-port pinner for Android<4.2) CertPinManager
- Worklight Androidgap WLCertificatePinningPlugin
- Netty FingerprintTrustManagerFactory
- Squareup CertificatePinner [OkHTTP<v3] (double bypass)
- Squareup OkHostnameVerifier [OkHTTP v3] (double bypass)
- Android WebViewClient (double bypass)
- Apache Cordova WebViewClient
- Boye AbstractVerifier

**FridaContainer 调用：FCAnd.anti.anti_ssl_unpinning();**

**小结：**主要的出发点就是想摆脱 Xposed 框架，调试时减少被检测的风险，要绕过检测只绕 Frida 一个就好了。

## dump dex

这里的 dump dex 是 dump 二代壳，通过 hook art so 的 `art::DexFile::OpenCommon` (`_ZN3art7DexFile10OpenCommonEPKhjRKNSt3__112basic_stri` dump 动态加载的 dex 文件。通过 frida 动态加载自定义的 dex，在 dex 中实现类的枚举和主动加载，来尝试加载所有的类，使其触发 dex 动态加载，再通过之前的 hook ，将加载的 dex dump 下来。

**FridaContainer 调用：FCAnd.dump_dex_common();**

也有通过搜索内存的方式将 dex dump 出来 https://github.com/hluwa/FRIDA-DEXDump

## multi-dex

有时候遇到动态加载的 dex 又不是用的 application 的 loader，`Java.use` 可能会提示找不到类，遇到这种情况可以用修改 java loader 的方式来应对。
比如遇到利用 InMemoryDexClassLoader 来加载内存中的 dex 时，可以 Hook 它，当其触发时先走原流程，让其动态加载 dex，然后利用此时的 loader object 修改 `Java.classFactory.loader`，再使用 `Java.use` 来获取类就可以了，使用后记得恢复现场，否则会崩溃。

**动态加载实例：**

```
}
                        利用 InMemoryDexClassLoader 进行 dex 动态加载
@TargetApi(26)
private static ClassLoader createInMemoryClassLoader(Context arg7) throws IOException {
    InputStream is = arg7.getAssets().open("audience_network.dex");
    ByteArrayOutputStream os = new ByteArrayOutputStream();
    byte[] buffer = new byte[0x400];
    while(true) {
        int v3 = is.read(buffer);
        if(v3 <= 0) {
            break;
        }

        os.write(buffer, 0, v3);
    }

    is.close();
    os.flush();
    os.close();
    return new InMemoryDexClassLoader(ByteBuffer.wrap(os.toByteArray()), DynamicLoaderFactory.class.getClassLoader());
}
```

读取 dex 文件

InMemoryDexClassLoader 动态加载

CSDN @擒贼先擒王

**完整代码：**

```
function anti_InMemoryDexClassLoader(callbackfunc) {
    //  dalvik.system.InMemoryDexClassLoader
    const InMemoryDexClassLoader = Java.use('dalvik.system.InMemoryDexClassLoader');
    InMemoryDexClassLoader.$init.overload('java.nio.ByteBuffer', 'java.lang.ClassLoader')
        .implementation = function (buff, loader) {
        this.$init(buff, loader);
        var oldcl = Java.classFactory.loader;
        Java.classFactory.loader = this;
        callbackfunc();
        Java.classFactory.loader = oldcl; // 恢复现场
    }
}
```

**使用：**

```
FCAnd.anti.anti_InMemoryDexClassLoader(function(){
    const cls = Java.use('find/same/multi/dex/class');
    ...
});
```

## frida 的 java 反射调用

有时 dex 使用了一些非常规的动态加载方式，找 loader 定位起来也比较麻烦。假设我们只想调用某 jni 函数，看输入输出值的话，还可以用 frida java 反射的方式调用。

**整体调用流程：**

```
// 获取 so 基址
var base = Module.findBaseAddress('libxxxx.so');
// 根据偏移获取 jni 函数地址
var jnifunc_ptr = libsgmainso.add(0xE729);
// 声明 jni 函数
var jnifunc = new NativeFunction(jnifunc_ptr, 'pointer', ['pointer', 'pointer', 'int', 'pointer']);
// ********* 拼装 obj *********
// JNIEnv
var env = Java.vm.getEnv();
// 调用
var retval = jnifunc(env.handle,  ptr(0),  10401,  obj);
```

**拼装 Obj：**

```
// staticVaMethod 实现 Integer.valueOf(7)
const Integer_jcls = env.findClass('java/lang/Integer');
const Integer_valueOf = env.getStaticMethodId(Integer_jcls, 'valueOf', '(I)Ljava/lang/Integer;');
const invorkeStaticOjbectMethod = env.staticVaMethod('pointer', ['int']);
var pIn2 = invorkeStaticOjbectMethod(env.handle, Integer_jcls, Integer_valueOf, 7);
// 利用 constructor | vaMethod 组装 HashMap obj
// new HashMap().put('INPUT', 'xxxxxxxxxx')
const HashMap_jcls = env.findClass('java/util/HashMap');
const invokeHashmap_constructor = env.constructor([]);
const HashMap_init = env.getMethodId(HashMap_jcls, '<init>', '()V');
var HashMap_obj = invokeHashmap_constructor(env.handle, HashMap_jcls, HashMap_init);
const HashMap_put = env.getMethodId(HashMap_jcls, 'put', '(Ljava/lang/Object;Ljava/lang/Object;)Ljava/lang/
const invokeOjbectMethod = env.vaMethod('pointer', ['pointer', 'pointer']);
invokeOjbectMethod(env.handle, HashMap_obj, HashMap_put, env.newStringUtf('INPUT'), env.newStringUtf('xxxxx
```

上面都是在反射调用的角度简单介绍了 `staticVaMethod`, `vaMethod`, `constructor` 的使用方法。

## trace java methods

用 Frida 也可以实现 java 层的 trace 。
因为基于 Frida 框架，如果直接 trace 所有的类效率太慢，也容易崩溃。所以这里是以白名单的方式实现的。核心方法就是枚举所有类，按过滤名单，匹配需要 trace 的类，Hook 目标类的所有方法（可指定），在方法被调用时，将其入参和返回值记录下来。

**核心逻辑：**遍历所有类，Hook 白名单中的类及方法。

**调用方式：**

```
/**
 * java 方法追踪
 * @param clazzes 要追踪类数组 ['M:Base64', 'E:java.lang.String'] ，类前面的 M 代表 match 模糊匹配，E 代表 equal
 * @param whitelist 指定某类方法 Hook 细则，可按白名单或黑名单过滤方法。
 *                  { '类名': {white: true, methods: ['toString', 'getBytes']} }
 * @stackFilter 按匹配字串打印堆栈。如果要匹配 bytes 数组需要十进制无空格字串，例如："104,113,-105"
 */
FCAnd.traceArtMethods(
    ['M:MainActivity', 'E:java.lang.String'],
    {'java.lang.String': {white: true, methods:['substring', 'getChars']}},
    "match_str_show_stacks"
);
```

- 支持精确/模糊匹配类名
- 支持某类按白名单方式 trace 方法
- 支持匹配到指定值时收集栈信息

**具体实现：**

```
Java.enumerateLoadedClassesSync().forEach((curClsName, index, array) => {
    dest_cls.forEach((destCls) => {
        // 按规则匹配是否需要 trace
        if (match(destCls, curClsName)) {
            // trace 核心方法
            traceArtMethodsCore(curClsName);
            return false; // end forEach
        }
    });
});
// Hook 核心逻辑
function traceArtMethodsCore(clsname: string) {
    let cls = Java.use(clsname);
    // 枚举方法
    let methods = cls.class.getDeclaredMethods();
    methods.forEach(function (method: any) {
        ...
        // 枚举重载
        let methodOverloads = cls[methodName].overloads;
        methodOverloads.forEach(function (overload: any) {
            ...
            // Hook
            overload.implementation = function () {
                // ... send entry msg
                // 利用 js 参数特性 arguments ，调用原函数以适配所有 Hook 方法的传参
                const retval = this[methodName].apply(this, arguments);
                // ... send exit msg
                return retval;
            }
        }
    }
}
```

通过 `python/android/traceLogCleaner.py` 脚本收集 trace 日志，将回传的日志按线程、格式化输出日志，并且对字节数组，尝试进行 string 和 hex 转换以方便搜索。

**格式化 trace 效果：**

```
{"tid": 10421, "status": "exit", "tname": "Statistics-WriteData#0", "classname": "java.lang.String", "method": "pu
[+] (exit) java.lang.String
|- public byte[] java.lang.String.getBytes(java.nio.charset.Charset)
|= [103, 105, 107, 109, 66, 70, 66, 70, 38, 56, 63, 79, 65, 66, 67, 68]
|(str)== "gikmBFBF&8?OABCD"
|(hex)== 67,69,6b,6d,42,46,42,46,26,38,3f,4f,41,42,43,44

{"tid": 10421, "status": "exit", "tname": "Statistics-WriteData#0", "classname": "java.lang.String", "method": "pu
[+] (exit) java.lang.String
|- public byte[] java.lang.String.getBytes(java.nio.charset.Charset)
|= [103, 105, 107, 109, 66, 70, 66, 70, 38, 56, 63, 79, 65, 66, 67, 68]
|(str)== "gikmBFBF&8?OABCD"
|(hex)== 67,69,6b,6d,42,46,42,46,26,38,3f,4f,41,42,43,44

{"tid": 10421, "status": "entry", "tname": "Statistics-WriteData#0", "classname": "javax.crypto.Cipher", "method":
[+] (entry) javax.crypto.Cipher
|- public static final javax.crypto.Cipher javax.crypto.Cipher.getInstance(java.lang.String,java.security.Provider
|- AES/ECB/PKCS5Padding

{"tid": 10421, "status": "entry", "tname": "Statistics-WriteData#0", "classname": "javax.crypto.Cipher", "method":
[+] (entry) javax.crypto.Cipher
|- static final javax.crypto.Cipher javax.crypto.Cipher.createCipher(java.lang.String,java.security.Provider) thro
|- AES/ECB/PKCS5Padding
|- None
```

CSDN @擒贼先擒王

单条日志：


```
⊟{
    "status":"exit",
    "tryval":⊟{
        "p0":⊟{
            "trystr":"nvnetwork real url :https://[illegible]n?
dataCount=1&post_id=9dd6f8b5-657b-4e27-95bd-7c01c352d92c",
            "tryhex":"6e,76,6e,65,74,77,6f,72,6b,20,72,65,61,6c,20,75,72,6c,20,3a,68,74,7
6d,61,70,69,2e,64,69,61,6e,70,69,6e,67,2e,63,6f,6d,2f,6d,61,70,69,2f,6d,6c,6f,67,2f,7a,6
e,3f,64,61,74,61,43,6f,75,6e,74,3d,31,26,70,6f,73,74,5f,69,64,3d,39,64,64,36,66,38,62,35,
,34,65,32,37,2d,39,35,62,64,2d,37,63,30,31,63,33,35,32,64,39,32,63"
        }
    },
    "stacks":"java.lang.Exception
at java.lang.String.getBytes(Native Method)
at java.lang.String.getBytes(String.java:925)
at java.lang.String.getBytes(Native Method)
at com.[illegible]g.a.d$2.run(Unknown Source:40)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1162)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:636)
at java.lang.Thread.run(Thread.java:764)
",
    "classname":"java.lang.String",
    "tname":"pool-3-thread-1",
    "tid":22079,
    "retval":⊞Array[121],
    "method":"public byte[] java.lang.String.getBytes(java.nio.charset.Charset)"
}
```


CSDN @擒贼先擒王

小结：

- 优点：用 Frida 做 java 方法级的 trace ，优点就是方便、灵活、轻量级。
- 缺点：限于 Frida 框架，该方式效率较低，trace 方法过多容易崩溃，所以无法做全量 trace。

推荐用于轻量级的 Java 方法 trace 可以有效的定位核心算法。

## jni hook & trace

利用 Frida 的 `Java.vm.getEnv()` 获取 `JNIEnv` 指针，再根据 jni 结构体函数偏移，可以获取到各个 jni 函数地址，之后就可以根据需要进行 Hook 了。
例如

- 可以将其封装成更便捷的获取各 jni 函数地址的功能，方便 Hook

**核心逻辑：**

```
// 列出 JNI 函数数组
const jni_struct_array = [
    "reserved0",
    "reserved1",
    "reserved2",
    "reserved3",
    "GetVersion",
    "DefineClass",
    "FindClass",
    "FromReflectedMethod",
    ...
];
// 获取 JNIEnv 地址
var env = Java.vm.getEnv();
var env_ptr = env.handle.readPointer();
// 根据函数名计算索引偏移
var offset = jni_struct_array.indexOf(func_name) * Process.pointerSize;
// 读取函数地址
jnienv_addr.add(offset).readPointer();
// Hook
Interceptor.attach(addr, callbacksOrProbe);
```

**应用：**

```
FCAnd.jni.hookJNI('NewStringUTF', {
    onEnter: function (args) {
      ...
    }
});
```

- 可以 Hook `RegistNatives` 来获取动态注册的 jni 函数地址

```
export function hook_registNatives() {
    const tag = 'fridaRegstNtv';
    Jni.hookJNI("RegisterNatives", {
        onEnter: function (args) {
            var env = Java.vm.getEnv();
            var p_size = Process.pointerSize;
            var methods = args[2];
            var methodcount = args[3].toInt32();
            // 获取类名
            var name = env.getClassName(args[1]);
            DMLog.i(tag, "==== class: " + name + " ====");
            DMLog.i(tag, "==== methods: " + methods + " nMethods: " + methodcount + " ====");
            /** 根据函数结构原型遍历动态注册信息
             typedef struct {
                const char* name;
                const char* signature;
                void* fnPtr;
             } JNINativeMethod;
             jint RegisterNatives(JNIEnv* env, jclass clazz, const JNINativeMethod* methods, jint nMethods)
             */
            for (var i = 0; i < methodcount; i++) {
                var idx = i * p_size * 3;
                var fnPtr = methods.add(idx + p_size * 2).readPointer();
                const module = Process.getModuleByAddress(fnPtr);
                if (module) {
                    const modulename = module.name;
                    const modulebase = module.base;
                    var logstr = "name: " + methods.add(idx).readPointer().readCString()
                        + ", signature: " + methods.add(idx + p_size).readPointer().readCString()
                        + ", fnPtr: " + fnPtr
                        + ", modulename: " + modulename + " -> base: " + modulebase;
                    if (null != modulebase) {
                        logstr += ", offset: " + fnPtr.sub(modulebase);
                    }
                    DMLog.i(tag, logstr);
                }
                else {
                    DMLog.e(tag, 'module is null');
                }
            }
        }
    });
}
```

返回效果

```
==== class: com.xxxx.class.name ====
==== methods: 0xcd52d428 nMethods: 41 ====
[INFO][fridaRegstNtv]: name: initialize, signature: ()V, fnPtr: 0xcd50b6bd, modulename: libxxxx.so -> base:
[INFO][fridaRegstNtv]: name: onExit, signature: ()V, fnPtr: 0xcd50b6c7, modulename: libxxxx.so -> base: 0xc
[INFO][fridaRegstNtv]: name: getMMKVWithID, signature: (Ljava/lang/String;ILjava/lang/String;)J, fnPtr: 0xc
[INFO][fridaRegstNtv]: name: encodeBool, signature: (JLjava/lang/String;Z)Z, fnPtr: 0xcd50b76d, modulename:
[INFO][fridaRegstNtv]: name: decodeBool, signature: (JLjava/lang/String;Z)Z, fnPtr: 0xcd50b7bf, modulename:
[INFO][fridaRegstNtv]: name: encodeInt, signature: (JLjava/lang/String;I)Z, fnPtr: 0xcd50b80f, modulename:
[INFO][fridaRegstNtv]: name: decodeInt, signature: (JLjava/lang/String;I)I, fnPtr: 0xcd50b85b, modulename:
[INFO][fridaRegstNtv]: name: encodeLong, signature: (JLjava/lang/String;J)Z, fnPtr: 0xcd50b8a5, modulename:
[INFO][fridaRegstNtv]: name: decodeLong, signature: (JLjava/lang/String;J)J, fnPtr: 0xcd50b8f7, modulename:
[INFO][fridaRegstNtv]: name: encodeFloat, signature: (JLjava/lang/String;F)Z, fnPtr: 0xcd50b953, modulename
......
```

**FridaContainer 调用：FCAnd.jni.hook_registNatives();**

此功能因功能比较固定、使用频率高，也单独拆出了一个仓库：https://github.com/deathmemory/fridaRegstNtv

- 另外 jni 的部分，还可以做成 jni trace

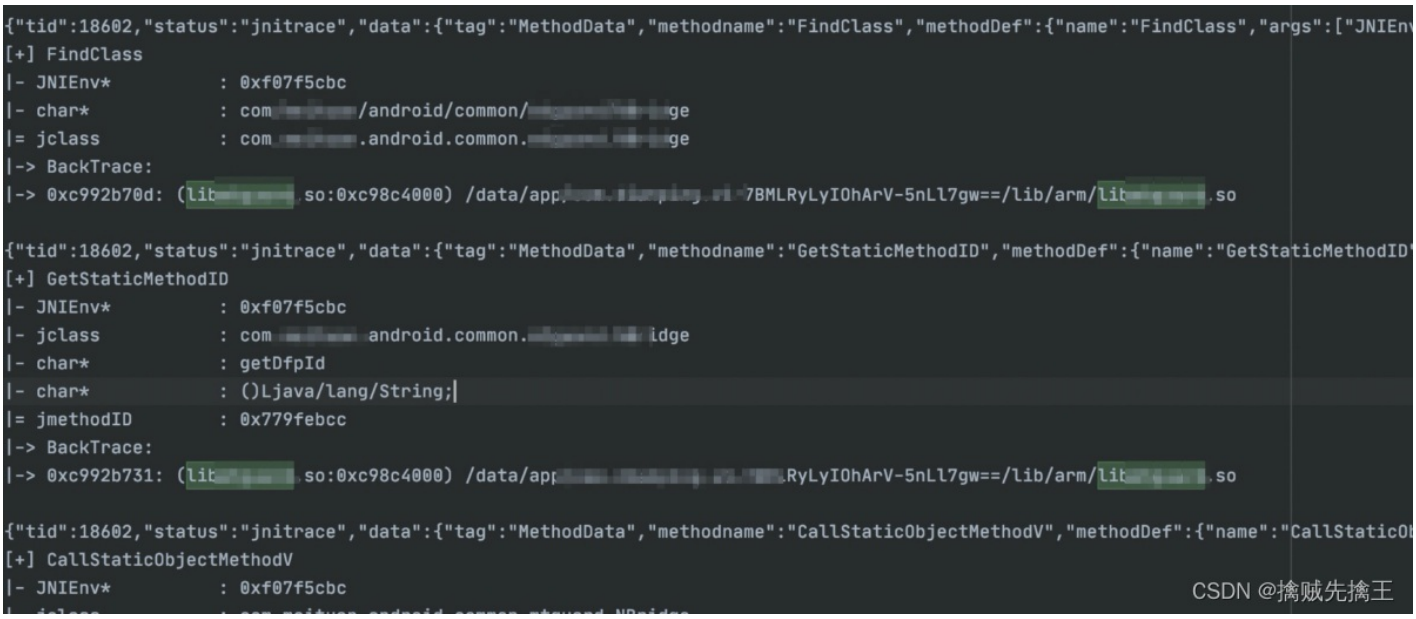
这里参考了 https://github.com/chame1eon/jnitrace 做了简化和嵌入版。

**具体实现：**因为有了上面的基础，使 Jni 的 trace 变得简洁了

```
export function traceAllJNISimply() {
    // 遍历 Hook Jni 函数
    jni_struct_array.forEach(function (func_name, idx) {
        if (!func_name.includes("reserved")) {
            Jni.hookJNI(func_name, {
                onEnter(args) {
                    // 触发时将信息保存到对象中
                    let md = new MethodData(this.context, func_name, JNI_ENV_METHODS[idx], args);
                    this.md = md;
                },
                onLeave(retval) {
                    // 退出时将返回值追加到对象中
                    this.md.setRetval(retval);
                    // 发送日志
                    send(JSON.stringify({tid: this.threadId, status: "jnitrace", data: this.md}));
                }
            });
        }
    })
}
```

因为性能问题，在信息收集时去除了 `jnitrace` 中的 `Thread.backtrace` 和 `DebugSymbol.fromAddress` 可能会造成线程阻塞的因素，对 trace 效果没有太多影响。

**格式化日志输出效果：**




**FridaContainer 调用：FCAnd.jni.traceAllJNISimply();**

参考：https://github.com/chame1eon/jnitrace

小结：jni 的 trace 做了速度上的优化，去除了一些线程阻塞的因素，使得在 jni 全量 trace 下也可以很好的运行。结合上面的 Java trace ，寻常难度的算法定位效率可以有一个很好的提高。

## Stalker 的应用

Stalker 部分的内容，bmax 大佬已经发了一篇贴子，大家可以跳转看一下。
地址：https://bbs.pediy.com/thread-264680.htm

## Frida 的检测方式

1. 文件名 `frida-agent**`
2. 默认端口 `27042`
3. 特征字符 `frida:rpc`、`LIBFRIDA`
4. 端口应答特征

```
if (connect(sock , (struct sockaddr*)&sa , sizeof sa) != -1) {
  memset(res, 0 , 7);

  send(sock, "\x00", 1, NULL);
  send(sock, "AUTH\r\n", 6, NULL);

  usleep(100); // Give it some time to answer

  if ((ret = recv(sock, res, 6, MSG_DONTWAIT)) != -1) {
    if (strcmp(res, "REJECT") == 0) {
      __android_log_print(ANDROID_LOG_VERBOSE, APPNAME,  "FRIDA DETECTED [1] - frida server running on port
    }
  }
}
```

参考：**https://github.com/b-mueller/frida-detection-demo/blob/master/AntiFrida/app/src/main/cpp/native-lib.cpp**

在逆向工程里，有两个方面对逆向的提效是帮助非常大的，一个是 trace 一个是算法识别。而 Frida 在有了 Stalker 的加持下，这两个方面都可以实现。在轻量级和便捷性上，首选推荐。

- FridaContainer：https://github.com/deathmemory/FridaContainer
- fridaRegstNtv：https://github.com/deathmemory/fridaRegstNtv

上面是两个仓

## Frida 使用

Frida 使用：https://zhuanlan.zhihu.com/p/339504595

- 1. 打印某个类的所有成员变量
- 2. 获取成员变量的值
- 3. 打印调用堆栈

frida 两种启动 hook 方式：

- 1. attach 进程名 ：APP 启动后再 hook，不能 hook app 启动阶段
- 2. spawn ：重启 APP，适合 hook app 启动阶段

frida 自定义端口

```
/data/local/tmp/fs64 -l 0.0.0.0:1234
adb forward tcp:1234 tcp:1234
frida -H 127.0.0.1:1234 package_name -l hook.js
```

通过 wifi 连接

```
adb devices
adb connect 192.168.1.xxx
adb forward tcp:27042 tcp:27042
adb forward tcp:27043 tcp:27043
```

连接设备

```
# 从usb设备连接frida-server，参数0表示不限等待时间，没有这个参数容易出现找不到设备
# device = frida.get_usb_device(0)

# 从TCP连接frida-server，指定IP和端口
# device = frida.get_device_manager().add_remote_device("192.168.31.176:8002")

# 以spawn方式启动app并在最早时机阻塞，后续可通过attach后resume恢复app执行流程
# pid = device.spawn(["test"])
# session = device.attach(pid)
# session = device.attach("com.eg.android.AlipayGphone")
# device.resume(pid)

# attach附加不影响app执行流程，get_frontmost_application可以获取到最前台的app，懒得去找进程号进程名
# app = device.get_frontmost_application()
# session = device.attach(app.pid)
```

hook 签名校验，定位签名校验位置、然后修改 smali过签名校验

```
import frida, sys


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


test_sig = '''
Java.perform(
    function(){
        var Signature = Java.use('android.content.pm.Signature')
        Signature.hashCode.implementation = function() {
            console.log('hashCode')
            return this.hashCode()
        }

        Signature.toByteArray.implementation = function() {
            console.log('toByteArray')
            printstack()
            return this.toByteArray()
        }

        function printstack() {
            console.log(Java.use("android.util.Log").getStackTraceString(Java.use("java.lang.Exception").$n
        }

        var AA = Java.use('com.chaozhuo.texteditor.widget.a')
        AA.a.overload('android.content.Context').implementation = function() {
            return true
            //this.a()
        }
    }
)

'''


test_context = '''
Java.perform(
    function(){
        var currentApplication = Java.use('android.app.ActivityThread').currentApplication()
        var context = currentApplication.getApplicationContext()

        console.log(context.getPackageName())

        console.log(context.getPackageManager().getPackageInfo(context.getPackageName(), 64).signatures.val

        send(context.getPackageManager().getPackageInfo(context.getPackageName(), 64).signatures.value[0].t

        console.log(context.getPackageManager().getPackageInfo(context.getPackageName(), 64).signatures.val
    }
)
'''


# 两种启动方式
```

```
# 启动方式 1
# process = frida.get_usb_device(-1).attach('com.chaozhuo.texteditor')
# script = process.create_script(test_context)
# script.on('message', on_message)
# script.load()
# sys.stdin.read()


# 启动方式 2
# spawn 重启APP 可以hook APP启动阶段
device = frida.get_usb_device(-1)
pid = device.spawn(['com.chaozhuo.texteditor'])
process = device.attach(pid)

script = process.create_script(test_context)
script.on('message', on_message)
print('[*] Running')
script.load()
device.resume(pid)
sys.stdin.read()
```

frida Hook Java 类 小提示：

- **访问 成员变量 写法：this.成员变量名.value**
- **hook 匿名类 写法：Java.use('类$类')　// smali文件里找**
- **从 匿名类 / 内部类 访问 外部类 的 属性 写法：this.this$0.value.外部类的属性名.value**
- **新建一个对象，并 调用构造函数 的写法：类.方法名.$new(参数)**
- hook 类的方法：类.方法名.implementation
- **hook 重载写法：类.方法名.overload(参数1，参数2.....).implementation**
- **hook 构造方法：类.$init().implementation**

## java 层 hook

- 1. hook模板
- 2. hook方法（非重载方法不用写方法类型）
- 3. hook重载方法
- 4. hook所有重载方法
- 5. hook 构造方法
- 6. 对象实例化
- 7. 修改类的字段
- 8. hook内部类与匿名类
- 9. hook类的所有方法
- 10. Hook动态加载的dex(Android 7以上)
- 11. Java特殊类型的遍历与修改（Map举例）
- 12. 打印HashMap
- 13. Java层主动调用
- 14. 删除对象引用
- 15. 获取参数类型
- 16. 用frida注入dex文件
- 17. 端口检测解决方案
- 18. frida启动前注入

## so 层 hook

- 1. 枚举导入导出表(ELF即so文件)
- 2. hook导出函数
- 3. 函数地址计算
- 4. Hook未导出函数
- 5. 获取指针参数返回值
- 6. Hook_dlopen
- 7. 内存读写
- 8. 主动调用JNI函数
- 9. jni函数Hook(计算地址方式)
- 10. jni函数Hook(libart.so)
- 11. so层函数主动调用
- 12. frida读写文件

## RPC

```
import frida
import sys

rdev = frida.get_usb_device()
session = rdev.attach("com.yuanrenxue.onlinejudge2020")  # 包名

js_code = """
rpc.exports = {
    getsign: function(i) {
        Java.perform(function() {
            console.log("get_sign");
            var my_class1 = Java.use("com.yuanrenxue.onlinejudge2020.OnlineJudgeApp");
            var reslut = my_class1.getSign1(i);
            console.log(reslut);
            send({ "sign": reslut, "num": i })
            return reslut;
        });
    },
};

"""

script = session.create_script(js_code)


def on_message(message, data):
    sign = message.get("payload").get("sign")
    num = message.get("payload").get("num")


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

script.exports.getsign(1) # 调用的函数

sys.stdin.read()
```

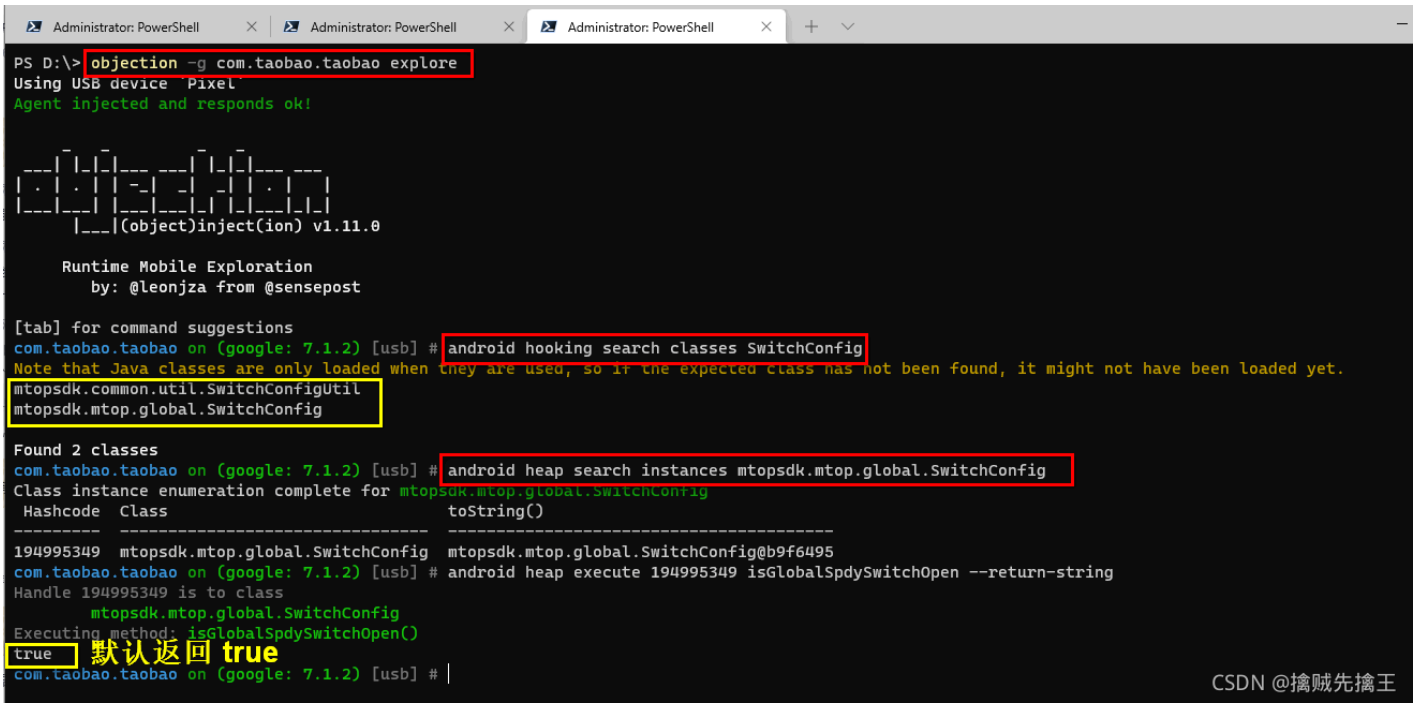
### frida rpc 枚举 实现接口 的 类

```
import sys
import frida

'''
frida rpc 枚举类
'''



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


hook = """
Java.perform(function(){
    Java.enumerateLoadedClasses({
        "onMatch" : function(classname){
            if(classname.indexOf("com.csair.mbp") < 0){
                return;
            }
            // 实现类 implements
            try{
                var hookCls = Java.use(classname)
                var interFaces = hookCls.class.getInterfaces();
                if (interFaces.length > 0) {
                    console.log(classname)
                    for (var i in interFaces) {
                        // 接口类 interFaces
                        console.log("\t", interFaces[i].toString())
                    }
                }
                }catch(e){
                    console.log(e)
                }
            },
        "onComplete" : function(){}
    })
})
"""

process = frida.get_usb_device().attach('com.csair.mbp')
script = process.create_script(hook)
script.on('message', on_message)
print('[*] Running CTF')
script.load()
sys.stdin.read()
```

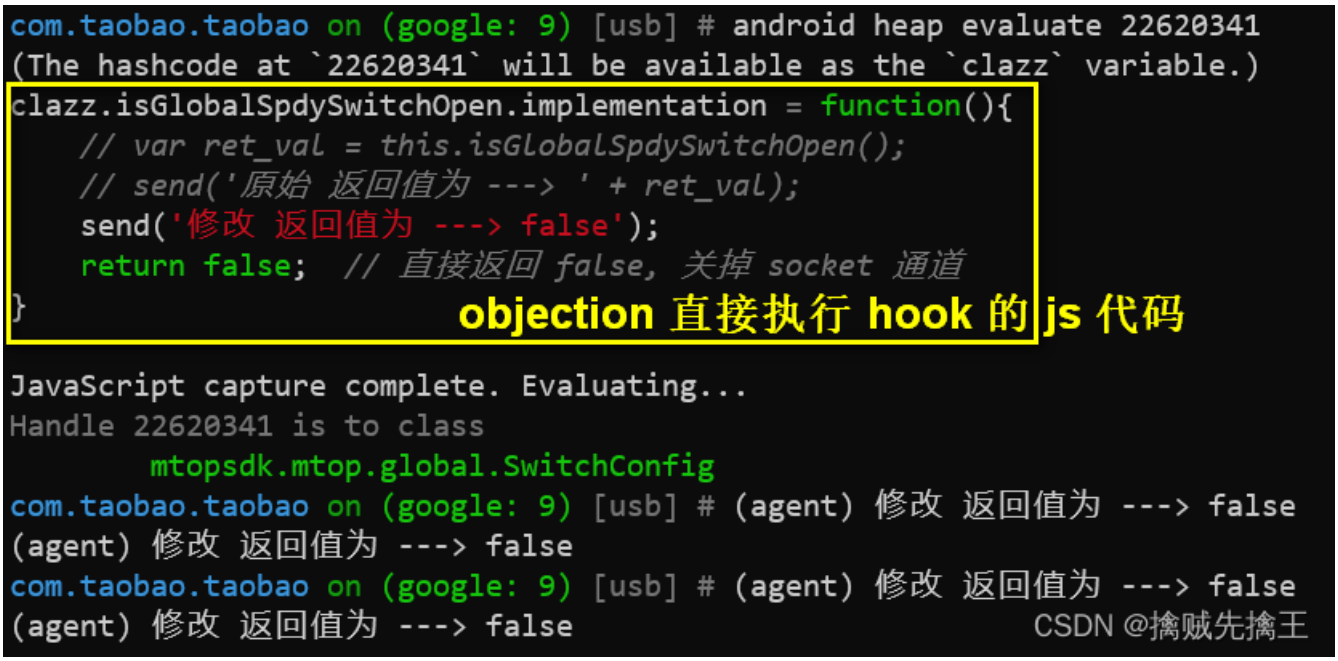
方法 2：

```
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)
```

hook tcp 和 http 开关（ 手淘 ）：




通过 objection 直接执行 hook 的 js 代码



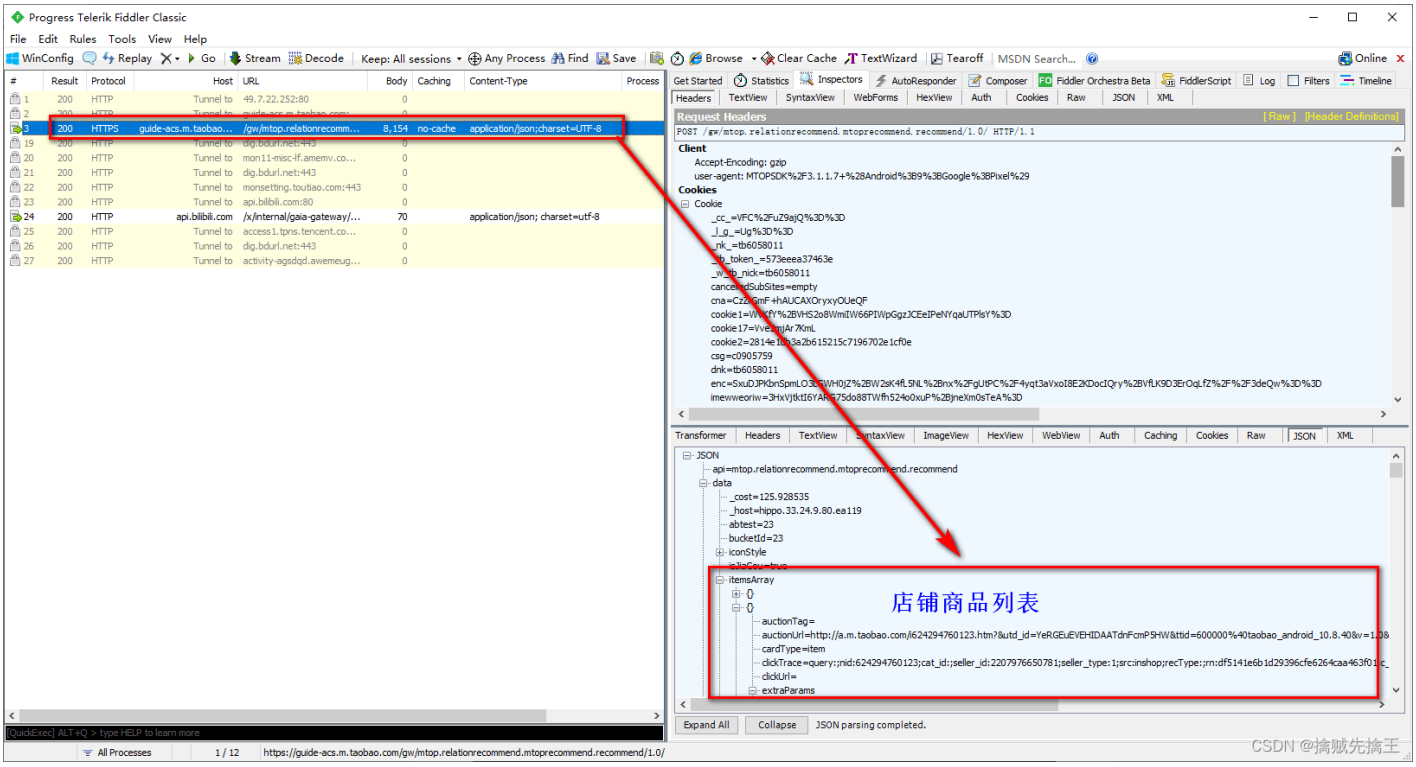

通过 python 代码执行 hook：

```
import sys
import frida

hook_code = '''
Java.perform(
    function(){
        send("开始 hook");
        var SwitchConfig = Java.use('mtopsdk.mtop.global.SwitchConfig')
        SwitchConfig.isGlobalSpdySwitchOpen.implementation = function(){
            // var ret_val = this.isGlobalSpdySwitchOpen();
            // send('原始 返回值为 ---> ' + ret_val);
            send('修改 返回值为 ---> false');
            return false;  // 直接返回 false, 关掉 socket 通道
        }
    }
)
'''


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


# device = frida.get_usb_device()
device = frida.get_remote_device()

# 直接使用 包名 运行报错，就使用 pid 号 进行 attach
# process = device.attach('com.taobao.taobao')
process = device.attach('淘宝')
script = process.create_script(hook_code)
script.on('message', on_message)
script.load()
sys.stdin.read()
```

运行截图：

```
[*] 开始 hook
[*] 修改 返回值为 ---> false
[*] 修改 返回值为 ---> false
[*] 修改 返回值为 ---> false
```

抓包结果截图：




通过 wifi 连接，然后用 adb 实现群控

```
import frida
import os
import time

app = "com.xxx.xxx";

device_ids = [];
devices = frida.enumerate_devices();
for device in devices :
    # print(device)
    ## 枚举所有通过wifiadb 连的机器
    if device.id.find(":") > 0:
        #print(device.id)
        device_ids.append(device.id.replace("5555", "9999"))

for id in device_ids :
    print(id)
    device = frida.get_device_manager().add_remote_device(id)
    print(device)
    pid = device.spawn([app])
    print(pid)
    device.resume(pid)
    time.sleep(1)
    session = device.attach(pid)
    with open("load_hook.js") as f:
        script = session.create_script(f.read())
        script.load()
input()
常用命令
```

## 使用群控软件进行群控

掘金网安卓群控系统：https://www.54nb.com/androids/index.html

## sekiro全家桶(sekiro 群控不迷路)

：https://github.com/langgithub

- sekiro-ios https://github.com/langgithub/SekiroIOS
- sekiro-Android https://github.com/langgithub/sekiro-lang
- sekiro-python https://github.com/langgithub/SekiroPython
- sekiro-java https://github.com/langgithub/unidbg-lang
- sekiro-unidbg https://github.com/langgithub/unidbg-lang
- sekiro-frida https://github.com/langgithub/FridaInject
- sekiro-xposed https://github.com/langgithub/hello
- sekiro-chrome https://github.com/langgithub/frida_app_hook/blob/master/baoxianshi.html

## 开源群控软件（ 推荐 ）

**QtScrcpy：https://github.com/barry-ran/QtScrcpy**


可以根据 github 上源码，编译 linux 版本。网上编译好的 Linux 版本。

## 常用命令

- frida-ps -U　　列出正在运行的进程
- frida-ps -Uai　　列出安装的程序
- frida-ps -Ua　　列出运行中的程序（查看包名很方便）
- frida-ps -D 设备id 连接frida到一个指定的设备上
- 另外还有四个分别是： frida-trace, frida-discover, frida-ls-devices, frida-kill

## objection 使用

Frida 只是提供了各种 API 供我们调用，在此基础之上可以实现具体的功能，比如禁用证书绑定之类的脚本，就是使用 Frida 的各种 API 来组合编写而成。于是有大佬将各种常见、常用的功能整合进一个工具，供我们直接在命令行中使用，这个工具便是 objection。

> 安装：**pip3 install objection**

连接 app

- objection -g 包名 explore

Memory 指令

- memory list modules // 查看内存中加载的库
- memory list exports libssl.so // 查看库的导出函数
- memory list exports libart.so --json /root/libart.json //将结果保存到json文件中
- memory search --string --offsets-only //搜索内存
- memory search "64 65 78 0a 30 35 00"

root

- android root disable // 尝试关闭app的root检测
- android root simulate // 尝试模拟root环境

activities

- android hooking list activities // 可以列出app具有的所有avtivity

内存漫游

```
//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 [search_name]

//在内存中所有已加载的方法中搜索包含特定关键词的方法
android hooking search methods [search_name]

//直接生成hook代码
android hooking generate simple [class_name]

// 查看类的全部广法
android hooking list class_methods [class_name]
```

hook 方式。hook指定方法，如果有重载会 hook 所有重载。

- --dump-args : 打印参数
- --dump-backtrace : 打印调用栈
- --dump-return : 打印返回值

```
// 查看方法的参数、返回值和调用栈
android hooking watch class_method com.xxx.xxx.methodName --dump-args --dump-backtrace --dump-return

//获取全部tostring的返回来值
android hooking watch class_method java.lang.StringBuilder.toString --dump-return

//弹窗
android hooking watch class_method android.app.Dialog.show --dump-args --dump-backtrace --dump-return

//hook指定类，会打印该类下的所有调用
android hooking watch class com.xxx.xxx

//设置返回值(只支持bool类型)
android hooking set return_value com.xxx.xxx.methodName false
```

**关闭 app 的 ssl 校验：android sslpinning disable**

**Spawn 方式 Hook （ app 启动之前就 hook ）**

从Objection 的使用操作中我们可以发现，Obejction 采用 Attach 附加模式进行 Hook，这可能会让我们错过较早的 Hook 时机，可以通过如下的代码启动 Objection，引号中的 objection 命令会在启动时就注入App。命令：**objection -g packageName explore --startup-command 'android hooking watch xxx'**

## ARIDA （ 管理PRC脚本，自动生成 http 接口的工具 ）

**1. 安装**

下载：git clone git@github.com:lateautumn4lin/arida.git

**使用 conda 安装**

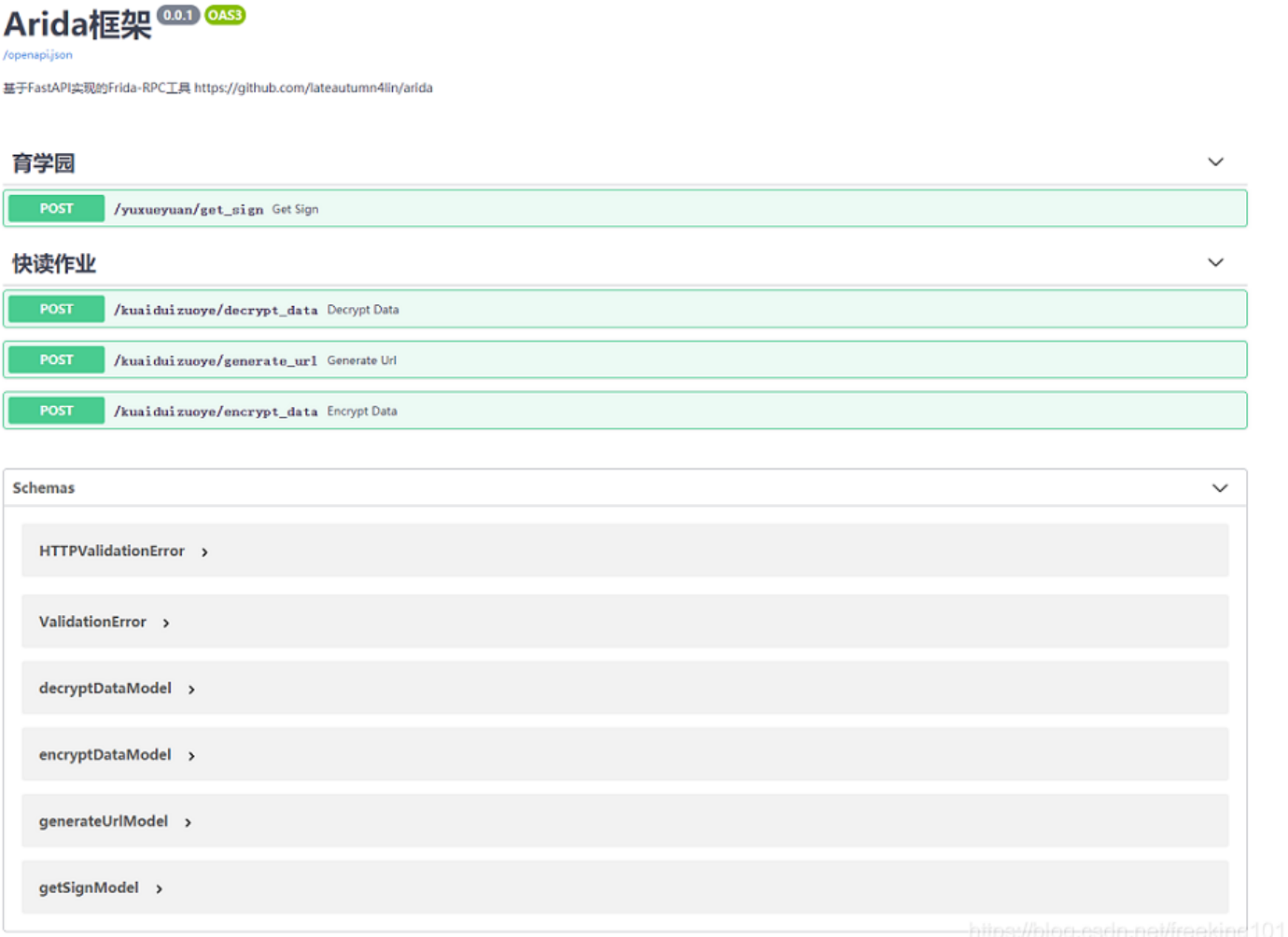
```
conda create -n arida python==3.8
conda install --yes --file requirements.txt
```

使用 pip 安装

```
virtualenv venv
source venv/bin/activate

// 下载 pip 安装格式的 requirements.txt
//  https://github.com/Boris-code/arida/blob/master/requirements.txt

pip install -r requirements.txt
```

2. 运行

```
uvicorn main:app --reload

watch 127.0.0.1:8000/docs
```




3. 开发

Config文件中写入自己的App信息

apps 目录写开发相应的Frida-Js脚本，可参考其他两个文件

## 参考：（objection、Firda API、ARIDA）

Objection 的安装和简单使用：https://www.cnblogs.com/qiaorui/p/13455420.html
Firda API 示例：https://juejin.cn/post/6844904127705661448#heading-9
ARIDA：https://github.com/lateautumn4lin/arida

# 1. Frida Hook Android 常用方法

Android 常用 Hook 方法汇总

- Hook 一般函数 --- 使用 implementation
- Hook 重载函数 --- 使用 overload
- Hook 构造函数 --- 使用 $init
- Hook 生成对象 --- 使用 $new
- Hook 内部类 --- 使用 $
- Hook native 函数
- Hook 静态变量（属性）/ 参考文章中有反射的方法
- 打印堆栈
- byte 转 String
- String 转 Uint8Array
- int 转 bytes
- String 转 ArrayBuffer
- ArrayBuffer 转 String
- 添加时间戳
- byte 转 Hex
- Hex 转 byte

## 1、概述

在逆向过程中，Frida 是非常常用的 Hook 工具，这个工具在日常使用的过程中，有很多通用方法，这里记录一下，方便查阅，部分函数使用的时候，可能需要稍微修改一下。
主要是搬迁的一下参考文章的片段
Android逆向之旅—Hook神器家族的Frida工具使用详解
Frida官方文档
看雪 Frida官方手册 - JavaScript API（篇一）
看雪 Frida官方手册 - JavaScript API（篇二）
Js中几种String Byte转换方法
Frida learn by example
补充一个参考文章–Hook 属性

## 2、Hook Java 代码篇

From：https://www.52pojie.cn/forum.php?mod=viewthread&tid=931872

Frida 文档：Welcome | Frida • A world-class dynamic instrumentation framework

Frida 从入门到入门 --- 安卓逆向菜鸟的 frida 使用说明：[原创]Frida从入门到入门—安卓逆向菜鸟的frida食用说明-Android安全-看雪论坛-安全社区|安全招聘|bbs.pediy.com

frida入门总结：https://www.52pojie.cn/thread-1128884-1-1.html
**********
Ta的论坛
初识Frida--Android逆向之Java层hook (一)
初识Frida--Android逆向之Java层hook (二)
进阶Frida--Android逆向之动态加载dex Hook（三）（上篇）
进阶Frida--Android逆向之动态加载dex Hook（三）（下篇）
**********

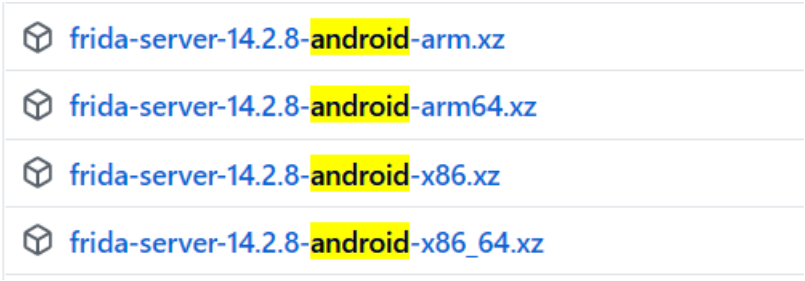
frida分 **客户端环境** 和 **服务端环境**。

- 在 **客户端**，我们可以编写 Python 代码，用于连接远程设备，提交要注入的代码到远程，接受服务端的发来的消息等。
- 在 **服务端**，我们需要用 Javascript 代码注入到目标进程，操作内存数据，给客户端发送消息等操作。
- 可以把 **客户端理解成控制端，服务端理解成被控端**。假如我们要用PC来对Android设备上的某个进程进行操作，那么PC就是客户端，而Android设备就是服务端。

### Frida 服务端 环境

> 注：Windows 系统执行命令可以在 CMD 中进行；Linux 和 MacOS 执行命令可以在终端中进行。adb 是 Android一个调试工具，具体安装方法不是本文的重点。

- 根据自己的平台下载 frida 服务端并解压：Releases · frida/frida · GitHub




- 执行命令，将 **服务端** 推到手机的 **/data/local/tmp** 目录：**adb push frida-server /data/local/tmp/frida-server**
- 执行命令，修改 frida-server 文件权限：**adb shell chmod 777 /data/local/tmp/frida-server**
- 运行 frida-server：**./frida-server &**

### Frida 客户端 环境

- 在 PC 上安装 Python 环境（ 推荐 Python3 ），安装完成后执行下面的命令安装 frida：**pip install frida**
- 将脚本注入到 Android 目标进程：**frida -U -l myhook.js com.xxx.xxxx**
  **参数解释：**
  -U　指定对USB设备操作
  -l　指定加载一个Javascript脚本
  最后指定一个进程名，如果想指定进程 pid，用 -p 选项。正在运行的进程可以用 frida-ps -U 命令查看
  frida 运行过程中，执行 %resume 重新注入，执行 %reload 来重新加载脚本；执行 exit 结束脚本注入

Frida 客户端命令行的参数解释：

```
kali@kali:~$ frida -h
Usage: frida [options] target

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
  -f FILE, --file=FILE  spawn FILE
  -F, --attach-frontmost
                        attach to frontmost application
  -n NAME, --attach-name=NAME
                        attach to NAME
  -p PID, --attach-pid=PID
                        attach to PID
  --stdio=inherit|pipe  stdio behavior when spawning (defaults to “inherit”)
  --aux=option          set aux option when spawning, such as “uid=(int)42”
                        (supported types are: string, bool, int)
  --runtime=duk|v8      script runtime to use
  --debug               enable the Node.js compatible script debugger
  --squelch-crash       if enabled, will not dump crash report to console
  -O FILE, --options-file=FILE
                        text file containing additional command line options
  -l SCRIPT, --load=SCRIPT
                        load SCRIPT
  -P PARAMETERS_JSON, --parameters=PARAMETERS_JSON
                        parameters as JSON, same as Gadget
  -C CMODULE, --cmodule=CMODULE
                        load CMODULE
  -c CODESHARE_URI, --codeshare=CODESHARE_URI
                        load CODESHARE_URI
  -e CODE, --eval=CODE  evaluate CODE
  -q                    quiet mode (no prompt) and quit after -l and -e
  --no-pause            automatically start main thread after startup
  -o LOGFILE, --output=LOGFILE
                        output to log file
  --exit-on-error       exit with code 1 after encountering any exception in
                        the SCRIPT
kali@kali:~$                                  https://blog.csdn.net/freeking101
```

**启动方式 1：使用 js 脚本启动**

```
frida -U -l exploit.js -f com.package.name
```

其中 js 脚本的写作方式如下

```
setImmediate(function() { //prevent timeout
    console.log("[*] Starting script");

    Java.perform(function() {
      myClass = Java.use("com.package.name.xxActivity");
      myClass.implementation = function(v) {
         // do sth.
      }

    })
})
```

### 启动方式 2：使用 Python 脚本启动

```
import frida, sys

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

jscode = """
Java.perform(function () {
    // Function to hook is defined here
    var MainActivity = Java.use('com.example.hook.MainActivity');
    // hook method is setString
    MainActivity.setString.implementation = function (str) {
        // Show a message to know that the function got called
        send('hook success');
        console.log('string is: ' + str));
    };
});
"""

process = frida.get_usb_device().attach('com.example.hook')
#pid = device.spawn(["com.android.chrome"])
#session = device.attach(pid)
#device.resume(pid)
script = process.create_script(jscode)
script.on('message', on_message)
print('[*] Hook Start Running')
script.load()
sys.stdin.read()
```

示例 2：

```
# -*- coding: utf-8 -*-
# @Author  : 佛祖保佑, 永无 bug
# @Date    : 2021-03-24, 0024
# @File    : frida_test.py
# @Software: PyCharm
# @description : XXX
```

```
import sys
import frida

'''
/* frida hook 格式示例 */
console.log("Script loaded successfully ");
Java.perform(function x() {
    console.log("Inside java perform function");
    //定位类
    var my_class = Java.use("com.example.demo2.MainActivity");
    console.log("Java.Use.Successfully!");//定位类成功!
    //在这里更改类的方法的实现（implementation）
    my_class.fun.implementation = function(x,y){
        //打印替换前的参数
        console.log( "original call: fun("+ x + ", " + y + ")");
        //把参数替换成2和5，依旧调用原函数
        var ret_value = this.fun(2, 5);
        return ret_value;
    }
});
'''

# 从此处开始定义用来 Hook 的 javascript 代码
js_code_1 = """
console.log("Script loaded successfully ");

function get_android_version() {
    // 获取当前安卓设备的安卓版本
    var android_version = Java.androidVersion;
    // send('android version:' + android_version);
    console.log(android_version);
}

function get_load_classes() {
    //获取该应用加载的类
    var class_names = Java.enumerateLoadedClassesSync();
    for (var i = 0; i < class_names.length; i++){
        // send('[class_name]:' + class_names[i]);
        console.log('[class name]:' + class_names[i]);
    }
}

function change_return_value() {
    //获得 MainActivity 类
    var MainActivity = Java.use('com.yaotong.crackme.MainActivity');
    var java_string = Java.use('java.lang.String');
    MainActivity.securityCheck.implementation = function(java_string){
        send('I am here');      // 发送信息，用于回调python中的函数
        return true;            //劫持返回值，修改为我们想要返回的字符串
    }
}

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

    get_android_version();
    get_load_classes();
    change_return_value();
});
"""
```

```
"""

js_code_2 = '''
function hook_main_activity_onCreate() {
    var main_activity = Java.use('com.yaotong.crackme.MainActivity');
    main_activity.onCreate.overload('') = function () {
        send('I am here');
        this.onCreate()
    }
}
Java.perform(function(){
    hook_main_activity_onCreate()
})
'''

js_code_3 = '''
//var walk = Process.enumerateModules();
//for(var i=0; i<walk.length; i++){
//    send(walk[i].name);
//}
var base_address = Module.findBaseAddress('libc.so');
send('base_address:' + base_address);
var func_address = Module.findExportByName("libc.so", "dlopen");
send('func_address:' + func_address);
var lib_module = Process.findModuleByAddress(base_address);
send("lib_module_address:" + lib_module.name);
Interceptor.attach(func_address, {
    onEnter: function(args){
        //send("open(" + args[0] + "," + args[1] + ")" );
        send("open(" + Memory.readUtf8String(args[0]) + "," + args[1] + ")");
    },
    onLeave: function(ret_val){
        send("ret_val:" + ret_val);
    }
});
'''



def js_callback_func(msg, data):
    # js中执行send函数后要回调的函数
    if 'send' == msg['type']:
        print(f'[*] {msg["payload"]}')
    else:
        print(msg)


'''
# 使用 get_usb_device 函数来获取设备一直报错找不到设备，改用 get_remote_device 函数即可解决这个问题
# process = frida.get_usb_device().attach('com.yaotong.crackme')
'''



def test_1():
    # 要 hook 的 app 必须首先启动，才能开始 hook
    process = frida.get_remote_device().attach('com.yaotong.crackme')
    script = process.create_script(js_code_1)  # 创建js脚本
    script.on('message', js_callback_func)  # 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.load()  # 加载脚本
    sys.stdin.read()
```

```
def test_2():
    # 可以不启动 app 直接 hook，程序会自动启动 app
    devices = frida.get_remote_device()
    pid = devices.spawn(['com.yaotong.crackme'])
    process = devices.attach(pid)

    script = process.create_script(js_code_3)
    script.on('message', js_callback_func)
    print('[*] Running CTF')
    script.load()
    devices.resume(pid)
    sys.stdin.read()


if __name__ == '__main__':
    test_1()
    # test_2()
```

### Java.use 载入 Java类：

- Java.use 方法用于声明一个Java类，Java类 必须 先声明，后使用。
- 比如声明一个 String 类，要指定完整的类名：**var StringClass = Java.use("java.lang.String");**

### 函数 的 参数类型 表示

不同的参数类型都有自己的表示方法

对于基本类型，直接用它在Java中的表示方法就可以了，不用改变，例如：

- int
- short
- char
- byte
- boolean
- float
- double
- long

### 基本类型 的 数组：用 左中括号 接上 基本类型 的缩写

基本类型缩写表示表：

| 基本类型 | 缩写 |
| --- | --- |
| boolean | Z |
| byte | B |
| char | C |
| double | D |
| float | F |
| int | I |

| 基本类型 | 缩写 |
| --- | --- |
| long | J |
| short | S |

- 例如：int[] 类型，在重载时要写成 [I
- **任意类**，直接写 完整类名 即可。例如：java.lang.String
- **对象 数组**，用 左中括号 **接上** 完整类名 **再接上** 分号。例如：[java.lang.String;

### 带参数 的 构造函数

修改参数为 byte[] 类型的构造函数的实现

```
ClassName.$init.overload('[B').implementation=function(param){
    //do something
}
```

注：ClassName 是使用 Java.use定义的类，param 是可以在函数体中访问的参数

改多参数的构造函数的实现

```
ClassName.$init.overload('[B','int','int').implementation=function(param1,param2,param3){
    //do something
}
```

### 无参数 的 构造函数

```
ClassName.$init.overload().implementation=function(){
    //do something
}
```

调用原构造函数

```
ClassName.$init.overload().implementation=function(){
    //do something
    this.$init();
    //do something
}
```

> 注意：当 **构造函数** 或者 **函数** 有**多种重载**形式，比如一个类中有两个形式的 func：void func() 和 void func(int)，要加上 overload 来对函数进行重载，**否则可以省略overload**

### Hook 构造方法

Hook 构造方法时，与普通方法类似，只不过将方法名改成了 $init

```
// Hook构造方法时与普通方法类似，只不过将方法名改成了 $init
Java.perform(function () {
    var hook_cls = Java.use('com.threetails.demo.Utils');
    hook_cls.$init.implementation = function (a, b) {
        console.log("Hook Start...");

        send('arg0: '+ a);
        send('arg1: '+ b);

        return this.$init(a, b);
    }
});
```

### Hook 重载方法

```
Java.perform(function () {
    var hook_cls = Java.use('com.threetails.demo.Utils');

    /* ovldMethod是一个重载方法，必须加上overload(参数类型)才能Hook。
    若参数类型为字符串，应该传java.lang.String，因为字符串在java中与int不同，它不是基本类型，而是类。
    如果不知道参数类型，可以去掉overload，执行一下程序会抛出错误，在错误提示中可以看到对应的参数类型。*/
    hook_cls.ovldMethod.overload("int").implementation = function (a) {
        console.log("Hook Start...");

        send('arg0: '+ a);

        send('result: '+ this.ovldMethod.overload("int")(a));

        return this.ovldMethod.overload("int")(a);
    }
});
```

### 创建 类 的 实例，并调用函数

和 Java 一样，创建 **类实例** 就是调用 **构造函数**，用 **$new** 表示一个构造函数。

```
var ClassName=Java.use("com.luoye.test.ClassName");
var instance = ClassName.$new();
```

类实例化以后，就可以调用其他函数

```
var ClassName=Java.use("com.luoye.test.ClassName");
var instance = ClassName.$new();
instance.func();
```

### 构造对象参数

如果方法参数的类型并非 java 的内置类型，而是一个自定义类，你又想在Hook时传入自定义对象，怎么办？假设现在有个自定义类叫 Dog，它有两个属性：name 和 age。看看下面这个示例：

```
Java.perform(function () {
    var hook_cls = Java.use('com.threetails.demo.Utils');

    // 首先声明一个dog类
    var dog_cls = Java.use('com.threetails.demo.Dog');

    hook_cls.feed.implementation = function (dog_obj) {
        console.log("Hook Start...");
        send('old dog:'+dog_obj);

        // 调用dog_cls.$new来实例化一个自定义的dog对象
        var new_dog = dog_cls.$new("大黄", 5);
        send('new dog:'+new_dog);

        // 调用原方法并将对象参数替换为自定义的dog
        return this.feed(new_dog);
    }
});
```

### 修改对象属性

还是以 Dog 类为例， 现在我们需要将传入的 Dog 对象的 age 修改为 1。

### 正常修改

```
Java.perform(function () {
    var hook_cls = Java.use('com.threetails.demo.Utils');

    hook_cls.feed.implementation = function (dog_obj) {
        console.log("Hook Start...");
        // 需要使用.value来获取属性值
        send(dog_obj.age.value);

        // 需要使用.value来修改属性值
        dog_obj.age.value = 1;
        send(dog_obj.name.value);

        return this.feed(dog_obj);
    }
});
```

### 使用反射

修改对象属性还有另外一种方式，也就是使用 java 的反射， 这种方式相对复杂一些。

```
Java.perform(function () {
    var hook_cls = Java.use('com.threetails.demo.Utils');

    // 首先需要声明一个类构造器
    var clazz = Java.use('java.lang.Class');

    hook_cls.feed.implementation = function (dog_obj) {
        console.log("Hook Start...");

        // 调用Java.cast方法，传入类和类构造器得到类的id，再调用getDeclaredField得到属性id
        var ageid = Java.cast(dog_obj.getClass(),clazz).getDeclaredField('age');

        // 设置该属性为可访问
        ageid.setAccessible(true);

        // 调用get方法获取属性值
        var old_value = ageid.get(dog_obj);

        // 此处必须使用console.log才能打印我们需要的值
        console.log(old_value);

        // 调用setInt修改属性值
        ageid.setInt(dog_obj, 1);

        return this.feed(dog_obj);
    }
});
```

### Hook 普通方法（包括 私有、公有、静态）

```
Java.perform(function () {
    // Java.use(类的路径)，声明一个要hook的类
    var hook_cls = Java.use('com.threetails.demo.Utils');

    // 修改该类下normalMethod的实现，参数个数需要与原方法保持一致
    hook_cls.normalMethod.implementation = function (a, b) {
        console.log("Hook Start...");

        send('arg0: '+ a);
        send('arg1: '+ b);

        send('result: '+ this.normalMethond(a,b));

        /* 为了让程序正常运行，需要调用原方法将结果返回。若想修改参数值，可以在调用时直接将参数替换，
        如：return this.normalMethond(1,2)，但需注意参数类型的一致性。*/
        return this.normalMethond(a,b);
    }
});
```

### 一般函数

> 注： 在修改函数实现时，如果原函数有返回值，那么我们在实现时，也要返回合适的值。（ **构造函数 没有返回值** ）

修改函数名为 func，参数为 byte[] 类型的函数的实现。**如果原函数有返回值，则修改后函数也需要有返回值。**

```
ClassName.func.overload('[B').implementation=function(param){
    // do something
    // return ...
}
```

**无参数的函数**

**如果原函数有返回值，则修改后函数也需要有返回值。**

```
ClassName.func.overload().implementation=function(){
    // do something
    // return
}
```

带返回值的函数修改

```

ClassName.func.overload().implementation=function(){
    // do something
    return this.func();
}
```

### Java.cast 进行 类型转换

用 Java.cast 方法来对一个对象进行类型转换，

示例：将 variable 转换成 java.lang.String

```
var StringClass = Java.use("java.lang.String");
var NewTypeClass = Java.cast(variable, StringClass);
```

### Java.available 字段

这个字段标记 Java虚拟机（例如： Dalvik 或者 ART）是否已加载, 操作Java任何东西的之前，要确认这个值是否为 true

### Java.perform 方法

当 Javascript 代码被成功附加到目标进程时，则调用 **Java.perform(fn)** 方法，我们核心的代码要在里面写。格式：

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

## 3、常用 Hook 方法 汇总

**Hook 一般函数 --- 使用 implementation**

```
var MainActivity = Java.use('ese.xposedtest.MainActivity');
//外部类 修改返回值
MainActivity.OutClass.implementation = function (arg) {
    var ret = this.OutClass(arg);
    console.log('Done:' + arg);
    return ret;
}
```

### Hook 重载函数 --- 使用 overload

```
// If two methods of a class have the same name
// you need to use 'overload'
// hook method 1
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
}

// hook method 2
//待补充
```

### Hook 构造函数 --- 使用 $init

```
// Intercept the initialization of java.lang.Stringbuilder's overloaded constructor.
// Write the partial argument to the console.
const StringBuilder = Java.use('java.lang.StringBuilder');
//We need to overwrite .$init() instead of .$new(), since .$new() = .alloc() + .init()
StringBuilder.$init.overload('java.lang.String').implementation = function (arg) {
    var partial = "";
    var result = this.$init(arg);
    if (arg !== null) {
         partial = arg.toString().replace('\n', '').slice(0,10);
    }
    // console.log('new StringBuilder(java.lang.String); => ' + result)
    console.log('new StringBuilder("' + partial + '");')
    return result;
}
console.log('[+] new StringBuilder(java.lang.String) hooked');
```

### Hook 新生成的 对象的实例 --- 使用 $new

```
const JavaString = Java.use('java.lang.String');
var exampleString1 = JavaString.$new('Hello World, this is an example string in Java.');
console.log('[+] exampleString1: ' + exampleString1);
```

### Hook 内部类—使用 $

```
var inInnerClass = Java.use('ese.xposedtest.MainActivity$inInnerClass');

inInnerClass.methodInclass.implementation = function()
{
    var arg0 = arguments[0];
    var arg1 = arguments[1];
    send("params1: "+ arg0 +" params2: " + arg1);
    return this.formInclass(1,"Frida");
}
```

### Hook native 函数

```
Interceptor.attach(Module.findExportByName("xxx.so" , "xxxx"), {
    onEnter: function(args) {
        send("open(" + Memory.readCString(args[0])+","+args[1]+")");
    },
    onLeave:function(retval){

    }
});
```

### Hook 静态变量（属性）/ 参考文章中有反射的方法

```
var ah = Java.use("com.ah");
console.log("To Log: " + ah.a.value);
ah.a.value = true;
```

### 打印堆栈

```
AndroidLog = Java.use("android.util.Log")
AndroidException = Java.use("java.lang.Exception")
function printStackTrace(){
 console.log(AndroidLog .getStackTraceString(AndroidException .$new()));
}
```

### byte[ ] 转 String

```
function byte2string(array){
    var result = "";
    for(var i = 0; i < array.length; ++i){
        result+= (String.fromCharCode(array[i]));
    }
   return result;
}
```

ArrayBuffer转String: 解决中文乱码(模板)

```
function ab2str(buf) {
    return new Uint16Array(buf)
    // encodedString = String.fromCodePoint.apply(null, new Uint16Array(buf));
    // // decodedString = encodeURI(encodedString);//没有这一步中文会乱码
    // // console.log(decodedString);
    // return encodedString
}
```

### String 转 Uint8Array

```
function string2byte(str){
  for (var i = 0,arr=[]; i < str.length;i++){
    arr.push(str.charCodeAt(i));
  }
  return new Uint8Array(arr);
}
```

字符串 转 Uint8Array (模板)

```
function stringToUint8Array(str){
  var arr = [];
  for (var i = 0, j = str.length; i < j; ++i) {
    arr.push(str.charCodeAt(i));
  }

  var tmpUint8Array = new Uint8Array(arr);
  return tmpUint8Array
}
```

string 转 ArrayBuffer (模板)

```
function str2ab(str) {
    var buf = new ArrayBuffer(str.length * 2); // 每个字符占用2个字节
    var bufView = new Uint16Array(buf);
    for (var i = 0, strLen = str.length; i < strLen; i++) {
        bufView[i] = str.charCodeAt(i);
    }
    return buf;
}
```

Uint8Array 转 字符串 (模板)

```
function Uint8ArrayToString(fileData){
    console.log(fileData)
  var dataString = "";
  for (var i = 0; i < fileData.length; i++) {
    dataString += String.fromCharCode(fileData[i]);
  }

  return dataString
}
```

### int 转 bytes

```
function intTobytes(n) {
  var bytes = [];
  for (var i = 0; i < 2; i++) {
    bytes[i] = n >> (8 - i * 8);

  }
  return bytes;
}
```

### string 转 ArrayBuffer

```
function str2arraybuffer(str) {
  var buf = new ArrayBuffer(str.length * 2); // 每个字符占用2个字节
  var bufView = new Uint16Array(buf);
  for (var i = 0, strLen = str.length; i < strLen; i++) {
    bufView[i] = str.charCodeAt(i);
  }
  return buf;
}
```

### ArrayBuffer 转 String

```
function ab2str(buf) {
  return String.fromCharCode.apply(null, new Uint8Array(buf));
}
```

### 添加时间戳

```
 console.log(new Date(new Date().getTime())
```

### byte 转 Hex

这个非常常用，因为有些byte是没有办法转成string的，只能转成hex，用来查看

```
function byteToHexString(uint8arr) {
  if (!uint8arr) {
    return '';
  }

  var hexStr = '';
  for (var i = 0; i < uint8arr.length; i++) {
    var hex = (uint8arr[i] & 0xff).toString(16);
    hex = (hex.length === 1) ? '0' + hex : hex;
    hexStr += hex;
  }

  return hexStr.toUpperCase();
}
```

### Hex 转 byte

```
function hexStringToByte(str) {
  if (!str) {
    return new Uint8Array();
  }

  var a = [];
  for (var i = 0, len = str.length; i < len; i+=2) {
    a.push(parseInt(str.substr(i,2),16));
  }
  return new Uint8Array(a);
}
```

### frida hook so 导出函数

```
# -*- coding: UTF-8 -*-
import frida, sys

jsCode = """
Java.perform(function(){
    var nativePointer = Module.findExportByName("libhello.so", "Java_com_xiaojianbang_app_NativeHelper_add"
    send("native: " + nativePointer);
    Interceptor.attach(nativePointer, {
        onEnter: function(args){
            send(args[0]);
            send(args[1]);
            send(args[2].toInt32());
            send(args[3].toInt32());
            send(args[4].toInt32());
        },
        onLeave: function(retval){
            send(retval.toInt32());
        }
    });
});
""";

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

process = frida.get_remote_device().attach("com.xiaojianbang.app")
script= process.create_script(jsCode)
script.on("message", message)
script.load()
sys.stdin.read()
```

基于frida框架Hook native中的函数(1)：基于frida框架Hook native中的函数(1)_Fly20141201. 的专栏-CSDN博客_frida hook so

安卓逆向入门之使用 frida 框架简单 Hook native 层的函数：安卓逆向入门之使用frida框架简单Hook native层的函数_测试0901-1-CSDN博客

抖音数据采集Frida教程，Frida Java Hook 详解：代码及示例（下）：https://segmentfault.com/a/1190000039041770

hook工具 frida 原理及使用：hook工具frida原理及使用 - 简书

**frida hook AES DES 3DES RSA MD5 SHA MAC firda 自吐算法**

哔哩哔哩：frida hook AES DES RSA 自吐算法_哔哩哔哩_bilibili

frida hook AES DES RSA 自吐算法：frida hook AES DES RSA 自吐算法 - 移动安全王铁头 - 博客园
frida hook java原生算法同时打印调用堆栈：frida hook java原生算法同时打印调用堆栈_weixin_34365417的博客-CSDN博客
FridaHookSysAPI：GitHub - zhudongjie/FridaHookSysAPI: System level encryption algorithm Hook from Frida
Frida 各种 Hook 示例 js ：Frida Hook - 简书

**百度关键字：frida hook 安卓 抓包　或者　frida hook okhttp 抓包**

精品连载丨安卓 App 逆向课程之四 frida 注入 Okhttp 抓包中篇：精品连载丨安卓 App 逆向课程之四 frida 注入 Okhttp 抓包中篇_静觅-CSDN博客

Android的网络请求库选择：Android的网络请求库选择 - 简书

android 网络请求库的比较：android 网络请求库的比较 - 水马 - 博客园

示例代码：

```
# -*- coding: UTF-8 -*-

import sys
import frida


'''
哔哩哔哩:https://www.bilibili.com/video/av927982888/
frida hook AES DES RSA 自吐算法:
    https://www.cnblogs.com/shlyd/p/14057423.html
frida hook java原生算法同时打印调用堆栈:
    https://blog.csdn.net/weixin_34365417/article/details/93088342
FridaHookSysAPI
    https://github.com/zhudongjie/FridaHookSysAPI
Frida 各种 Hook 示例 js
    https://www.jianshu.com/p/4291ee42c412
'''


js_code = """
function showStacks() {
    send(Java.use("android.util.Log").getStackTraceString(Java.use("java.lang.Exception").$new()));
}
// 这两种打印 堆栈信息的方法都可以
function showStacks_2() {
    Java.perform(function () {
        send(Java.use("android.util.Log").getStackTraceString(Java.use("java.lang.Exception").$new()));
    });
}
function bytesToHex(arr)
{
    var str = "";
    for(var i=0; i<arr.length; i++)
```

```
    for(var i=0; i<arr.length; i++)
    {
       var tmp = arr[i];
       if (tmp < 0) {
       tmp =(255+tmp+1).toString(16);
       } else {
            tmp = tmp.toString(16);
       }
       if(tmp.length == 1)
       {
           tmp = "0" + tmp;
       }
       str += tmp;
    }
    return str;
}
function bytesToBase64(arr)
{
    var str = "";
    for(var i=0; i<arr.length; i++)
    {
       var tmp = arr[i];
       if (tmp < 0) {
       tmp =(255+tmp+1).toString(16);
       } else {
            tmp = tmp.toString(16);
       }
       if(tmp.length == 1)
       {
           tmp = "0" + tmp;
       }
       str += tmp;
    }
    return str;
}
function byteArray2String(byteArray) {  // 把 byte 转换成 对应的ASCII字符
    var buffer = Java.array('byte', byteArray);
    // console.log(buffer.length)
    var result = "";
    for (var i = 0; i < buffer.length; i++) {
        result += (String.fromCharCode(buffer[i]));
    }
    return result;
}
function bytesToString(arr)  //把 byte 转换成 16进制的ASCII值
{
    var str = "";
    for(var i=0; i<arr.length; i++)
    {
       var tmp = arr[i];
       if (tmp < 0) {
       tmp =(255+tmp+1).toString(16);
       } else {
            tmp = tmp.toString(16);
       }
       if(tmp.length == 1)
       {
           tmp = "0" + tmp;
       }
       str += tmp;
```

```
    }
    return str;
}
Java.perform(function () {
    var secretKeySpec = Java.use('javax.crypto.spec.SecretKeySpec');
    secretKeySpec.$init.overload('[B','java.lang.String').implementation = function (a,b) {
        showStacks();
        var result = this.$init(a, b);
        send("====================================");
        send("算法名：" + b + "|Dec密钥:" + bytesToString(a));
        send("算法名：" + b + "|Hex密钥:" + bytesToHex(a));
        return result;
    }

    var mac = Java.use('javax.crypto.Mac');
    mac.getInstance.overload('java.lang.String').implementation = function (a) {
        showStacks();
        var result = this.getInstance(a);
        send("====================================");
        send("算法名：" + a);
        return result;
    }
    mac.update.overload('[B').implementation = function (a) {
        showStacks();
        this.update(a);
        send("====================================");
        send("update:" + bytesToString(a))
    }
    mac.update.overload('[B','int','int').implementation = function (a,b,c) {
        showStacks();
        this.update(a,b,c)
        send("====================================");
        send("update:" + bytesToString(a) + "|" + b + "|" + c);
    }
    mac.doFinal.overload().implementation = function () {
        showStacks();
        var result = this.doFinal();
        send("====================================");
        send("doFinal结果:" + bytesToHex(result));
        send("doFinal结果:" + bytesToBase64(result));
        return result;
    }
    mac.doFinal.overload('[B').implementation = function (a) {
        showStacks();
        var result = this.doFinal(a);
        send("====================================");
        send("doFinal参数:" + bytesToString(a));
        send("doFinal结果:" + bytesToHex(result));
        send("doFinal结果:" + bytesToBase64(result));
        return result;
    }

    var md = Java.use('java.security.MessageDigest');
    md.getInstance.overload('java.lang.String','java.lang.String').implementation = function (a,b) {
        showStacks();
        send("====================================");
        send("算法名：" + a);
        send("参数：" + b);
        return this.getInstance(a, b);
    }
```

```
}
md.getInstance.overload('java.lang.String').implementation = function (a) {
    showStacks();
    send("=====================================");
    send("算法名：" + a);
    return this.getInstance(a);
}
md.update.overload('[B').implementation = function (byte_array) {
    showStacks();
    send("=====================================");
    //send("update:" + bytesToString(byte_array));
    //send("ori:" + bytesToString(byte_array));
    send("ori:" + byteArray2String(byte_array));
    return this.update(byte_array);
}
md.update.overload('byte').implementation = function (byte_array) {
    showStacks();
    send("=====================================");
    //send("update:" + bytesToString(byte_array));
    send("update:" + byteArray2String(byte_array));
    return this.update(byte_array);
}
md.update.overload('java.nio.ByteBuffer').implementation = function (byte_array) {
    showStacks();
    send("=====================================");
    //send("update:" + bytesToString(byte_array));
    send("update:" + byteArray2String(byte_array));
    return this.update(byte_array);
}

md.update.overload('[B','int','int').implementation = function (a,b,c) {
    showStacks();
    send("=====================================");
    send("update:" + bytesToString(a) + "|" + b + "|" + c);
    return this.update(a,b,c);
}
md.digest.overload().implementation = function () {
    showStacks();
    send("=====================================");
    var result = this.digest();
    send("digest结果:" + bytesToHex(result));
    send("digest结果:" + bytesToBase64(result));
    return result;
}
md.digest.overload('[B').implementation = function (a) {
    showStacks();
    send("=====================================");
    send("digest参数:" + bytesToString(a));
    var result = this.digest(a);
    send("digest结果:" + bytesToHex(result));
    send("digest结果:" + bytesToBase64(result));
    return result;
}

var ivParameterSpec = Java.use('javax.crypto.spec.IvParameterSpec');
ivParameterSpec.$init.overload('[B').implementation = function (a) {
    showStacks();
    var result = this.$init(a);
    send("=====================================");
    send("iv向量:" + bytesToString(a));
```

```
    send("iv向量:" + bytesToHex(a));
    return result;
}

var cipher = Java.use('javax.crypto.Cipher');
cipher.getInstance.overload('java.lang.String').implementation = function (a) {
    showStacks();
    var result = this.getInstance(a);
    send("====================================");
    send("模式填充:" + a);
    return result;
}
cipher.update.overload('[B').implementation = function (a) {
    showStacks();
    var result = this.update(a);
    send("====================================");
    send("update:" + bytesToString(a));
    return result;
}
cipher.update.overload('[B','int','int').implementation = function (a,b,c) {
    showStacks();
    var result = this.update(a,b,c);
    send("====================================");
    send("update:" + bytesToString(a) + "|" + b + "|" + c);
    return result;
}
cipher.doFinal.overload().implementation = function () {
    showStacks();
    var result = this.doFinal();
    send("====================================");
    send("doFinal结果:" + bytesToHex(result));
    send("doFinal结果:" + bytesToBase64(result));
    return result;
}
cipher.doFinal.overload('[B').implementation = function (a) {
    showStacks();
    var result = this.doFinal(a);
    send("====================================");
    send("doFinal参数:" + bytesToString(a));
    send("doFinal结果:" + bytesToHex(result));
    send("doFinal结果:" + bytesToBase64(result));
    return result;
}

var x509EncodedKeySpec = Java.use('java.security.spec.X509EncodedKeySpec');
x509EncodedKeySpec.$init.overload('[B').implementation = function (a) {
    showStacks();
    var result = this.$init(a);
    send("====================================");
    send("RSA密钥:" + bytesToBase64(a));
    return result;
}

var rSAPublicKeySpec = Java.use('java.security.spec.RSAPublicKeySpec');
rSAPublicKeySpec.$init.overload('java.math.BigInteger','java.math.BigInteger').implementation = functio
    showStacks();
    var result = this.$init(a,b);
    send("====================================");
    //send("RSA密钥:" + bytesToBase64(a));
    send("RSA密钥N:" + a.toString(16));
```

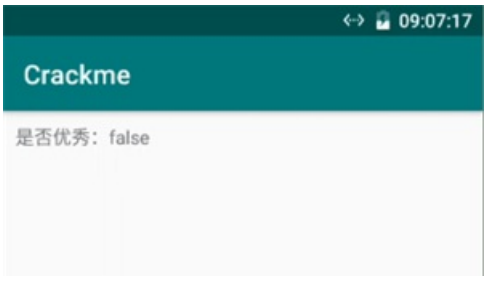
```
        send("RSA密钥N:" + a.toString(16));
        send("RSA密钥E:" + b.toString(16));
        return result;
    }
});
"""


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


if __name__ == '__main__':
    flag = 1
    if 1 == flag:
        process = frida.get_remote_device().attach("com.iCitySuzhou.suzhou001")
        script = process.create_script(js_code)
        script.on("message", js_callback_func)
        script.load()
        sys.stdin.read()
    elif 2 == flag:
        devices = frida.get_remote_device()
        pid = devices.spawn(['com.iCitySuzhou.suzhou001'])
        process = devices.attach(pid)

        script = process.create_script(js_code)
        script.on('message', js_callback_func)
        print('[*] Running CTF')
        script.load()
        devices.resume(pid)
        sys.stdin.read()
```

## 实例 1：修改 返回值 和 参数

有了以上的基础知识，我们就可以进行编写代码了

**修改返回值**

**场景：**假设有以下的程序，给isExcellent方法传入两个值，通过计算，返回一个布尔值，表示是否优秀。默认情况下，它是只会显示是否优秀：`false`的，因为我们默认传入的数很小:




**Java 代码：**

```
public class MainActivity extends AppCompatActivity {
    private  String TAG="Crackme";
    private TextView textView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textView =findViewById(R.id.tv);
        textView.setText("是否优秀："+isExcellent(46,54));
    }

    private  boolean isExcellent(int chinese, int math){
        if( chinese + math >=180){
            return true;
        }
        else{
            return false;
        }
    }

}
```

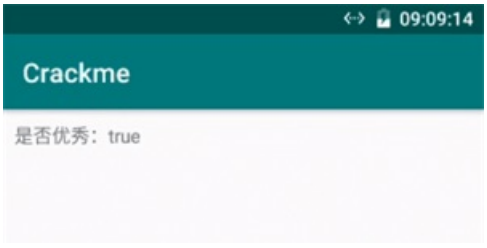
我们编写一个脚本来 Hook isExcellent函数，使它返回true，显示为是否优秀：true。对于这种简单的场景，直接修改返回值就可以了，因为只有结果是重要的。

### JavaScript 代码

想直接返回结果很简单，直接在匿名方法里return即可。

```
if(Java.available){
    Java.perform(function(){
        var MainActivity = Java.use("com.luoyesiqiu.crackme.MainActivity");
        MainActivity.isExcellent.implementation=function(){
            return true;
        }
    });

}
```

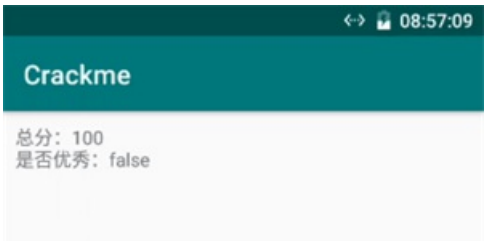
- 将上面的代码保存为：exp1.js
- 执行 adb shell 'su -c /data/local/tmp/frida-server' 启动服务端
- 运行目标 App
- 执行 frida -U -l exp1.js com.luoyesiqiu.crackme 注入代码
- 按返回键返回桌面，再重新打开App, 发现达到预期
- 在命令行输入exit，回车，停止注入代码



注：这里为什么要打开两次App？第一打开是为了让 frida 能够找到进程，第二次打开是为了验证结果，即使 Hook 成功了，界面是有缓存的，并不能实时显示 Hook 结果，所以需要重新打开 App

**修改参数**

**场景：**假设有以下场景，isExcellent 除了返回是否优秀以外，方法的内部还把分数打印出来。




**Java 代码：**

```
public class MainActivity extends AppCompatActivity {
    private  String TAG="Crackme";
    private TextView textView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textView =findViewById(R.id.tv);
        textView.append("是否优秀：+isExcellent(46,54)+"\n");
    }

    private  boolean isExcellent(int chinese, int math){
        textView.append("语文+数学总分："+(chinese+math)+"\n");
        if( chinese + math >=180){
            return true;
        }
        else{
            return false;
        }
    }
}
```

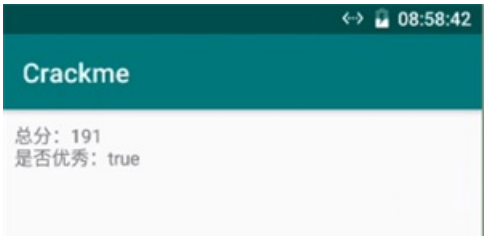
这种情况下我们不可能只返回是否优秀吧，显示的总分很低，但是却返回优秀，是很尴尬的...所以我们要修改 isExcellent 方法的参数，使其通过计算打印和返回合理的值。

**JavaScript 代码**

```
if(Java.available){
    Java.perform(function(){
        var MainActivity = Java.use("com.luoyesiqiu.crackme.MainActivity");
        MainActivity.isExcellent.overload("int","int").implementation=function(chinese,math){
            return this.isExcellent(95,96);
        }
    });

}
```

上面的代码，通过 overload 方法重载参数，修改 isExcellent 方法实现，并在实现函数里调用原来的方法，得到新的返回值

- 将上面的代码保存为：exp2.js
- 执行 adb shell 'su -c /data/local/tmp/frida-server'启动服务端（如果上面启动的服务端还开着可省略这一步）
- 运行目标 App
- 执行 frida -U -l exp2.js com.luoyesiqiu.crackme 注入代码
- 按返回键，再重新打开 App, 发现达到预期
- 在命令行输入exit，回车，停止注入代码




### 配合 Python 脚本注入

在本文刚开始的时候说到，我们可以编写 Python 代码来配合 Javascript 代码注入。

下面我们来看看，怎么使用，先看一段代码：

```
# -*- coding: UTF-8 -*-

import frida, sys

jscode = """
if(Java.available){
    Java.perform(function(){
        var MainActivity = Java.use("com.luoyesiqiu.crackme.MainActivity");
        MainActivity.isExcellent.overload("int","int").implementation=function(chinese,math){
            console.log("[javascript] isExcellent be called.");
            send("isExcellent be called.");
            return this.isExcellent(95,96);
        }
    });
}
"""

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

# 查找USB设备并附加到目标进程
session = frida.get_usb_device().attach('com.luoyesiqiu.crackme')
# 在目标进程里创建脚本
script = session.create_script(jscode)
# 注册消息回调
script.on('message', on_message)
print(' Start attach')
# 加载创建好的javascript脚本
script.load()
# 读取系统输入
sys.stdin.read()
```

- 将上面的代码，保存为 exp3.py
- 执行 adb shell 'su -c /data/local/tmp/frida-server'启动服务端（如果上面启动的服务端还开着可省略这一步）
- 运行目标 App
- 执行 python exp3.py 注入代码
- 按返回键，再重新打开 App, 发现达到预期
- 按 Ctrl+C 停止脚本和停止注入代码

上面是一段 Python 代码，我们来分析它的步骤：

1. 通过调用 **frida.get_usb_device()** 方法来得到一个连接中的 USB 设备（ Device类 ）实例
2. 调用 Device类的 **attach()** 方法来附加到目标进程并得到一个 **会话**（ **Session类**）实例，该方法有一个参数，就是需要注入的进程名
3. 接着调用 **Session类** 的 **create_script()** 方法创建一个脚本，传入需要注入的 **javascript** 代码并得到 **Script类** 的 实例
4. 调用 **Script类** 的 **on()方法** 添加一个消息回调，第一个参数是**信号名**，乖乖传入message 就行，第二个是**回调函数**
5. 最后调用 **Script类** 的 **load()方法** 来加载刚才创建的脚本。

> 注： 1. 如果想在 javascript 输出日志，可以调用 console.log()方法。
> 2. 如果想给客户端发送消息，可以在 javascript 代码里调用 send()方法，
> 并在客户端 Python 代码里注册一个消息回调来接收服务端发来的消息。

可以看到，结合python代码，使注入更加的灵活了。

如果想看 Python端 frida 模块的代码，可以访问：frida-python/core.py at master · frida/frida-python · GitHub

参考：Welcome | Frida • A world-class dynamic instrumentation framework

**示例代码：**

```
# -*- coding: UTF-8 -*-

import frida, sys

jscode = """
if(Java.available){
    Java.perform(function(){
        var MainActivity = Java.use("com.lanshifu.demo_module.ui.activity.DemoMainActivity");

        MainActivity.testCrash.overload("int").implementation=function(chinese,math){
            console.log("[javascript] testCrash method be called.");
            send("hook  testCrash method success>>>");
            return this.testCrash(12345678);
        }

        //声明一个Java类
        var MyClass = Java.use("com.lanshifu.demo_module.ui.activity.DemoHookTestActivity$MyClass");

        //hook 无参构造
        MyClass.$init.overload().implementation=function(){
                //调用原来构造
                this.$init();
                send("hook 无参构造 ");
        }

        //hook 有参构造，入参是 int[]
```

```
        MyClass.$init.overload('[I').implementation=function(param){
             send("hook  有参构造 init(int[] i) method ");
        }

        //hook 多个参数构造，入参是 int，int
        MyClass.$init.overload("int","int").implementation=function(param1,param2){
             send("hook  success  ");
             send(param1);
             send(param2);
             //修改返回值
             return 100;
        }

        //void函数
        MyClass.method1.overload().implementation=function(){
            send("hook  method1  ");
            this.method1();
        }

        //有入参的函数
        MyClass.method2.overload("int","int").implementation=function(param1,param2){
            send("hook  method2  ");

            //实例化一个类并调用它的method3方法
            var ClassName=Java.use("com.lanshifu.demo_module.ui.activity.DemoHookTestActivity$MyClass");
           var instance = ClassName.$new();
           instance.method3(1,2,3);

            this.method2(100,100)
        }

        //
        MyClass.method3.overload("int","int","int").implementation=function(param1,param2,param3){
            send("hook  method3  ");
            this.method3(100,100,100)
        }


    });
}
"""

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

# 查找USB设备并附加到目标进程
session = frida.get_usb_device().attach('com.lanshifu.demo_module')

# 在目标进程里创建脚本
script = session.create_script(jscode)

# 注册消息回调
script.on('message', on_message)
print('[*] Start attach')

# 加载创建好的javascript脚本
```

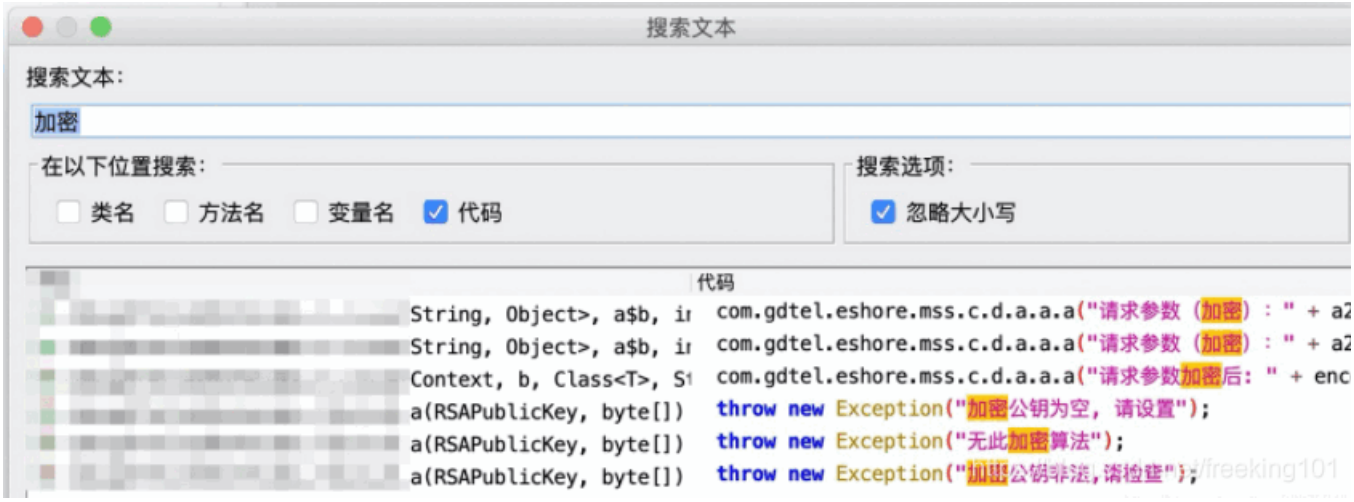
```
script.load()

# 读取系统输入
sys.stdin.read()
```

## 实例 2：frida hook 加解密函数

### 定位关键函数

取消勾选反混淆，否则影响关键函数定位


尝试搜索 AES、DES、RSA、加密、encode、decode 等关键字，也可以仔细跟进 http 等请求发起过程定位加解密函数




分析实现加密与解密的函数。加密函数：传入了公钥以及需要加密的字节数组

加密函数：传入了公钥以及需要加密的字节数组

```
340    public static byte[] a(RSAPublicKey rSAPublicKey, byte[] bArr) {
           byte[] bArr2;
341        if (rSAPublicKey == null) {
341            throw new Exception("加密公钥为空，请设置");
           }
           try {
346            Cipher instance = Cipher.getInstance(f3476b);
347            instance.init(1, rSAPublicKey);
348            int length = bArr.length;
365            ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
365            int i = 0;
365            int i2 = 0;
               while (true) {
365                int i3 = length - i;
365                if (i3 <= 0) {
365                    byte[] byteArray = byteArrayOutputStream.toByteArray();
366                    byteArrayOutputStream.close();
357                    return byteArray;
                   }
357                if (i3 > 117) {
357                    bArr2 = instance.doFinal(bArr, i, 117);
                   } else {
359                    bArr2 = instance.doFinal(bArr, i, i3);
                   }
378                byteArrayOutputStream.write(bArr2, 0, bArr2.length);
378                i2++;
378                i = i2 * 117;
               }
           } catch (NoSuchAlgorithmException unused) {
369            throw new Exception("无此加密算法");
           } catch (NoSuchPaddingException e) {
371            e.printStackTrace();
```

解密函数：传入了私钥以及需要解密的字节数组

```
451    public static byte[] a(RSAPrivateKey rSAPrivateKey, byte[] bArr) {
           byte[] bArr2;
452        if (rSAPrivateKey == null) {
452            throw new Exception("解密私钥为空，请设置");
           }
           try {
458            Cipher instance = Cipher.getInstance(f3476b);
459            instance.init(2, rSAPrivateKey);
460            ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(bArr);
461            ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
463            byte[] bArr3 = new byte[128];
               while (true) {
466                int read = byteArrayInputStream.read(bArr3);
```

https://blog.csdn.net/freeking101

编写 hook.js

```
function main() {
    if (Java.available) {
        console.log("********** hook start ***********");
        Java.perform(function() {
            var JniUtils = Java.use("com.gdtel.eshore.mss.lib.b.b");
            JniUtils.a.overload("java.security.interfaces.RSAPrivateKey","[B").implementation=function(arg1
                console.log("********** decodeByAES  start ***********");
                var a =this.a(arg1,arg2);
                var String = Java.use("java.lang.String")
                var data = String.$new(a)
                console.log("from data: " + data)
                return a;
            }
            JniUtils.a.overload("java.security.interfaces.RSAPublicKey","[B").implementation=function(arg3,
                console.log("********** encodeByRSA  start ***********");
                   var b =this.a(arg3,arg4);
                   var String1 = Java.use("java.lang.String")
                   var data1 = String1.$new(arg4)
                   console.log("from data: " + data1)
                   return b;
           });
            }
    function printTrace(){
        console.log("***************** printTrace start **********************");
        var jAndroidLog = Java.use("android.util.Log");
        var jException = Java.use("java.lang.Exception");

        var threadef = Java.use('java.lang.Thread');
        var threadinstance = threadef.$new();
        var stack = threadinstance.currentThread().getStackTrace();
        console.log("Full call stack:");
        for(var i = 0; i < stack.length; ++i){
            console.log(stack[i].toString());
        }
        console.log("***************** printTrace finish **********************");
    }
}

setImmediate(main)
```

运行：frida -U -l demo.js xxx.xxx.xxx.xxx

## 2. frida hook 混淆后的方法

关键字：**frida hook 混淆后的方法**

求助] Frida Hook混淆后的类、方法、变量异常

GitHub - xiaokanghub/Android: Android 加固应用Hook方式-Frida

[求助]这种反混淆的代码，frida如何hook-问答版块-看雪论坛-安全社区|安全招聘|bbs.pediy.com

## 3. FRIDA 实用手册

：FRIDA 实用手册-阿里云开发者社区

## Python 部分

### JS 中文支持

使用 `codecs.open(scriptpath, "r", "utf-8")` 打开文件读取 js 即可。

### 获取指定 UID 设备

```
device = frida.get_device_manager().get_device("094fdb0a0b0df7f8")
```

### 获取远程设备

```
mgr = frida.get_device_manager()
device = mgr.add_remote_device("30.137.25.128:13355")
```

### 启动调试进程

```
pid = device.spawn([packename])
process = device.attach(pid)
script = process.create_script(jscode)
script.on('message', on_message)
script.load()
device.resume(pid)
```

### python 与 js 交互的官方示例

```
import sys
import frida


session = frida.attach("hello")
script = session.create_script("""
Interceptor.attach(ptr("%s"), {
    onEnter: function(args) {
        send(args[0].toString());
        var op = recv('input', function(value) {
            args[0] = ptr(value.payload);
        });
        op.wait();
    }
});
""" % int(sys.argv[1], 16))


def on_message(message, data):
    print(message)
    val = int(message['payload'], 16)
    script.post({'type': 'input', 'payload': str(val * 2)})


script.on('message', on_message)
script.load()
sys.stdin.read()
```

### 从 bytecode 加载脚本

```
# -*- coding: utf-8 -*-

import frida

system_session = frida.attach(0)
bytecode = system_session.compile_script(name="bytecode-example", source="""\
'use strict';
rpc.exports = {
  listThreads: function () {
    return Process.enumerateThreadsSync();
  }
};
""")

session = frida.attach("Twitter")
script = session.create_script_from_bytes(bytecode)
script.load()
api = script.exports

# 这里的 list_threads 是 listThreads 驼峰命名法自动转换后的结果，
# 由 rpc exports 功能导出给 python 调用
print("api.list_threads() =>", api.list_threads())
```

## JS 部分

### hook Android 短信发送 SendDataMessage

```
function hook_sms() {
    var SmsManager = Java.use('android.telephony.SmsManager');
    SmsManager.sendDataMessage.implementation = function (
        destinationAddress, scAddress, destinationPort, data, sentIntent, deliveryIntent) {
        console.log("sendDataMessage destinationAddress: " + destinationAddress + " port: " + destinationPo
        showStacks();
        this.sendDataMessage(destinationAddress, scAddress, destinationPort, data, sentIntent, deliveryInte
    }
}
```

### 定时执行函数

1. `setTimeout` 延迟执行一次

```
setTimeout(funcA, 15000);
```

1. `setInterval` 间隔循环执行

```
var id_ = setInterval(funcB, 15000);
clearInterval(id_);    // 终止
```

### bin array 转字符串

```
function bin2String(array) {
    if (null == array) {
        return "null";
    }
    var result = "";
    try {
        var String_java = Java.use('java.lang.String');
        result = String_java.$new(array);
    } catch (e) {
        dmLogout("== use bin2String_2 ==");
        result = bin2String_2(array);
    }
    return result;
}

function bin2String_2(array) {
    var result = "";
    try {
        var tmp = 0;
        for (var i = 0; i < array.length; i++) {
            tmp = parseInt(array[i]);
            if (tmp == 0xc0
                || (tmp < 32 && tmp != 10)
                || tmp > 126) {
                return result;
            }  // 不是可见字符就返回了, 换行符除外
            result += String.fromCharCode(parseInt(array[i].toString(2), 2));
        }
    } catch (e) {
        console.log(e);
    }
    return result;
}
```

### 自己封装输出函数（ 加入 线程ID 和 时间 ）

```
function getFormatDate() {
    var date = new Date();
    var month = date.getMonth() + 1;
    var strDate = date.getDate();
    if (month >= 1 && month <= 9) {
        month = "0" + month;
    }
    if (strDate >= 0 && strDate <= 9) {
        strDate = "0" + strDate;
    }
    var currentDate = date.getFullYear() + "-" + month + "-" + strDate
        + " " + date.getHours() + ":" + date.getMinutes() + ":" + date.getSeconds();
    return currentDate;
}

function dmLogout(str) {
    var threadid = Process.getCurrentThreadId();
    console.log("[" + threadid + "][" + getFormatDate() + "]" + str);
}
```

### 打印 Android Java 层堆栈

```
var showStacks = function () {
    Java.perform(function () {
        dmLogout(Java.use("android.util.Log").getStackTraceString(Java.use("java.lang.Exception").$new()));
    });
}
```

### TracerPid fgets 反调试

```
var anti_fgets = function () {
    dmLogout("anti_fgets");
    var fgetsPtr = Module.findExportByName("libc.so", "fgets");
    var fgets = new NativeFunction(fgetsPtr, 'pointer', ['pointer', 'int', 'pointer']);
    Interceptor.replace(fgetsPtr, new NativeCallback(function (buffer, size, fp) {
        var retval = fgets(buffer, size, fp);
        var bufstr = Memory.readUtf8String(buffer);
        if (bufstr.indexOf("TracerPid:") > -1) {
            Memory.writeUtf8String(buffer, "TracerPid:\t0");
            // dmLogout("tracerpid replaced: " + Memory.readUtf8String(buffer));
        }
        return retval;
    }, 'pointer', ['pointer', 'int', 'pointer']));
};
```

### 反调试时读取 LR 寄存器溯源

```
var anti_antiDebug = function () {
    var funcPtr = null;

    funcPtr = Module.findExportByName("xxxx.so", "p57F7418DCD0C22CD8909F9B22F0991D3");

    dmLogout("anti_antiDebug " + funcPtr);
    Interceptor.replace(funcPtr, new NativeCallback(function (pathPtr, flags) {
        dmLogout("anti ddddddddddddddebug LR: " + this.context.lr);
        return 0;
    }, 'int', ['int', 'int']));
};
```

### hook JNI API NewStringUTF

```
function hook_native_newString() {
    var env = Java.vm.getEnv();
    var handlePointer = Memory.readPointer(env.handle);
    dmLogout("env handle: " + handlePointer);
    var NewStringUTFPtr = Memory.readPointer(handlePointer.add(0x29C));
    dmLogout("NewStringUTFPtr addr: " + NewStringUTFPtr);
    Interceptor.attach(NewStringUTFPtr, {
        onEnter: function (args) {
        ...
        }
    });
}
```

### hook JNI API GetStringUTFChars

```
function hook_native_GetStringUTFChars() {
    var env = Java.vm.getEnv();
    var handlePointer = Memory.readPointer(env.handle);
    dmLogout("env handle: " + handlePointer);
    var GetStringUTFCharsPtr = Memory.readPointer(handlePointer.add(0x2A4));
    dmLogout("GetStringUTFCharsPtr addr: " + GetStringUTFCharsPtr);
    Interceptor.attach(GetStringUTFCharsPtr, {
        onEnter: function (args) {
            var str = "";
            Java.perform(function () {
                str = Java.cast(args[1], Java.use('java.lang.String'));
            });
            dmLogout("GetStringUTFChars: " + str);
            if (str.indexOf("linkData:") > -1) {    // 设置过滤条件
                dmLogout("========== found linkData LR: " + this.context.lr + "  ==========");
            }
        }
    });
};
```

### hook Android URI 打印堆栈

```
var hook_uri = function () {
    // coord: (7520,0,19) | addr: Ljava/net/URI;->parseURI(Ljava/lang/String;Z)V | loc: ?
    var uri = Java.use('java.net.URI');
    uri.parseURI.implementation = function (a1, a2) {
        a1 = a1.replace("xxxx.com", "yyyy.com");

        dmLogout("uri: " + a1);
        showStacks();
        return this.parseURI(a1, a2);
    }
}
```

### hook KXmlSerializer 拼装内容

```
function hook_xml() {
    var xmlSerializer = Java.use('org.kxml2.io.KXmlSerializer');    // org.xmlpull.v1.XmlSerializer
    xmlSerializer.text.overload('java.lang.String').implementation = function (text) {
        dmLogout("xtext: " + text);
        if ("GPRS" == text) {
            dmLogout("======>>> found GPRS");
            showStacks();
        }
        return this.text(text);
    }
}
```

### hook Android Log 输出

```
function hook_log() {
    dmLogout(TAG, "do hook log");
    var Log = Java.use('android.util.Log');
    Log.v.overload('java.lang.String', 'java.lang.String').implementation = function (tag, content) {
        dmLogout(tag + " v", content);
    };
    Log.d.overload('java.lang.String', 'java.lang.String').implementation = function (tag, content) {
        dmLogout(tag + " d", content);
    };
    Log.w.overload('java.lang.String', 'java.lang.String').implementation = function (tag, content) {
        dmLogout(tag + " w", content);
    };
    Log.i.overload('java.lang.String', 'java.lang.String').implementation = function (tag, content) {
        dmLogout(tag + " i", content);
    };
    Log.e.overload('java.lang.String', 'java.lang.String').implementation = function (tag, content) {
        dmLogout(tag + " e", content);
    };
}
```

### native 主动调用

```
var friendlyFunctionName = new NativeFunction(friendlyFunctionPtr, 'void', ['pointer', 'pointer']);
var returnValue = Memory.alloc(sizeOfLargeObject);
friendlyFunctionName(returnValue, param1);
```