一.常见使用方式

1.1方式一

py脚本和js脚本在同一个文件中。

inject_demo1.py

```
import frida, sys

jscode=
```
Java.perform(function(){

    要注入的代码

});
```
def on_message(messgae,data):
    print(message)
device=frida.get_remote_device()   //process=frida.get_usb_device()   获取设备
process=device.attach("xxxx.包名")   //附加到进程
script=process.create_script(jscode)  //这里加载要注入的js代码
script.on('message',on_message)    //加载回调函数，也就是js执行send函数规定要执行的方法。
script.load()
script.stdin.read()
```

1.2 方式二

使用frida直接将js注入进程

inject_demo2.js

```
setImmediate(main_inject;
function main_inject(){
    Java.perform(function(){
        待注入的代码
    });
}
```

注入命令

frida -U -f com.zzw.test   -l inject_demo2.js

-u   待调试应用包名

-l   注入脚本名

1.3 方式三

通过open打开js脚本的方式注入，易于维护解耦。

inject_demo3.js

```
setImmediate(main_inject;
function main_inject(){
    Java.perform(function(){
        待注入的代码
    });
}
```

inject.py

```
import frida, sys

def on_message(messgae,data):
    print(message)
device=frida.get_remote_device()   //process=frida.get_usb_device()   获取设备
process=device.attach("xxxx.包名")   //附加到进程
with open('inject_demo3.js') as f
    jscode=f.read()
script=process.create_script(jscode)  //这里加载要注入的js代码
script.on('message',on_message)    //加载回调函数，也就是js执行send函数规定要执行的方
法。
script.load()
sys.stdin.read()
```

二.Java层注入常见用法

2.1常见知识点

2.1.1 api介绍

- Java.perform(function)。frida的main，所有脚本都必须放在这个里面。
- Java.use(classname)。用于动态获取一个类的对象，为以后调用对象方法的实现。

2.1.2 hook普通函数

```
Java.perform(function(){
   var classname=Java.use('待hook类名')
   classname.方法名.implementation = function(){  //无参情况下
      console.log(arguments[0])
   }
   classname.方法名.implementation = function(p1,p2){  //无重载，带参情况下
      console.log("第一个参数",p1);
      console.log("第一个参数",arguments[0])
      console.log("第二个参数",p2);
      console.log("第二个参数",arguments[1])
   }
   classname.重载方法名.overload('java.lang.String','[B')implementation =
function(p1,p2){  //重载，带参情况下
      console.log("第一个参数",p1);
      console.log("第一个参数",arguments[0])
```

```
        console.log("第二个参数",p2);
        console.log("第二个参数",arguments[1])
    }
});
```

2.1.3 修改返回值

就以普通hook函数为例，假设函数方法名为invoke_method

```
Java.perform(function(){
    var classname=Java.use('待hook类名')
    classname.invoke_method.implementation = function(){  //无参情况下
        console.log(arguments[0])
    }
    classname.invoke_method.implementation = function(p1,p2){  //无重载，带参情况下
        console.log("第一个参数",p1);
        return this.invoke_method(p1,p2); //方式一 通过this.函数名（参数1，参数2）；
        return invoke_method.call(this,p1,p2); //方式二
        return invoke_method.apply(p1,p2);    //方式三
    }
});
```

call() 方法分别接受参数。

apply() 方法接受数组形式的参数。

2.1.4 hook构造函数

```
Java.perform(function(){
    var money=Java.use('com.zzw,test.Money')
    money.$init.implementation = function(p1,p2){  //无参情况下
        return this.$init(p1,p2);
    }
});
```

2.1.5 修改类中字段值

test.class

```
public class test{
  String aaa;
  String bbb;
  void abc(){};
  void bcd(){};
}
```

hook demo

```
Java.perform(function(){
     //静态字段的修改
     var money = Java.use("money类路径");
     //console.log(JSON.stringify(money.字段名));
     money.字段名.value = "xxxxx";
     console.log(money.flag.value);
```

```
    //非静态字段的修改
    Java.choose("money类路径", {
        onMatch: function(obj){
            obj._name.value = "ouyuan"; //字段名与函数名相同，前面加个下划线
            obj.name.value = "ouyuan"; //字段名与函数名不同
        },
        onComplete: function(){
        }
    });
});
```

或者通过java反射的方式

```
function setFieldValue(obj, fieldName, fieldValue) {
    var cls = obj.getClass();
    var field = cls.getDeclaredField(fieldName);
    field.setAccessible(true);
    field.set(obj, fieldValue);
}
```

2.2.6 hook类的所有方法

```
hook_code = """
Java.perform(function hookTest8(){
    Java.perform(function(){
        var md5 = Java.use("md5类路径");
        var methods = md5.class.getDeclaredMethods();
        for(var j = 0; j < methods.length; j++){
            var methodName = methods[j].getName();
            console.log(methodName);

            for(var k = 0; k < md5[methodName].overloads.length; k++){

                md5[methodName].overloads[k].implementation = function(){
                    for(var i = 0; i < arguments.length; i++){
                        console.log(arguments[i]);
                    }
                    return this[methodName].apply(this, arguments);
                }
            }
        }
    });
});
"""
```

####

2.2 常见需求

2.2.1 获取前台运行app

```
import frida
rdev = frida.get_remote_device()
front_app = rdev.get_frontmost_application()
print front_app
```

2.2.2 遍历所有的进程

```
import frida,sys
device = frida.get_usb_device()
all_process = device.enumerate_processes()
for process in all_process:
    print(process)
```

2.2.3 获取手机上的所有应用

```
import frida,sys
device = frida.get_usb_device()
all_app = device.enumerate_applications()
for application in all_app:
    print(application)
```

2.2.4 打印类中所有的方法

```
Java.perform(function()
{
   var classname=Java.use('包名');
   var all_method=classname.class.getDeclaredMethods();
   for(var i=0;i<all_method.length;i++)
   {
         console.log(all_method[i]);
         console.log(all_method[i].getName());
   }
});
```

这个方法在不知道类中有哪些函数，或者不知道参数类型的情况下很好用。

2.2.5 打印某个类的所有成员变量

```
function dumpAllFieldValue(obj) {
    if (obj === null) {
        return;
    }
    console.log("Dump all fields value for  " + obj.getClass() + " :");
    var cls = obj.getClass();
    while (cls !== null && !cls.equals(Java.use("java.lang.Object").class)) {
        var fields = cls.getDeclaredFields();
        if (fields === null || fields.length === 0) {
            cls = cls.getSuperclass();
            continue;
        }
        if (!cls.equals(obj.getClass())) {
            console.log("Dump super class  " + cls.getName() + " fields:");
        }
        for (var i = 0; i < fields.length; i++) {
            var field = fields[i];
            field.setAccessible(true);
            var name = field.getName();
            var value = field.get(obj);
            var type = field.getType();
            console.log(type + " " + name + "=" + value);
        }
        cls = cls.getSuperclass();
    }
}
```

2.2.6 获取成员变量的值

```
function getFieldValue(obj, fieldName) {
    var cls = obj.getClass();
    var field = cls.getDeclaredField(fieldName);
    field.setAccessible(true);
    var name = field.getName();
    var value = field.get(obj);
    // console.log("field: " + field + "\tname:" + name + "\tvalue:" + value);
    return value;
}
```

2.2.7 打印调用堆栈

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

2.2.8 对象实例化

```
Java.perform(function hookTest2(){
    var utils = Java.use("utils类路径");
    var money = Java.use("money类路径");
    utils.方法名.overload('重载参数').implementation = function(a){
        a = 888;
        var retval = this.方法名(money.$new("日元", 100000));//对象实例化
        console.log(a, retval);
        return retval;
    }
});
```

2.2.9 Hook动态加载的dex

```
hook_code = """
Java.perform(function () {
    Java.enumerateClassLoaders({
        onMatch: function (loader) {
            try {
                if (loader.loadClass("com.xiaojianbang.app.Dynamic")) {
                    Java.classFactory.loader = loader;
                    var Dynamic = Java.use("com.xiaojianbang.app.Dynamic");
                    console.log(Dynamic);
                    Dynamic.sayHello.implementation = function () {
                        return "9999999";
                    }
                }
            } catch (error) {
            }
        },
        onComplete: function () {
        }
    });
});
"""
```

2.2.10 Java 特殊类型的便遍历与修改（MAP举例）

```
hook_code = """
Java.perform(function () {
    var ShufferMap = Java.use("com.xiaojianbang.app.ShufferMap");
    console.log(ShufferMap);
    ShufferMap.show.implementation = function (map) {
        console.log(JSON.stringify(map));
        //Java map的遍历
        var key = map.keySet();
        var it = key.iterator();
        var result = "";
        while(it.hasNext()){
            var keystr = it.next();
            var valuestr = map.get(keystr);
            result += valuestr;
        }
        console.log(result);
        // return result;

        //Java map的修改
        map.put("pass", "zygx8");
        map.put("guanwang", "www.zygx8.com");

        var retval = this.show(map);
        console.log(retval);
        return retval;
    }
});
"""
```

2.2.11 打印HashMap

```
console.log(JSON.stringify(arguments))
var Map = Java.use('java.util.HashMap');
var args_map = Java.cast(arguments[1], Map);
console.log(args_map.toString());
```

2.2.12 java层主动调用

```
hook_code = """
Java.perform(function () {
    //静态方法的主动调用
    var rsa = Java.use("com.xiaojianbang.app.RSA");
    var str = Java.use("java.lang.String");
    var base64 = Java.use("android.util.Base64");
    var bytes = str.$new("xiaojianbang").getBytes();
    console.log(JSON.stringify(bytes));
    var retval = rsa.encrypt(bytes);
    var result = base64.encodeToString(retval, 0);
    console.log(result);
    //非静态方法的主动调用1（新建一个对象去调用）
    var res = Java.use("com.xiaojianbang.app.Money").$new("日元",
300000).getInfo();
    console.log(res);
    var utils = Java.use("com.xiaojianbang.app.Utils");
    res = utils.$new().myPrint(["xiaojianbang", "is very good", " ", "zygx8",
"is very good"]);
    console.log(res);
    //非静态方法的主动调用2（获取已有的对象调用）
    Java.choose("com.xiaojianbang.app.Money", {
        onMatch: function (obj) {
            if (obj._name.value == "美元") {
                res = obj.getInfo();
                console.log(res);
            }
        },
        onComplete: function () {
        }
    });
});
"""
```

2.2.13 获取参数类型

```
xxx.class.getType()
```

2.2.14 使用frida注入dex文件

```
hook_code = """
Java.perform(function () {
    Java.openClassFile("/data/local/tmp/xiaojianbang.dex").load();
    var xiaojianbang = Java.use("com.xiaojianbang.test.xiaojianbang");

    var ShufferMap = Java.use("com.xiaojianbang.app.ShufferMap");
    ShufferMap.show.implementation = function (map) {
        var retval = xiaojianbang.sayHello(map);
        console.log(retval);
        return retval;
    }

});
"""
```

2.2.15 多个函数同时hook

```
    var
InnerClasses=Java.use('com.example.androiddemo.Activity.FridaActivity4$InnerClas
ses');
    var all_method=InnerClasses.class.getDeclaredMethods();
    for(var i=0;i<all_method.length;i++)
    {
       var method=all_method[i].getName();
       InnerClasses[method].implementation = function()
       {return true;}
    }
```

2.2.16 枚举当前所有的线程

Process.enumerateThreads()：**枚举当前所有的线程，返回包含以下属性的对象数组：**

属性	含义
id	线程id
state	当前运行状态有running, stopped, waiting, uninterruptible or halted
context	带有键pc和sp的对象，它们是分别为ia32/x64/arm指定EIP/RIP/PC和ESP/RSP/SP的NativePointer对象。也可以使用其他处理器特定的密钥，例如eax、rax、r0、x0等。

```
    function frida_Process() {
        Java.perform(function () {
           var enumerateThreads =  Process.enumerateThreads();
           for(var i = 0; i < enumerateThreads.length; i++) {
            console.log("");
            console.log("id:",enumerateThreads[i].id);
            console.log("state:",enumerateThreads[i].state);
            console.log("context:",JSON.stringify(enumerateThreads[i].context));
            }
        });
    }
    setImmediate(frida_Process,0);
```

2.2.16 打印类所属对象

```
     var
FridaActivity5=Java.use('com.example.androiddemo.Activity.FridaActivity5')
     Java.choose('com.example.androiddemo.Activity.FridaActivity5',{
        onMatch :function(instance){
            console.log(instance.getDynamicDexCheck().$className);
        },
        onComplete : function(){
        }
     })
```

使用object.$className

三.Native方法hook

3.1框架原型

```
var str_name_so = "libxxxxx.so";    //要hook的so名
var str_name_func = "func_exp";          //要hook的函数名


var hook_addr_func = Module.findExportByName(libxxxxx.so , str_name_func);
console.log("func addr is ---" + n_addr_func);

Interceptor.attach(hook_addr_func, {
    //在hook函数之前执行的语句
    onEnter : function(args)
    {
        console.log("hook on enter")
    },
    //在hook函数之后执行的语句
    onLeave : function(retval)
    {
        console.log("hook on leave")
    }
});
```

3.2 常用需求

3.2.1 导出方法函数获取

```
var str_name_so = "libxxxxx.so";    //要hook的so名
var str_name_func = "func_exp";          //要hook的函数名

var exports = Module.enumberateExports(libxxxxx.so);
for(var i=0;i<exports.length;i++){
    if(exports[i].name=="想要的方法"){
        methodsaddress = exports[i].address;
    }
}

Interceptor.attach(methodsaddress, {
    //在hook函数之前执行的语句
    onEnter : function(args)
    {
        console.log("hook on enter")
    },
    //在hook函数之后执行的语句
    onLeave : function(retval)
    {
        console.log("hook on leave")
    }
});
```

3.2.2 未导出方法获取

```
var str_name_so = "libxxxxx.so";    //要hook的so名
var str_name_func = "func_exp";          //要hook的函数名

var base_addr = Module.findBaseAddress("libxxxxx.so");    //得到so的基址
var target_addr=base_addr.add(offset);   //添加偏移
var methodsaddress=ptr(target_addr);

Interceptor.attach(methodsaddress, {
    //在hook函数之前执行的语句
    onEnter : function(args)
    {
        console.log("hook on enter")
    },
    //在hook函数之后执行的语句
    onLeave : function(retval)
    {
        console.log("hook on leave")
    }
});
```

3.2.3 枚举导入表

```
var imports = Module.enumerateImports("libxiaojianbang.so");
for(var i = 0; i < imports.length; i++){
    if(imports[i].name == "strncat"){
        console.log(JSON.stringify(imports[i]));
        console.log(imports[i].address);
    }
}
```

3.2.4 枚举导出表

```
var exports = Module.enumerateExports("libxiaojianbang.so");
for(var i = 0; i < exports.length; i++){
    //if(exports[i].name == "strncat"){
        console.log(JSON.stringify(exports[i]));
    //}
}
```

3.2.5 函数地址计算

```
function hookTest14(){
    var soAddr = Module.findBaseAddress("libxiaojianbang.so");
    console.log(soAddr);
    var funcAddr = soAddr.add(0x23F4);
    console.log(funcAddr);
}
```

3.2.6 获取指针参数返回值

```
function hookTest5(){
    var soAddr = Module.findBaseAddress("libxiaojianbang.so");
    console.log(soAddr);
    var sub_930 = soAddr.add(0x930); //函数地址计算 thumb+1 ARM不加
    console.log(sub_930);

     var sub_208C = soAddr.add(0x208C); //函数地址计算 thumb+1 ARM不加
     console.log(sub_208C);
     if(sub_208C != null){
        Interceptor.attach(sub_208C,{
            onEnter: function(args){
                this.args1 = args[1];
            },
            onLeave: function(retval){
                console.log(hexdump(this.args1));
            }
        });
     }
}
```

3.2.7 内存读写

```
function hookTest7(){
    var soAddr = Module.findBaseAddress("libxiaojianbang.so");
    console.log(soAddr);
    if(soAddr != null){
        //console.log(soAddr.add(0x2C00).readCString());
        //console.log(hexdump(soAddr.add(0x2C00)));  //读取指定地址的字符串

        //var strByte = soAddr.add(0x2C00).readByteArray(16); //读内存
        //console.log(strByte);

        //soAddr.add(0x2C00).writeByteArray(stringToBytes("xiaojianbang")); //写
内存

        //console.log(hexdump(soAddr.add(0x2C00)));  //dump指定内存

        //var bytes = Module.readByteArray(soAddr.add(0x2C00), 16);
        //console.log(bytes);

    }
}
```

3.2.8 主动调用JNI函数

```
function hookTest8(){
    var funcAddr = Module.findExportByName("libxiaojianbang.so",
"Java_com_xiaojianbang_app_NativeHelper_helloFromC");
    console.log(funcAddr);
    if(funcAddr != null){
        Interceptor.attach(funcAddr,{
            onEnter: function(args){

            },
            onLeave: function(retval){
                var env = Java.vm.tryGetEnv();
                var jstr = env.newStringUtf("www.zygx8.com");  //主动调用jni函数
cstr转jstr
                retval.replace(jstr);
                var cstr = env.getStringUtfChars(jstr); //主动调用 jstr转cstr
                console.log(cstr.readCString());
                console.log(hexdump(cstr));
            }
        });
    }
}
```

3.2.9 jni函数hook (计算地址方式)

```
function hookTest9(){
    Java.perform(function(){
        //console.log(JSON.stringify(Java.vm.tryGetEnv()));
        var envAddr = ptr(Java.vm.tryGetEnv().handle).readPointer();
        var newStringUtfAddr = envAddr.add(0x538).readPointer();
        var registerNativesAddr = envAddr.add(1720).readPointer();
        console.log("newStringUtfAddr", newStringUtfAddr);
        console.log("registerNativesAddr", registerNativesAddr)
        if(newStringUtfAddr != null){
            Interceptor.attach(newStringUtfAddr,{
                onEnter: function(args){
                    console.log(args[1].readCString());
                    //args[1] = "xiaojianbang is very good!";
                },
                onLeave: function(retval){

                }
            });
        }
        if(registerNativesAddr != null){     //Hook registerNatives获取动态注册的函
数地址
            Interceptor.attach(registerNativesAddr,{
                onEnter: function(args){
                    console.log(args[2].readPointer().readCString());

 console.log(args[2].add(Process.pointerSize).readPointer().readCString());
                    console.log(args[2].add(Process.pointerSize *
2).readPointer());
                    console.log(hexdump(args[2]));
                    console.log("sub_289C",
Module.findBaseAddress("libxiaojianbang.so").add(0x289C));
                },
                onLeave: function(retval){

                }
            });
        }

    });
}
```

3.2.10 jni函数hook(libart.so)

```
  var artSym = Module.enumerateSymbols("libart.so");
    var NewStringUTFAddr = null;
    for(var i = 0; i < artSym.length; i++){
        if(artSym[i].name.indexOf("CheckJNI") == -1 &&
artSym[i].name.indexOf("NewStringUTF") != -1){
            console.log(JSON.stringify(artSym[i]));
            NewStringUTFAddr = artSym[i].address;
        }
    };

    if(NewStringUTFAddr != null){
        Interceptor.attach(NewStringUTFAddr,{
            onEnter: function(args){
                console.log(args[1].readCString());
            },
            onLeave: function(retval){

            }
        });
    }
```

3.2.11 So层函数主动调用

```
function hookTest11(){
    Java.perform(function(){
        var funcAddr = Module.findBaseAddress("libxiaojianbang.so").add(0x23F4);
        var func = new NativeFunction(funcAddr, "pointer", ['pointer',
'pointer']);
        var env = Java.vm.tryGetEnv();
        console.log("env: ", JSON.stringify(env));
        if(env != null){
            var jstr = env.newStringUtf("xiaojianbang is very good!!!");
            //console.log("jstr: ", hexdump(jstr));
            var cstr = func(env, jstr);
            console.log(cstr.readCString());
            console.log(hexdump(cstr));
        }
    });
}
```

3.2.12 frida读写文件

```
//frida API 读写文件
function hookTest12(){
    var ios = new File("/sdcard/xiaojianbang.txt", "w");
    ios.write("xiaojianbang is very good!!!\n");
    ios.flush();
    ios.close();
}
//Hook libc 读写文件
function hookTest13() {

    var addr_fopen = Module.findExportByName("libc.so", "fopen");
    var addr_fputs = Module.findExportByName("libc.so", "fputs");
    var addr_fclose = Module.findExportByName("libc.so", "fclose");

    console.log("addr_fopen:", addr_fopen, "addr_fputs:", addr_fputs,
"addr_fclose:", addr_fclose);
    var fopen = new NativeFunction(addr_fopen, "pointer", ["pointer",
"pointer"]);
    var fputs = new NativeFunction(addr_fputs, "int", ["pointer", "pointer"]);
    var fclose = new NativeFunction(addr_fclose, "int", ["pointer"]);

    var filename = Memory.allocUtf8String("/sdcard/xiaojianbang.txt");
    var open_mode = Memory.allocUtf8String("w");
    var file = fopen(filename, open_mode);
    console.log("fopen:", file);

    var buffer = Memory.allocUtf8String("zygxb\n");
    var retval = fputs(buffer, file);
    console.log("fputs:", retval);
    fclose(file);
}
```

3.2.13 枚举所有已加载的模块

```
function frida_Process() {
    Java.perform(function () {
        var process_Obj_Module_Arr = Process.enumerateModules();
        for(var i = 0; i < process_Obj_Module_Arr.length; i++) {
            console.log("",process_Obj_Module_Arr[i].name);
        }
    });
}
setImmediate(frida_Process,0);
```

3.

四.参考链接

https://blog.csdn.net/freeking101/article/details/109213383