frida hook 安装篇

frida安装

frida-12.7.24、frida-tools-5.3.0

```
pip install frida -i https://pypi.douban.com/simple --trusted-host
pypi.douban.com
pip install frida-tools -i https://pypi.douban.com/simple --trusted-host
pypi.douban.com
```

frida-server

https://github.com/frida/frida/releases

虚拟机

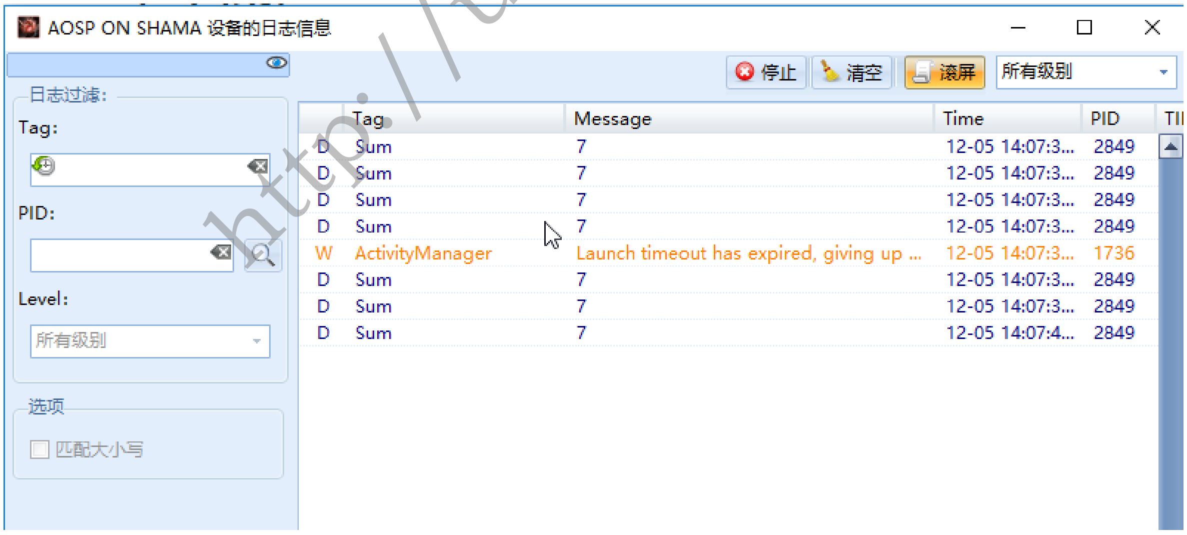
- frida-server-12.7.24-android-x86
- frida-server-12.7.24-android-x86_64

真机

- frida-server-12.7.24-android-arm
- frida-server-12.7.24-android-arm64

AndoridKiller

用于查看Android日志、进程、文件等等。



```
   Tag               Message                              Time              PID   TI
D  Sum               7                                    12-05 14:07:3...  2849
D  Sum               7                                    12-05 14:07:3...  2849
D  Sum               7                                    12-05 14:07:3...  2849
D  Sum               7                                    12-05 14:07:3...  2849
W  ActivityManager   Launch timeout has expired, giving up ...  12-05 14:07:3...  1736
D  Sum               7                                    12-05 14:07:3...  2849
D  Sum               7                                    12-05 14:07:3...  2849
D  Sum               7                                    12-05 14:07:4...  2849
```

frida启动

启动frida-server

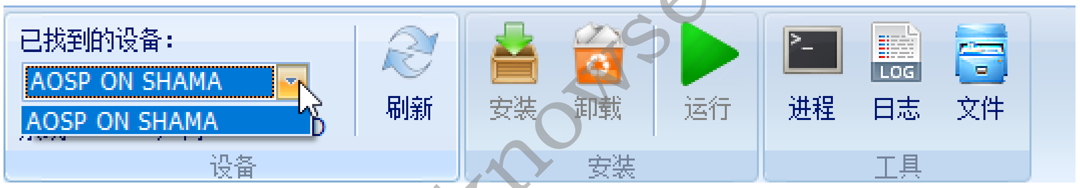
```
adb push frida-server-12.7.24-android-x86 /data/local/tmp/
adb shell
cd /data/local/tmp
chmod 777 frida-server-12.7.24-android-x86
./frida-server-12.7.24-android-x86
```

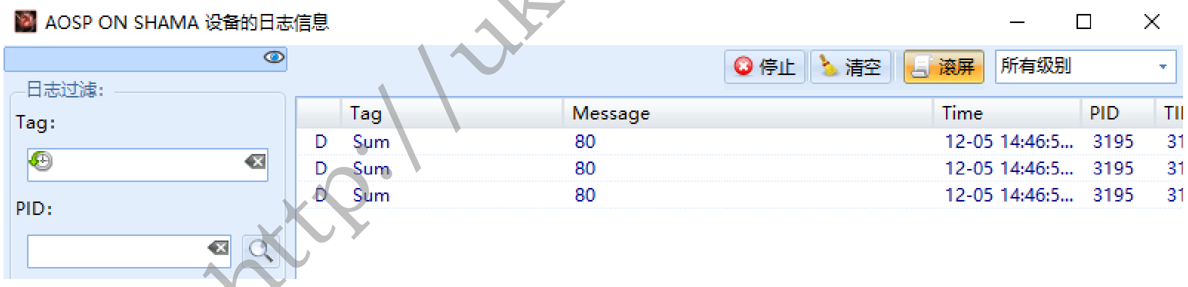
启动转发

```
adb forward tcp:27042 tcp:27042
adb forward tcp:27043 tcp:27043
```

Androidkiller adb连接

```
> adb.exe devices
List of devices attached
127.0.0.1:62026 device

> adb.exe connect 127.0.0.1:62026
adb server version (32) doesn't match this client (36); killing...
* daemon started successfully *
connected to 127.0.0.1:62026
```





```
   Tag      Message     Time               PID    TI
D  Sum      80          12-05 14:46:5...   3195   31
D  Sum      80          12-05 14:46:5...   3195   31
D  Sum      80          12-05 14:46:5...   3195   31
```

frida脚本

```
import time
import sys
import frida

js_code ='''
    console.log("Script loaded successfully ");
    Java.perform(function x() { //Silently fails without the sleep from the
python code
    console.log("Inside java perform function");
    //get a wrapper for our class
    var my_class = Java.use("com.example.a11x256.frida_test.my_activity");
    //replace the original implmenetation of the function `fun` with our custom
function
    my_class.fun.implementation = function (x, y) {
        //print the original arguments
        console.log("original call: fun(" + x + ", " + y + ")");
        //call the original implementation of `fun` with args (2,5)
        var ret_value = this.fun(2, 5);
        return ret_value;
    }
});
'''

device = frida.get_remote_device()
pid = device.spawn(["com.example.a11x256.frida_test"])
device.resume(pid)
time.sleep(1)  # Without it Java.perform silently fails
session = device.attach(pid)
script = session.create_script(js_code)
#with open("s1.js") as f:
#    script = session.create_script(f.read())
script.load()

# prevent the python script from terminating
sys.stdin.read()
```

- `session = frida.get_remote_device().attach("com.example.a11x256.frida_test")`，要hook的进程包名`com.yusakul.myapplication`，`get_remote_device()`获取远程设备，如果是真机usb接入的话，可修改为函数`get_usb_device()`。
- `script = session.create_script(js_code) script.on('message',on_message) script.load()`

  创建加载js脚本。同时也可以通过外部js文件的方式加载

  ```
  with open("s1.js") as f:
      script = session.create_script(f.read())
  ```

- `js_code`是hook脚本，为js脚本语言。

apk hook源码

```
package com.example.a11x256.frida_test;

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

public class my_activity extends AppCompatActivity {
    /* Access modifiers changed, original: protected */
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView((int) R.layout.activity_my_activity);
        while (true) {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            fun(50, 30);
```

```
        }
    }

    /* Access modifiers changed, original: 0000 */
    public void fun(int x, int y) {
        Log.d("Sum", String.valueOf(x + y));
    }
}
```

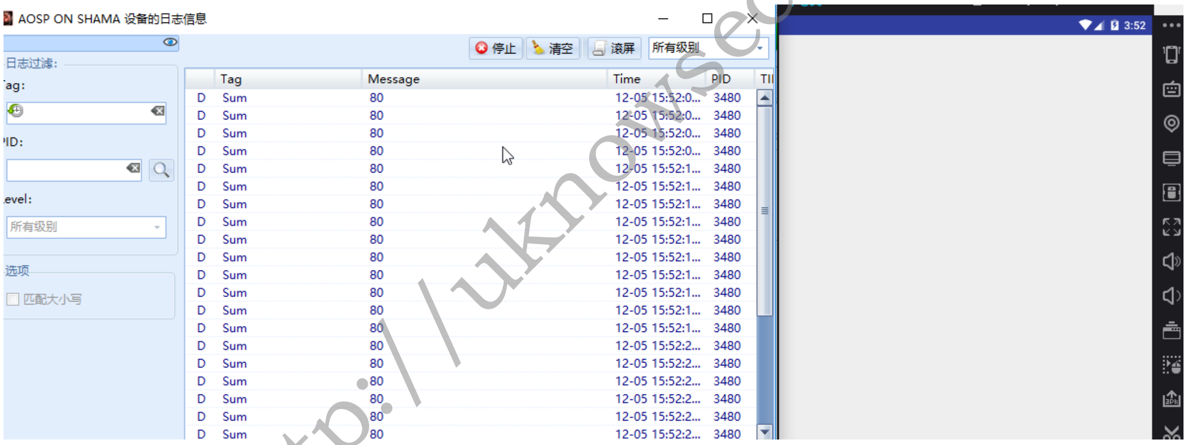
此App时间简单的运算，计算50+30并在日志中进行输出。而hook脚本的作用是用于打印原始参数 `console.log( "original call: fun("+ x + ", " + y + ")");`。

并调用原函数，并传参2,5。将函数执行结果返回。

```
var ret_value = this.fun(2, 5);
return ret_value;
```

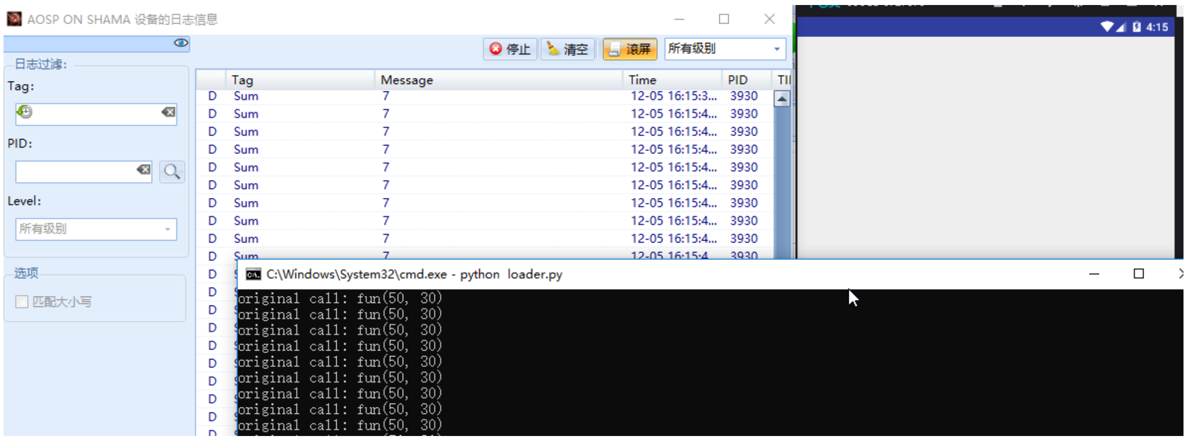
运行frida hook脚本

APP正常运行是输出50+30的结果。



```
   Tag   Message    Time               PID    TI
D  Sum   80         12-05 15:52:0...   3480
D  Sum   80         12-05 15:52:0...   3480
D  Sum   80         12-05 15:52:0...   3480
D  Sum   80         12-05 15:52:0...   3480
D  Sum   80         12-05 15:52:1...   3480
D  Sum   80         12-05 15:52:1...   3480
D  Sum   80         12-05 15:52:1...   3480
D  Sum   80         12-05 15:52:1...   3480
D  Sum   80         12-05 15:52:1...   3480
D  Sum   80         12-05 15:52:1...   3480
D  Sum   80         12-05 15:52:1...   3480
D  Sum   80         12-05 15:52:1...   3480
D  Sum   80         12-05 15:52:1...   3480
D  Sum   80         12-05 15:52:1...   3480
D  Sum   80         12-05 15:52:2...   3480
D  Sum   80         12-05 15:52:2...   3480
D  Sum   80         12-05 15:52:2...   3480
D  Sum   80         12-05 15:52:2...   3480
D  Sum   80         12-05 15:52:2...   3480
D  Sum   80         12-05 15:52:2...   3480
```

运行hook脚本



```
   Tag   Message    Time               PID    TI
D  Sum   7          12-05 16:15:3...   3930
D  Sum   7          12-05 16:15:4...   3930
D  Sum   7          12-05 16:15:4...   3930
D  Sum   7          12-05 16:15:4...   3930
D  Sum   7          12-05 16:15:4...   3930
D  Sum   7          12-05 16:15:4...   3930
D  Sum   7          12-05 16:15:4...   3930
D  Sum   7          12-05 16:15:4...   3930
D  Sum   7          12-05 16:15:4...   3930
D  Sum   7          12-05 16:15:4    3930
```

```
C:\Windows\System32\cmd.exe - python  loader.py
original call: fun(50, 30)
original call: fun(50, 30)
original call: fun(50, 30)
original call: fun(50, 30)
original call: fun(50, 30)
original call: fun(50, 30)
original call: fun(50, 30)
original call: fun(50, 30)
original call: fun(50, 30)
```

日志输出为hook脚本的给的参数，并在脚本中输出原始参数值。

frida hook 理论篇

Hook构造方法

要hook的是一个类的构造函数

```
public class Utils {
    public static Money getMoney() {
        return new Money(60, "RMB");
    }
...
}
```

我们可以通过`$init`来获取并修改一个类的构造方法。（注意，js是弱类型语言，而Java强类型，注意构造的时候的类型转换。）

获取参数可以通过自定义参数名也可以直接用系统隐含的arguments变量获取即可

下面是jscode

```
    var money=Java.use("com.XXXX.app.Money");
    money.$init.implementation = function(a, b)
    {
        console.log("Hook Start...");
        send(arguments[0]);
        send(arguments[1]);
        return this.$init(a,b);
    }
```

Hook重载方法

要hook的重载方法

```
public class Utils {
    public static String test() {
        return "this is test without argument";
    }

    public static String test(int num) {
        return "this is test with num "+num;
    }
...
}
```

在方法后面加一个`overload`属性，参数为函数的类型，类型用字符串传入，用于指定具体要hook的方法。例如`utils.test.overload("int").implementation`。注意，要填写类的路径，例如如果是字符串类型，则要用`overload("int","java.lang.String")` jscode:

```
utils.test.overload("int").implementation = function(a)
    {
        console.log("Hook Start...");
        send(arguments[0]);
        console.log("Hook Success...");
        return "This overload func is hooked";
    }
```

Hook对象参数的构造

要hook的以对象为参数的方法

```
package com.XXXX.app;

public class Utils {
    public static String test(Money money) {
        return money.getInfo();
    }
....
}
```

这里我们使用一个类型的 `$new` 来实例化一个类 jscode:

```
jscode = """
Java.perform(function(){
    var utils = Java.use("com.XXXX.app.Utils");
    var money=Java.use("com.XXXX.app.Money");

    utils.test.overload("com.XXXX.app.Money").implementation = function(a)
    {
        console.log("Hook Start...");
        send(a.getInfo());
        var m = money.$new(6666,"DOLLAR");
        send(m.getInfo());
        return m.getInfo();
        //return this.test(m);
    }
});
"""
```

修改对象属性的值

常规方法

- 如果 `obj.Attr` 的话，获取到的还是对象，例如 `a.name` 的返回值是 `[*] {'value': '美元', 'fieldType': 2, 'fieldReturnType': {'className': 'java.lang.String', 'name': 'Ljava/lang/String;', 'type': 'pointer', 'size': 1}} [*] 美元`
- 我们使用 `a.name.value` 就能获取对象属性的值了。 例如

```
utils.test.overload("com.xiaojianbang.app.Money").implementation = function(a)
    {
        console.log("Hook Start...");
        send(a.name); //object
        send(a.name.value); //real value
        var m = money.$new(6666,"DOLLAR");
        m.name.value="ForeignMoney";
        send(m.getInfo());
        return m.getInfo();
        //return this.test(m);
    }
```

Java反射

对于私有属性，我们可以用Java反射的方法设置。 在Java反射中，通过 `Java.cast(m.getClass(),clazz).getDeclaredField('num')`，m为一个实例化对象，后面 `getDeclaredField` 可以获得属性。通过 `Java.use('java.lang.Class');` 获取类的构造器 这里，通过属性的 `get(ObjectsInstantiated)` 可以获取值，通过属性的 `setInt(ObjectsInstantiated, value)` 可以设置一个对象的属性值。 `ObjectsInstantiated` 为一个对象。 这里，对于反射后的值，用 `console.log` 可以很好的输出值，而 `send` 在此处会打印对象。

```
Java.perform(function(){
    var utils = Java.use("com.XXXX.app.Utils");
    var money=Java.use("com.XXXX.app.Money");

    utils.test.overload("com.XXXX.app.Money").implementation = function(a)
    {
        console.log("Hook Start...");
        var m = money.$new(6666,"DOLLAR");
        var clazz = Java.use('java.lang.Class');
        var num_id = Java.cast(m.getClass(),clazz).getDeclaredField('num');
        num_id.setAccessible(true);
        var value = num_id.get(m);
        console.log(value);
        send(value); // have format problem
        num_id.setInt(m,23333);
        console.log(num_id.get(m));
        return this.test(m);
    }
});
```

Hook内部类

要hook的内部类。

在 Java 中，可以将一个类定义在另一个类里面或者一个方法里面，这样的类称为内部类。

```
class Circle {
    double radius = 0;

    public Circle(double radius) {
        this.radius = radius;
    }

    class Draw {     //内部类
        public void drawSahpe() {
            System.out.println("drawshape");
        }
    }
}
```

在获取要hook的类后加 `$` 和内部类名。

```
var inInnerClass = Java.use('com.XXXX.app.Circle$Draw');

inInnerClass.drawSahpe.implementation = function()
{
  ...
}
```

打印方法堆栈信息

用Java.use方法获取类型变量：var Exception = Java.use("java.lang.Exception")

然后是js中支持throw语法的，直接在需要打印堆栈信息的方法中调用即可。

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

总结

Java层代码Hook操作

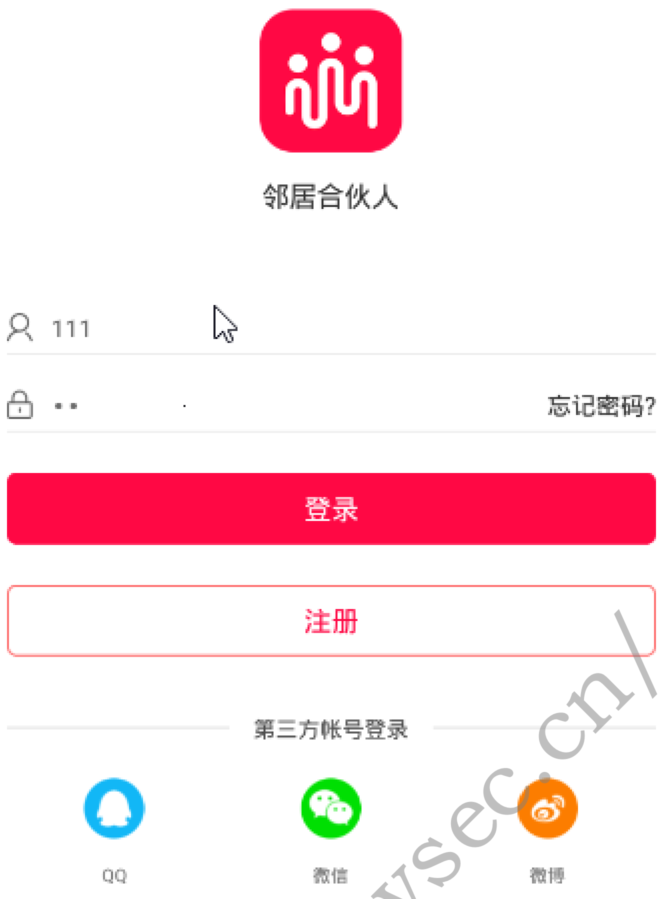
- 1、hook方法包括构造方法和对象方法，构造方法固定写法是$init，普通方法直接是方法名，参数可以自己定义也可以使用系统隐含的变量arguments获取。
- 2、修改方法的参数和返回值，直接调用原始方法通过传入想要修改的参数来做到修改参数的目的，以及修改返回值即可。
- 3、构造对象和修改对象的属性值，直接用反射进行操作，构造对象用固定写法的$new即可。
- 4、直接用Java的Exception对象打印堆栈信息，然后通过adb logcat -s AndroidRuntime来查看异常信息跟踪代码。

总结：获取对象的类类型是Java.use方法，方法有重载的话用overload(.......)解决。

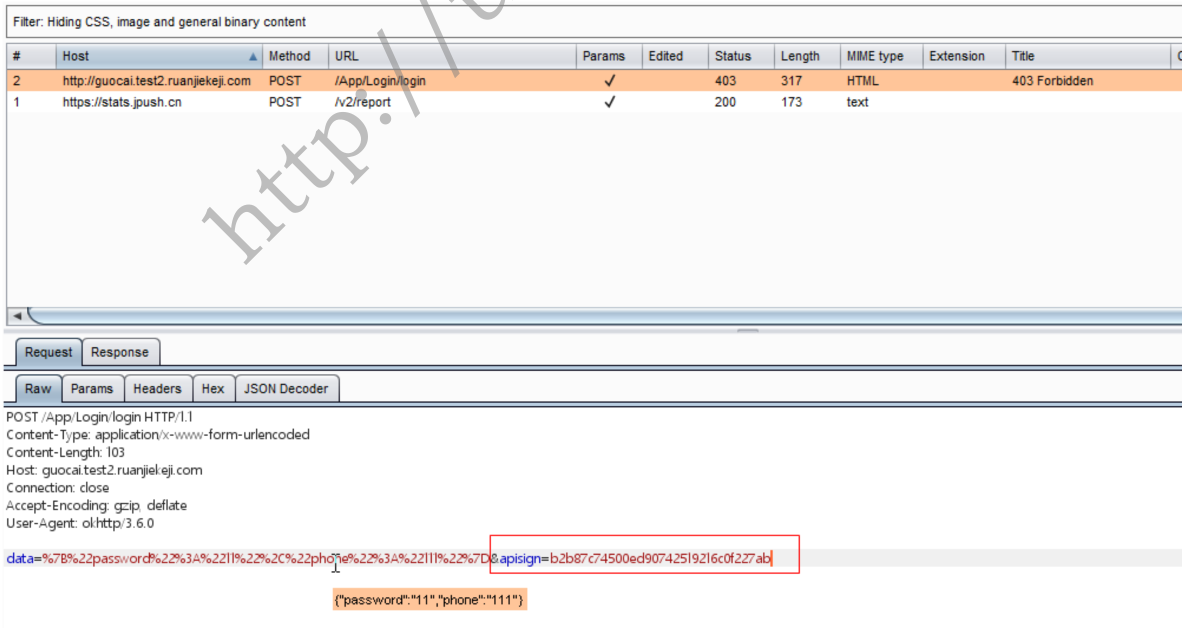
frida hook 实战篇

邻居合伙人Hook

对邻居合伙人APP登录sign签名算法的hook。



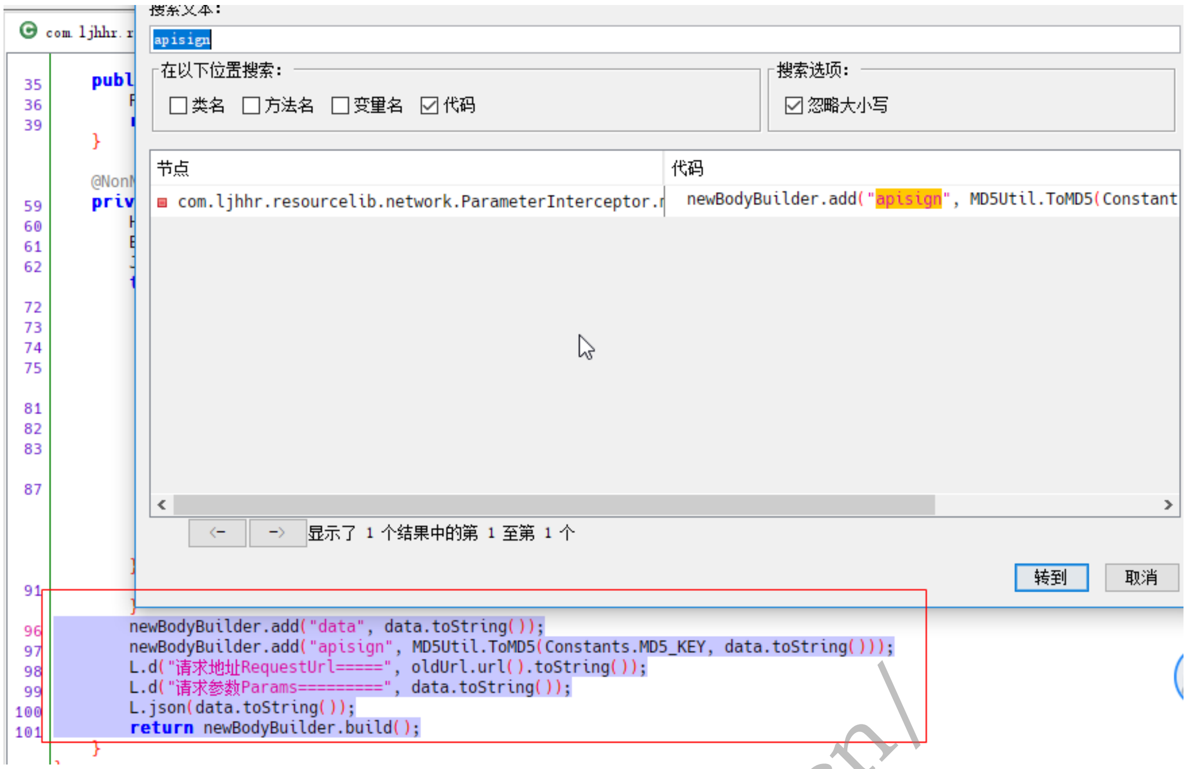
在登录处进行抓包操作，可以看到登录数据报中有apisign字段。



#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title
2	http://guocai.test2.ruanjiekeji.com	POST	/App/Login/login	✓		403	317	HTML		403 Forbidden
1	https://stats.jpush.cn	POST	/v2/report	✓		200	173	text		

```
POST /App/Login/login HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 103
Host: guocai.test2.ruanjiekeji.com
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/3.6.0

data=%7B%22password%22%3A%2211%22%2C%22phone%22%3A%22111%22%7D&apisign=b2b87c74500ed907425l9216c0f227ab

{"password":"11","phone":"111"}
```

对APK进行反编译，搜索apisign关键字。


搜索文本:
apisign
在以下位置搜索: ☐类名 ☐方法名 ☐变量名 ☑代码
搜索选项: ☑忽略大小写

节点	代码
com.ljhhr.resourcelib.network.ParameterInterceptor.r	newBodyBuilder.add("apisign", MD5Util.ToMD5(Constant

显示了 1 个结果中的第 1 至第 1 个

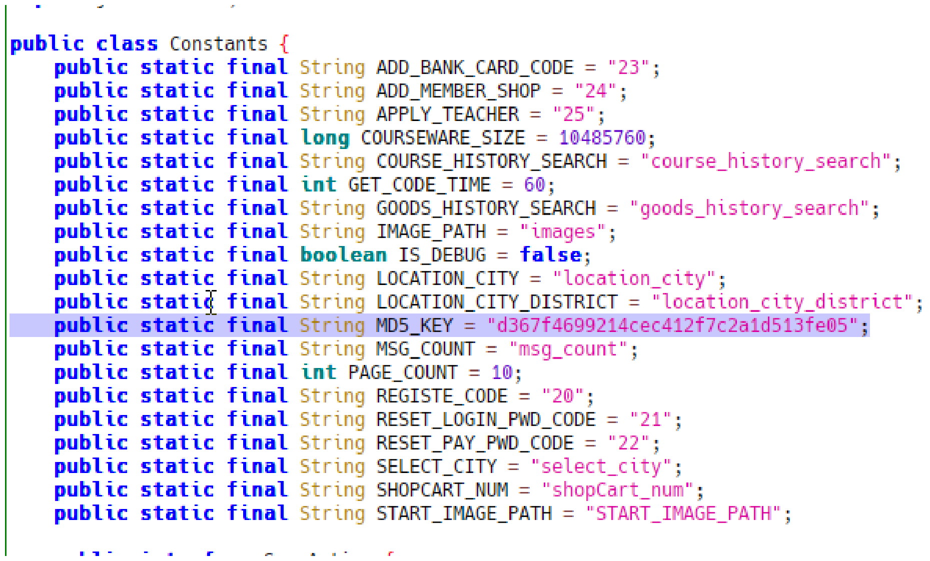
```
        newBodyBuilder.add("data", data.toString());
        newBodyBuilder.add("apisign", MD5Util.ToMD5(Constants.MD5_KEY, data.toString()));
        L.d("请求地址RequestUrl=====", oldUrl.url().toString());
        L.d("请求参数Params=========", data.toString());
        L.json(data.toString());
        return newBodyBuilder.build();
```

```
        newBodyBuilder.add("data", data.toString());
        newBodyBuilder.add("apisign", MD5Util.ToMD5(Constants.MD5_KEY,
data.toString()));
        L.d("请求地址RequestUrl=====", oldUrl.url().toString());
        L.d("请求参数Params=========", data.toString());
        L.json(data.toString());
        return newBodyBuilder.build();
```

根据代码可以看到，apisign内容主要是通过MD5Util类的ToMD5()方法生成的，跳转到ToMD5()方法内容：

```
public static String ToMD5(String secretKey, String pstr) {
    pstr = secretKey + pstr;
    char[] hexDigits = new char[]{'0', '1', '2', '3', '4', '5', '6', '7', '8',
'9', 'A', 'B', 'C', 'D', 'E', 'F'};
    try {
        MessageDigest md5Temp = MessageDigest.getInstance("MD5");
        md5Temp.update(pstr.getBytes("UTF8"));
        char[] str = new char[(j * 2)];
        int k = 0;
        for (byte byte0 : md5Temp.digest()) {
            int i = k + 1;
            str[k] = hexDigits[(byte0 >>> 4) & 15];
            k = i + 1;
            str[i] = hexDigits[byte0 & 15];
        }
        return new String(str).toLowerCase();
    } catch (Exception e) {
        return null;
    }
}
```

这个方法有两个参数，一个secretkey和pstr。且为普通函数。通过静态分析，MD5Util.ToMD5()传入两个值，一个是Constants.MD5_KEY,跳转发现是Constants类的静态变量。



```
public class Constants {
    public static final String ADD_BANK_CARD_CODE = "23";
    public static final String ADD_MEMBER_SHOP = "24";
    public static final String APPLY_TEACHER = "25";
    public static final long COURSEWARE_SIZE = 10485760;
    public static final String COURSE_HISTORY_SEARCH = "course_history_search";
    public static final int GET_CODE_TIME = 60;
    public static final String GOODS_HISTORY_SEARCH = "goods_history_search";
    public static final String IMAGE_PATH = "images";
    public static final boolean IS_DEBUG = false;
    public static final String LOCATION_CITY = "location_city";
    public static final String LOCATION_CITY_DISTRICT = "location_city_district";
    public static final String MD5_KEY = "d367f4699214cec412f7c2a1d513fe05";
    public static final String MSG_COUNT = "msg_count";
    public static final int PAGE_COUNT = 10;
    public static final String REGISTE_CODE = "20";
    public static final String RESET_LOGIN_PWD_CODE = "21";
    public static final String RESET_PAY_PWD_CODE = "22";
    public static final String SELECT_CITY = "select_city";
    public static final String SHOPCART_NUM = "shopCart_num";
    public static final String START_IMAGE_PATH = "START_IMAGE_PATH";
```

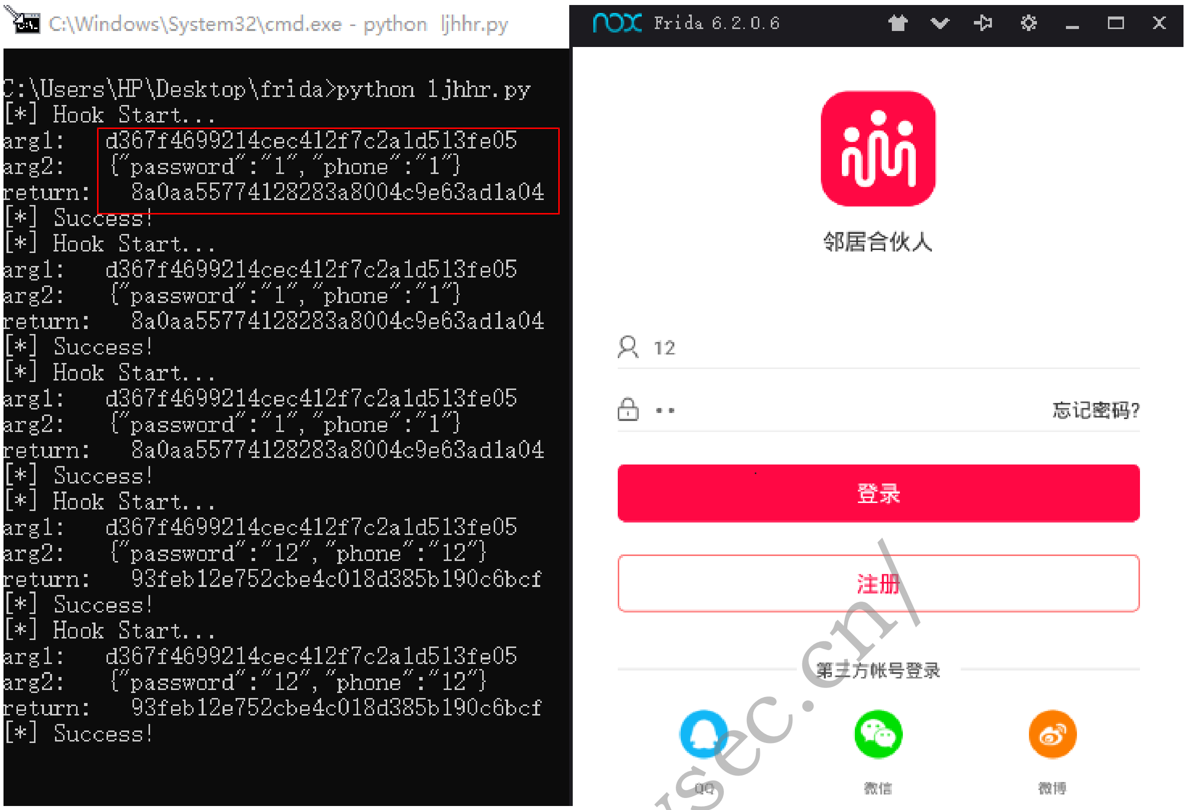
其值为：d367f4699214cec412f7c2a1d513fe05。另外一个变量则是app登录信息。

而用frida主要是对MD5Util.ToMD5()方法的两个变量进行hook。

编写如下hook脚本。

```
import frida
import sys
import time

jscode = """
Java.perform(function () {
    var md5 = Java.use('com.softgarden.baselibrary.utils.MD5Util');
    md5.ToMD5.implementation = function (a, b) {
        send("Hook Start...");
        console.log("arg1:   " + a );
        console.log("arg2:   " + b);
        var res = this.ToMD5(a, b);
        console.log("return:    " + res);
        send("Success!");
        return res;
    }
});
"""


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


device = frida.get_remote_device()
pid = device.spawn(["com.ljhhr.mobile"])
device.resume(pid)
time.sleep(1)  # Without it Java.perform silently fails
session = device.attach(pid)
script = session.create_script(jscode)
script.on("message", message)
script.load()
sys.stdin.read()
```

运行如上脚本进行hook，可以看到如下结果。



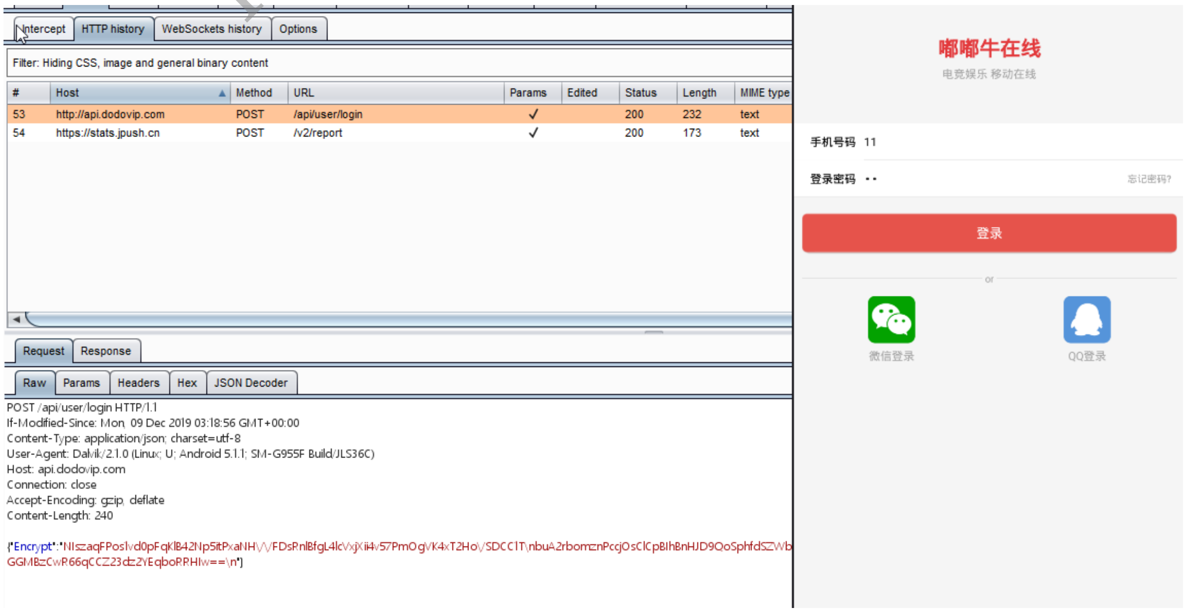
```
C:\Users\HP\Desktop\frida>python ljhhr.py
[*] Hook Start...
arg1:   d367f4699214cec412f7c2a1d513fe05
arg2:    {"password":"1","phone":"1"}
return:   8a0aa55774128283a8004c9e63ad1a04
[*] Success!
[*] Hook Start...
arg1:   d367f4699214cec412f7c2a1d513fe05
arg2:    {"password":"1","phone":"1"}
return:   8a0aa55774128283a8004c9e63ad1a04
[*] Success!
[*] Hook Start...
arg1:   d367f4699214cec412f7c2a1d513fe05
arg2:    {"password":"1","phone":"1"}
return:   8a0aa55774128283a8004c9e63ad1a04
[*] Success!
[*] Hook Start...
arg1:   d367f4699214cec412f7c2a1d513fe05
arg2:    {"password":"12","phone":"12"}
return:   93feb12e752cbe4c018d385b190c6bcf
[*] Success!
[*] Hook Start...
arg1:   d367f4699214cec412f7c2a1d513fe05
arg2:    {"password":"12","phone":"12"}
return:   93feb12e752cbe4c018d385b190c6bcf
[*] Success!
```

通过多次hook可以看到第一个变量为固定的，即我们静态分析代码得到的：d367f4699214cec412f7c2a1d513fe05

第二个参数为输入登录的输入内容。至此，完成对该APP的sign算法的hook。

嘟嘟牛在线Hook

同样在登录界面，进行抓包操作。

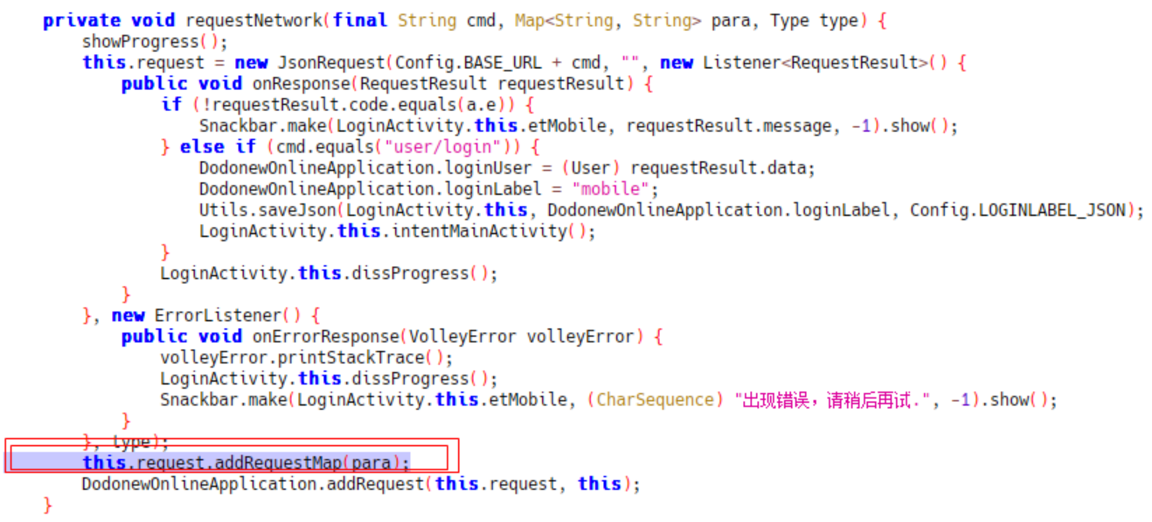


#	Host	Method	URL	Params	Edited	Status	Length	MIME type
53	http://api.dodovip.com	POST	/api/user/login	✓		200	232	text
54	https://stats.jpush.cn	POST	/v2/report	✓		200	173	text

```
POST /api/user/login HTTP/1.1
If-Modified-Since: Mon, 09 Dec 2019 03:18:56 GMT+00:00
Content-Type: application/json; charset=utf-8
User-Agent: Dalvik/2.1.0 (Linux; U; Android 5.1.1; SM-G955F Build/JLS36C)
Host: api.dodovip.com
Connection: close
Accept-Encoding: gzip, deflate
Content-Length: 240

{"Encrypt":"NIszaqFPos1vd0pFqKlB42Np5itPxaNH\/\/FDsRnlBfgL4lcVxjXii4v57PmOgVK4xT2Ho\/SDCC1T\nbuA2rbomznPccjOsClCpBIhBnHJD9QoSphfdSZWb
GGMBzCwR66qCCZ23dz2YEqboRRHIw==\n"}
```

同时对apk进行反编译。可以搜索url路径关键字：user/login，也可以搜索数据报关键字：Encrypt。

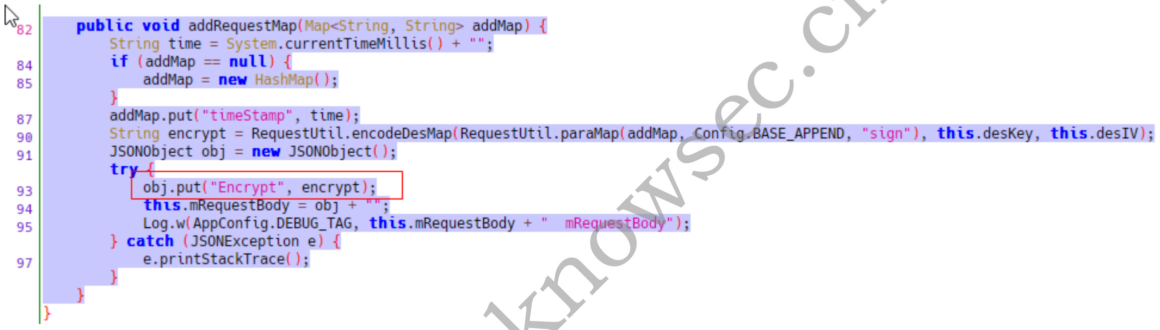
搜索user/login定位到类名：com.dodonew.online.ui.LoginActivity中的requestNetwork方法。



```
private void requestNetwork(final String cmd, Map<String, String> para, Type type) {
    showProgress();
    this.request = new JsonRequest(Config.BASE_URL + cmd, "", new Listener<RequestResult>() {
        public void onResponse(RequestResult requestResult) {
            if (!requestResult.code.equals(a.e)) {
                Snackbar.make(LoginActivity.this.etMobile, requestResult.message, -1).show();
            } else if (cmd.equals("user/login")) {
                DodonewOnlineApplication.loginUser = (User) requestResult.data;
                DodonewOnlineApplication.loginLabel = "mobile";
                Utils.saveJson(LoginActivity.this, DodonewOnlineApplication.loginLabel, Config.LOGINLABEL_JSON);
                LoginActivity.this.intentMainActivity();
            }
            LoginActivity.this.dissProgress();
        }
    }, new ErrorListener() {
        public void onErrorResponse(VolleyError volleyError) {
            volleyError.printStackTrace();
            LoginActivity.this.dissProgress();
            Snackbar.make(LoginActivity.this.etMobile, (CharSequence) "出现错误，请稍后再试.", -1).show();
        }
    }, type);
    this.request.addRequestMap(para);
    DodonewOnlineApplication.addRequest(this.request, this);
}
```

分析代码，可知数据报中的内容通过addRequestMap方法添加。跳转到com.dodonew.online.http.JsonRequest类的addRequestMap方法。

定位到我们在数据报中看到的关键字：Encrypt。



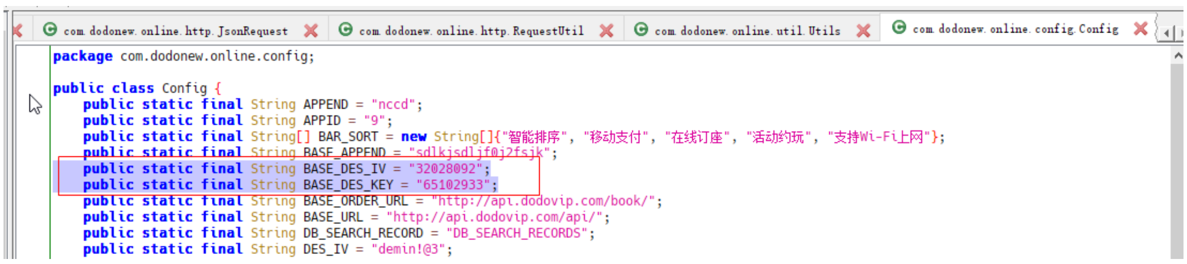
```
public void addRequestMap(Map<String, String> addMap) {
    String time = System.currentTimeMillis() + "";
    if (addMap == null) {
        addMap = new HashMap();
    }
    addMap.put("timeStamp", time);
    String encrypt = RequestUtil.encodeDesMap(RequestUtil.paraMap(addMap, Config.BASE_APPEND, "sign"), this.desKey, this.desIV);
    JSONObject obj = new JSONObject();
    try {
        obj.put("Encrypt", encrypt);
        this.mRequestBody = obj + "";
        Log.w(AppConfig.DEBUG_TAG, this.mRequestBody + "  mRequestBody");
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
}
```

此处的encrypt即加密后的内容。

```
String encrypt = RequestUtil.encodeDesMap(RequestUtil.paraMap(addMap,
Config.BASE_APPEND, "sign"), this.desKey, this.desIV);
```

通过des算法进行加密后的数据。该方法为encodeDesMap。其有三个参数。

跟踪变量desKey，desIV。跳转到com.dodonew.online.config.Config类。

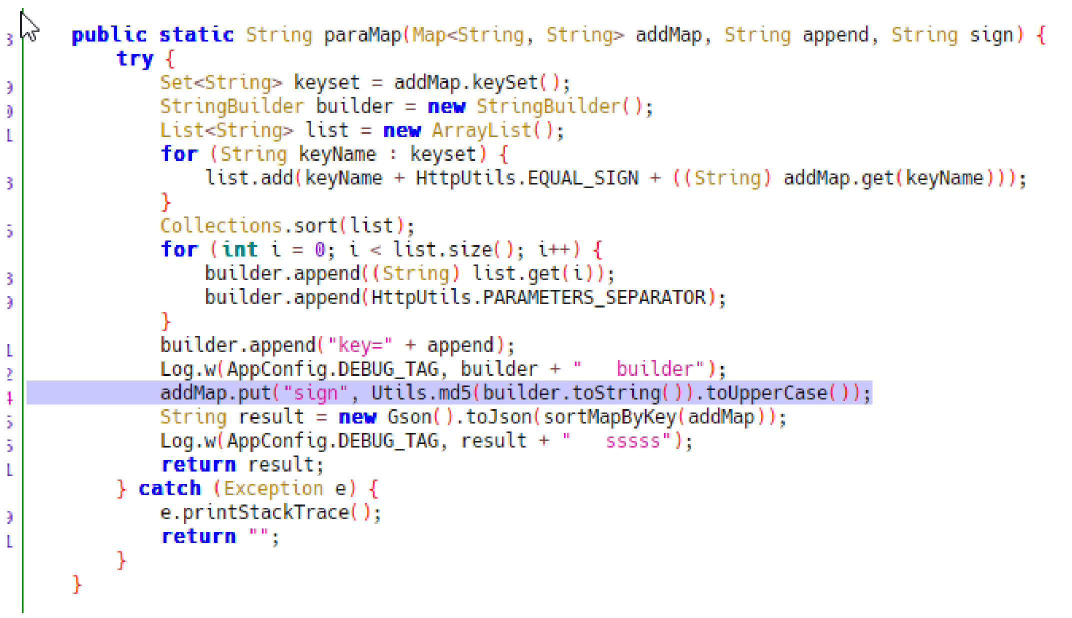

com.dodonew.online.http.JsonRequest
com.dodonew.online.http.RequestUtil
com.dodonew.online.util.Utils
com.dodonew.online.config.Config

```
package com.dodonew.online.config;

public class Config {
    public static final String APPEND = "nccd";
    public static final String APPID = "9";
    public static final String[] BAR_SORT = new String[]{"智能排序", "移动支付", "在线订座", "活动约玩", "支持Wi-Fi上网"};
    public static final String BASE_APPEND = "sdlkjsdljf0j2fsjk";
    public static final String BASE_DES_IV = "32028092";
    public static final String BASE_DES_KEY = "65102933";
    public static final String BASE_ORDER_URL = "http://api.dodovip.com/book/";
    public static final String BASE_URL = "http://api.dodovip.com/api/";
    public static final String DB_SEARCH_RECORD = "DB_SEARCH_RECORDS";
    public static final String DES_IV = "demin!@3";
```

可以得到des加密算法的key值为65102933，偏移值IV为：32028092。

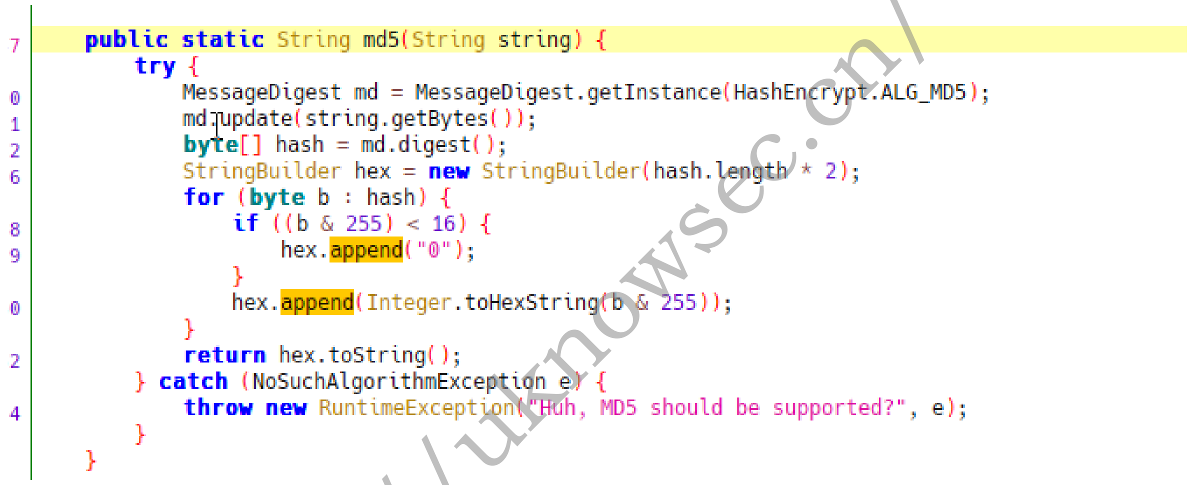
跟踪encodeDesMap方法的第一个参数。RequestUtil.paraMap(addMap, Config.BASE_APPEND, "sign")

跟进paraMap方法到com.dodonew.online.http.RequestUtil类。



```java
public static String paraMap(Map<String, String> addMap, String append, String sign) {
    try {
        Set<String> keyset = addMap.keySet();
        StringBuilder builder = new StringBuilder();
        List<String> list = new ArrayList();
        for (String keyName : keyset) {
            list.add(keyName + HttpUtils.EQUAL_SIGN + ((String) addMap.get(keyName)));
        }
        Collections.sort(list);
        for (int i = 0; i < list.size(); i++) {
            builder.append((String) list.get(i));
            builder.append(HttpUtils.PARAMETERS_SEPARATOR);
        }
        builder.append("key=" + append);
        Log.w(AppConfig.DEBUG_TAG, builder + "   builder");
        addMap.put("sign", Utils.md5(builder.toString()).toUpperCase());
        String result = new Gson().toJson(sortMapByKey(addMap));
        Log.w(AppConfig.DEBUG_TAG, result + "   sssss");
        return result;
    } catch (Exception e) {
        e.printStackTrace();
        return "";
    }
}
```

发现存在sign,跟进md5方法到com.dodonew.online.util.Utils类。



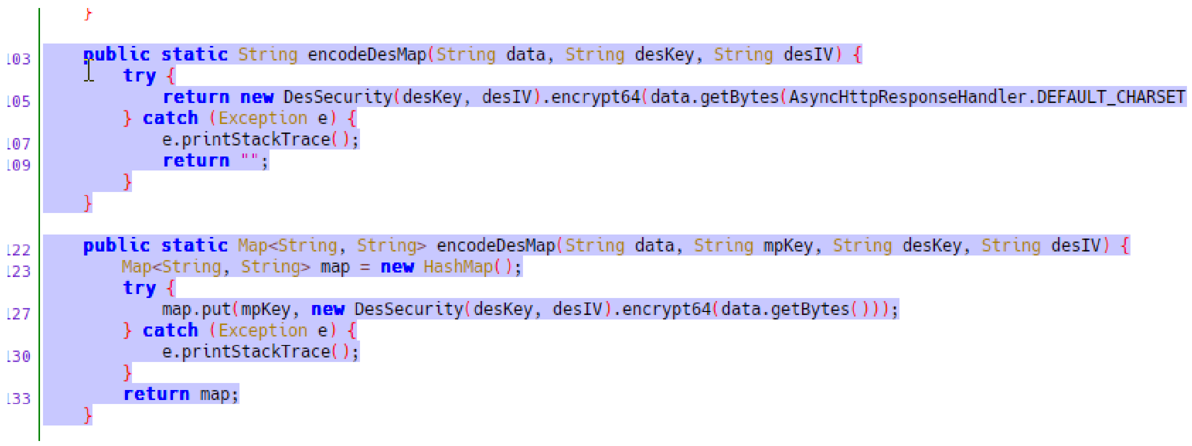
```java
public static String md5(String string) {
    try {
        MessageDigest md = MessageDigest.getInstance(HashEncrypt.ALG_MD5);
        md.update(string.getBytes());
        byte[] hash = md.digest();
        StringBuilder hex = new StringBuilder(hash.length * 2);
        for (byte b : hash) {
            if ((b & 255) < 16) {
                hex.append("0");
            }
            hex.append(Integer.toHexString(b & 255));
        }
        return hex.toString();
    } catch (NoSuchAlgorithmException e) {
        throw new RuntimeException("Huh, MD5 should be supported?", e);
    }
}
```

可以看到md5是一个普通的方法。如果需要进行hook，直接hook参数值和返回值就行了。

hook部分的js代码如下：

```javascript
var Utils = Java.use('com.dodonew.online.util.Utils');
Utils.md5.implementation = function (a) {
    send("Hook Start md5...");
    console.log("md5-arg:    " + a);
    var result = this.md5(a);
    console.log("md5-result:    " + result);
    send("Success!");
    return result;
}
```

而com.dodonew.online.http.RequestUtil类下的encodeDesMap方法是一个重载方法。



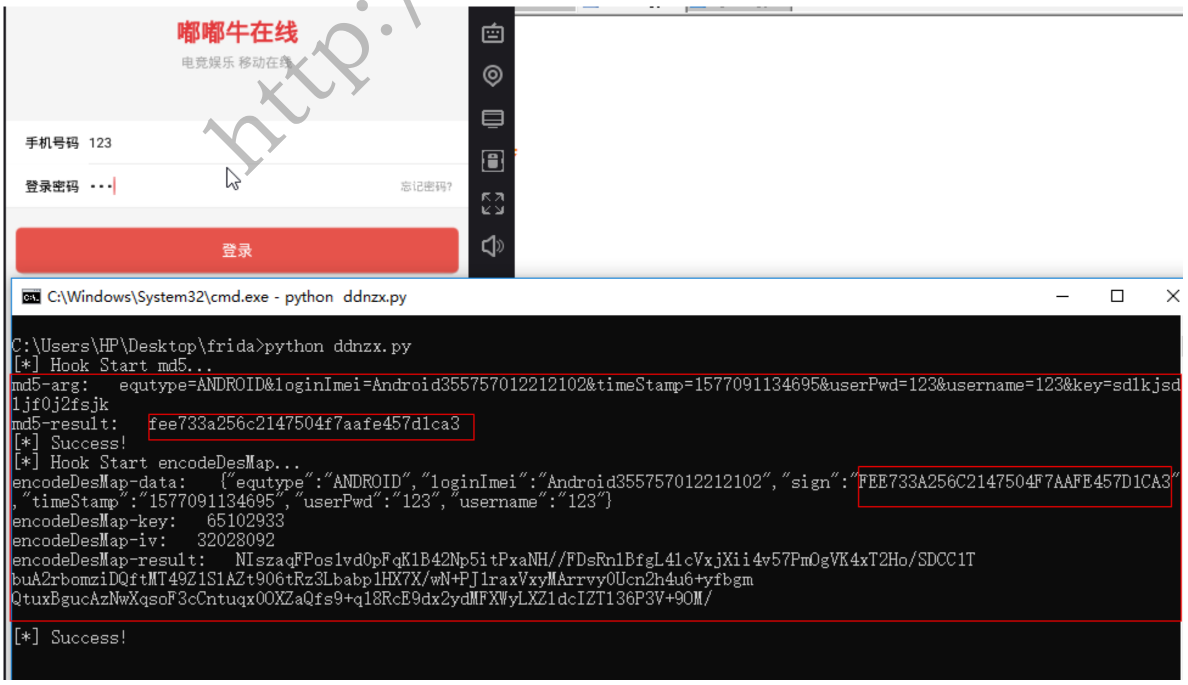
```
    }
    public static String encodeDesMap(String data, String desKey, String desIV) {
        try {
            return new DesSecurity(desKey, desIV).encrypt64(data.getBytes(AsyncHttpResponseHandler.DEFAULT_CHARSET
        } catch (Exception e) {
            e.printStackTrace();
            return "";
        }
    }

    public static Map<String, String> encodeDesMap(String data, String mpKey, String desKey, String desIV) {
        Map<String, String> map = new HashMap();
        try {
            map.put(mpKey, new DesSecurity(desKey, desIV).encrypt64(data.getBytes()));
        } catch (Exception e) {
            e.printStackTrace();
        }
        return map;
    }
```

所以在传入需要hook的方法是要用到overload。且这里的重载参数类型为String。所以要用overload('java.lang.String','java.lang.String','java.lang.String')。因为String在Java中是以类的形式存在的数据类型。同时String类在java.lang包中。 故此处的js代码为：

```
var RequestUtil = Java.use('com.dodonew.online.http.RequestUtil');
RequestUtil.encodeDesMap.overload('java.lang.String','java.lang.String','java.la
ng.String').implementation = function (data, key ,iv) {
        send("Hook Start encodeDesMap...");
        console.log("encodeDesMap-data:   " + data);
        console.log("encodeDesMap-key:   " + key);
        console.log("encodeDesMap-iv:   " + iv);
        var result = this.encodeDesMap(data, key ,iv);
        console.log("encodeDesMap-result:   " + result);
        send("Success!");
        return result;
    }
```

python通用hook模版加载两段js代码即可对该APP进行hook。

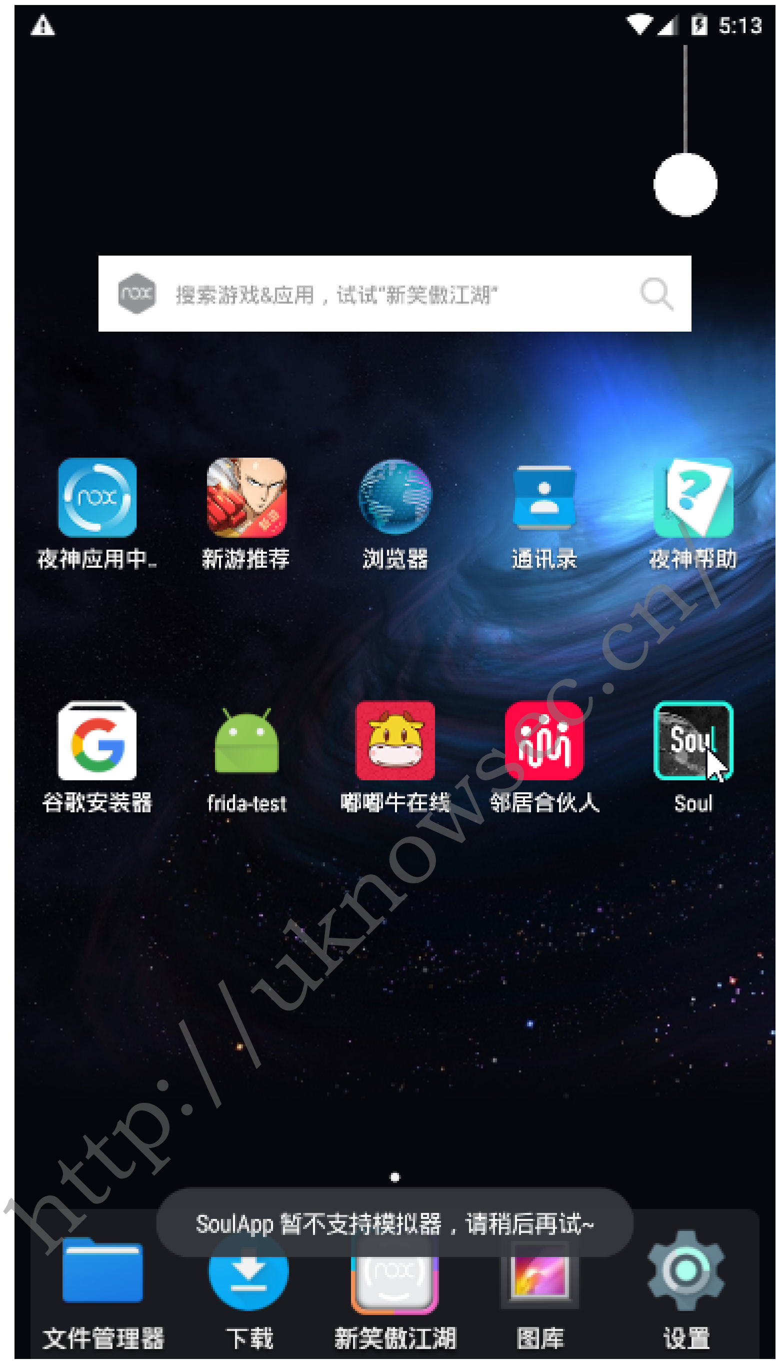
成功hook到所有参数。



```
C:\Users\HP\Desktop\frida>python ddnzx.py
[*] Hook Start md5...
md5-arg:   equtype=ANDROID&loginImei=Android355757012212102&timeStamp=1577091134695&userPwd=123&username=123&key=sdlkjsd
1jf0j2fsjk
md5-result:   fee733a256c2147504f7aafe457d1ca3
[*] Success!
[*] Hook Start encodeDesMap...
encodeDesMap-data:   {"equtype":"ANDROID","loginImei":"Android355757012212102","sign":"FEE733A256C2147504F7AAFE457D1CA3"
,"timeStamp":"1577091134695","userPwd":"123","username":"123"}
encodeDesMap-key:   65102933
encodeDesMap-iv:   32028092
encodeDesMap-result:   NIszaqFPos1vd0pFqK1B42Np5itPxaNH//FDsRn1BfgL41cVxjXii4v57PmOgVK4xT2Ho/SDCC1T
buA2rbomziDQftMT49Z1S1AZt906tRz3Lbabp1HX7X/wN+PJ1raxVxyMArrvy0Ucn2h4u6+yfbgm
QtuxBgucAzNwXqsoF3cCntuqx0OXZaQfs9+q18RcE9dx2ydMFXWyLXZ1dcIZT136P3V+9OM/

[*] Success!
```

Soul Hook

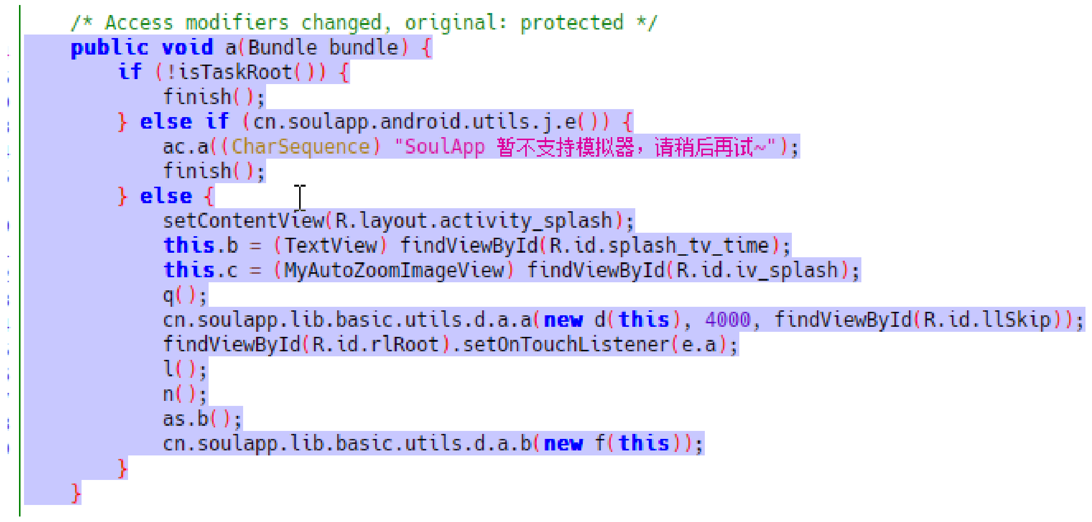
Soul是一款社交app，此APP在模拟器中是无法启动的,如下：



SoulApp 暂不支持模拟器，请稍后再试~

显示`SoulApp暂不支持模拟器，请稍后再试~`

搜索关键字`SoulApp暂不支持模拟器，请稍后再试~`定位到cn.soulapp.android.ui.splash.SplashActivity类下。



```
/* Access modifiers changed, original: protected */
public void a(Bundle bundle) {
    if (!isTaskRoot()) {
        finish();
    } else if (cn.soulapp.android.utils.j.e()) {
        ac.a((CharSequence) "SoulApp 暂不支持模拟器，请稍后再试~");
        finish();
    } else {
        setContentView(R.layout.activity_splash);
        this.b = (TextView) findViewById(R.id.splash_tv_time);
        this.c = (MyAutoZoomImageView) findViewById(R.id.iv_splash);
        q();
        cn.soulapp.lib.basic.utils.d.a.a(new d(this), 4000, findViewById(R.id.llSkip));
        findViewById(R.id.rlRoot).setOnTouchListener(e.a);
        l();
        n();
        as.b();
        cn.soulapp.lib.basic.utils.d.a.b(new f(this));
    }
}
```

可见此处判断if (cn.soulapp.android.utils.j.e())返回的是true。

跟进方法e到类。cn.soulapp.android.utils.j

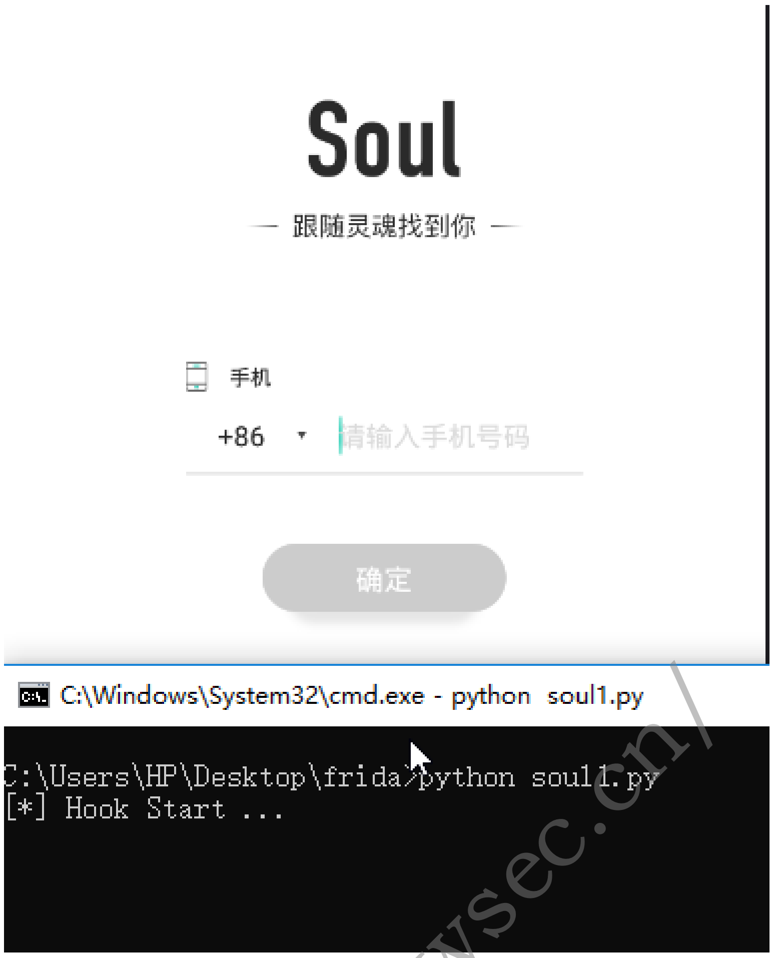
```
    public static boolean e() {
        try {
            SoulApp b = SoulApp.b();
            Intent intent = new Intent();
            intent.setData(Uri.parse("tel:123456"));
            intent.setAction("android.intent.action.DIAL");
            return Build.FINGERPRINT.startsWith("generic") ||
Build.FINGERPRINT.toLowerCase().contains("vbox") ||
Build.FINGERPRINT.toLowerCase().contains("test-keys") ||
Build.MODEL.contains("google_sdk") || Build.MODEL.contains("Emulator") ||
Build.SERIAL.equalsIgnoreCase("unknown") ||
Build.SERIAL.equalsIgnoreCase(DispatchConstants.ANDROID) ||
Build.MODEL.contains("Android SDK built for x86") ||
Build.MANUFACTURER.contains("Genymotion") || ((Build.BRAND.startsWith("generic")
&& Build.DEVICE.startsWith("generic")) || "google_sdk".equals(Build.PRODUCT) ||
((TelephonyManager)
b.getSystemService("phone")).getNetworkOperatorName().toLowerCase().equals(Dispa
tchConstants.ANDROID) || (intent.resolveActivity(b.getPackageManager()) != null
? 1 : null) == null);
        } catch (Exception e) {
            return false;
        }
    }
```

这里我们只需要把e方法的返回值改为false，即可跳过模拟器验证。

js代码如下：

```
    var j = Java.use('cn.soulapp.android.utils.j');
    j.e.implementation = function (a) {
        send("Hook Start ...");
        return false;
    }
```

运行hook脚本，成功进入登录界面



```
C:\Users\HP\Desktop\frida>python soul1.py
[*] Hook Start ...
```

Traceview+frida

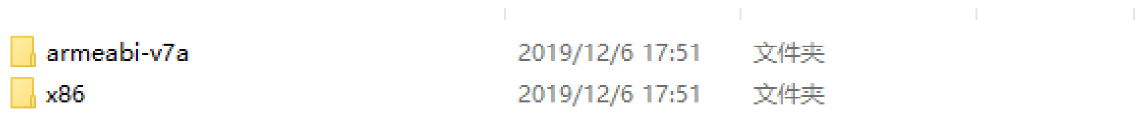
TraceView 是 Android SDK 中内置的一个工具，它可以加载 **trace** 文件，用图形的形式展示**代码的执行时间、次数及调用栈。**

利用mprop工具修改当前手机应用都可以调试

使用DDMS时，只能看到手机，看不到进程信息，这样我们就不能获取指定进程的信息

如果需要调试android 的程序，以下两个条件满足一个就行。第一是apk的配置文件内的AndroidManifest.xml的 android:debuggable="true"，第二就是/default.prop中ro.debuggable=1。两种方式第一种通常是解包添加属性再打包，随着加壳软件以及apk校验等，容易出现安装包异常。第二种由于一般的手机发布时ro.debuggable一般是0 也就是不允许调试，通过修改rom的办法在手机上比较麻烦，需要刷机等等，模拟器上一般是vmdk的虚拟机，也没法修改rom。

这里我们可以直接用mprop这个工具，可以直接修改android属性。



armeabi-v7a	2019/12/6 17:51	文件夹
x86	2019/12/6 17:51	文件夹

同样有不同的版本，通过adb上传到模拟机，修改属性值。

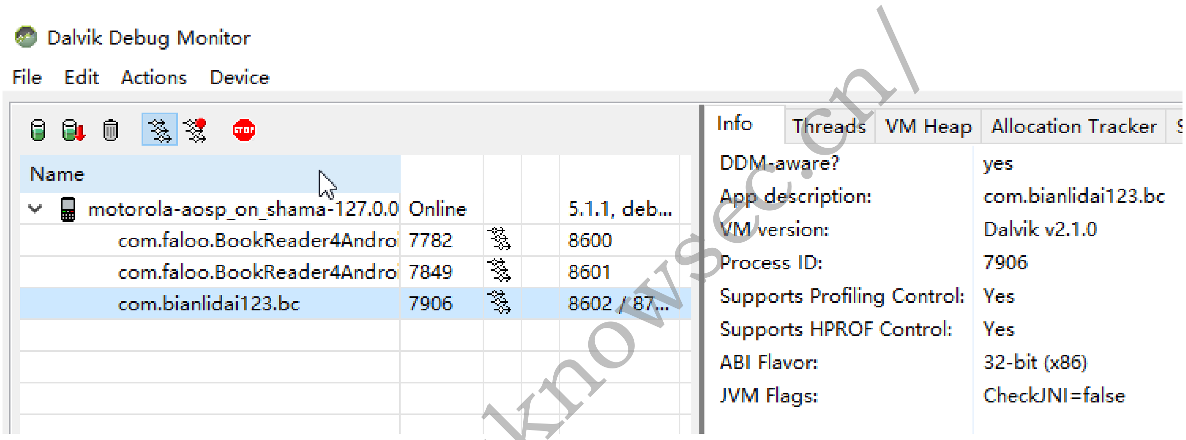
arm:

```
adb push .\libs\armeabi-v7a\mprop /data/local/tmp/
adb shell "chmod 755 /data/local/tmp/mprop"
adb shell "/data/local/tmp/mprop"
adb shell "setprop ro.debuggable 1"
```

x86:

```
adb push .\libs\x86\mprop /data/local/tmp/
adb shell "chmod 755 /data/local/tmp/mprop"
adb shell "/data/local/tmp/mprop"
adb shell "setprop ro.debuggable 1"
```

这样我们就能看到进行信息了



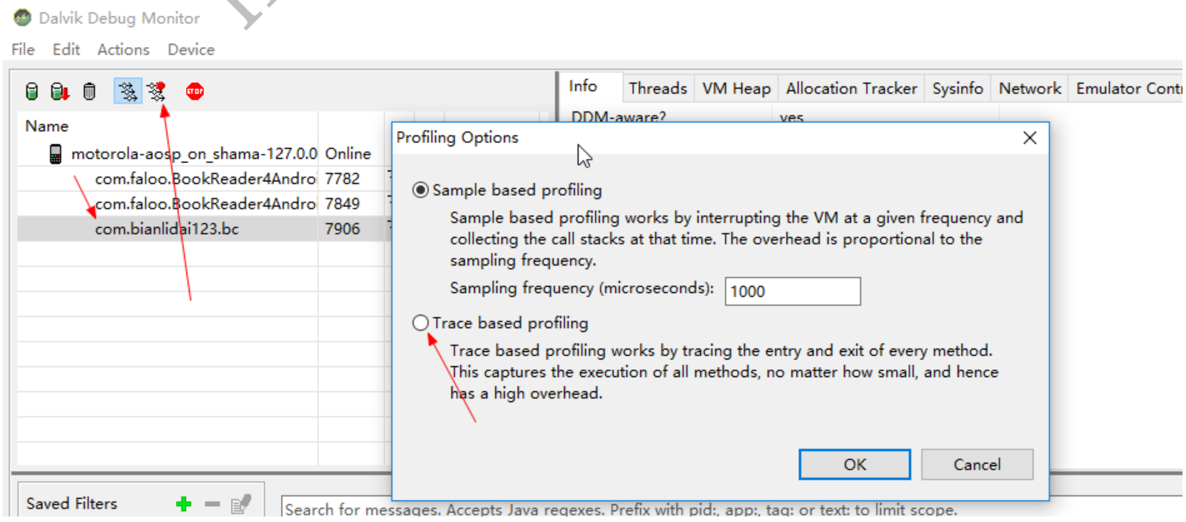
Name			
motorola-aosp_on_shama-127.0.0	Online		5.1.1, deb...
com.faloo.BookReader4Andro	7782		8600
com.faloo.BookReader4Andro	7849		8601
com.bianlidai123.bc	7906		8602 / 87...

DDM-aware?	yes
App description:	com.bianlidai123.bc
VM version:	Dalvik v2.1.0
Process ID:	7906
Supports Profiling Control:	Yes
Supports HPROF Control:	Yes
ABI Flavor:	32-bit (x86)
JVM Flags:	CheckJNI=false

TraceView 追踪函数

这里用到的是DDMS中的traceview工具，直接下载android-sdk工具包，运行里面的ddms.bat即可。

如图选定进程名，选择Trace based profiling。



Name	
motorola-aosp_on_shama-127.0.0	Online
com.faloo.BookReader4Andro	7782
com.faloo.BookReader4Andro	7849
com.bianlidai123.bc	7906

Profiling Options

[✓ Sample based profiling]

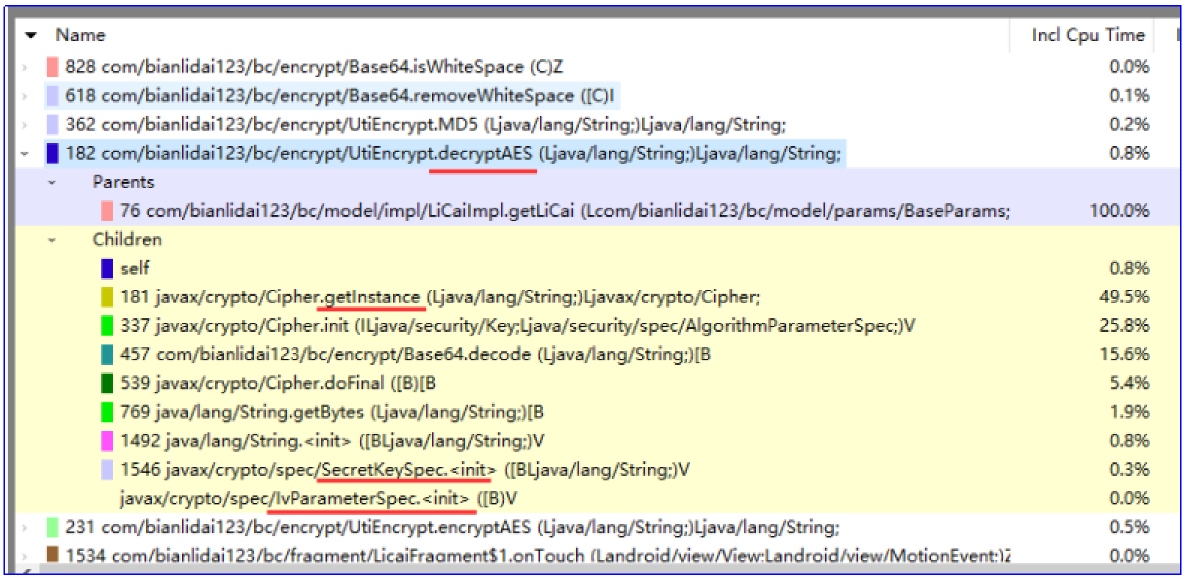
Sample based profiling works by interrupting the VM at a given frequency and collecting the call stacks at that time. The overhead is proportional to the sampling frequency.

Sampling frequency (microseconds): 1000

Trace based profiling

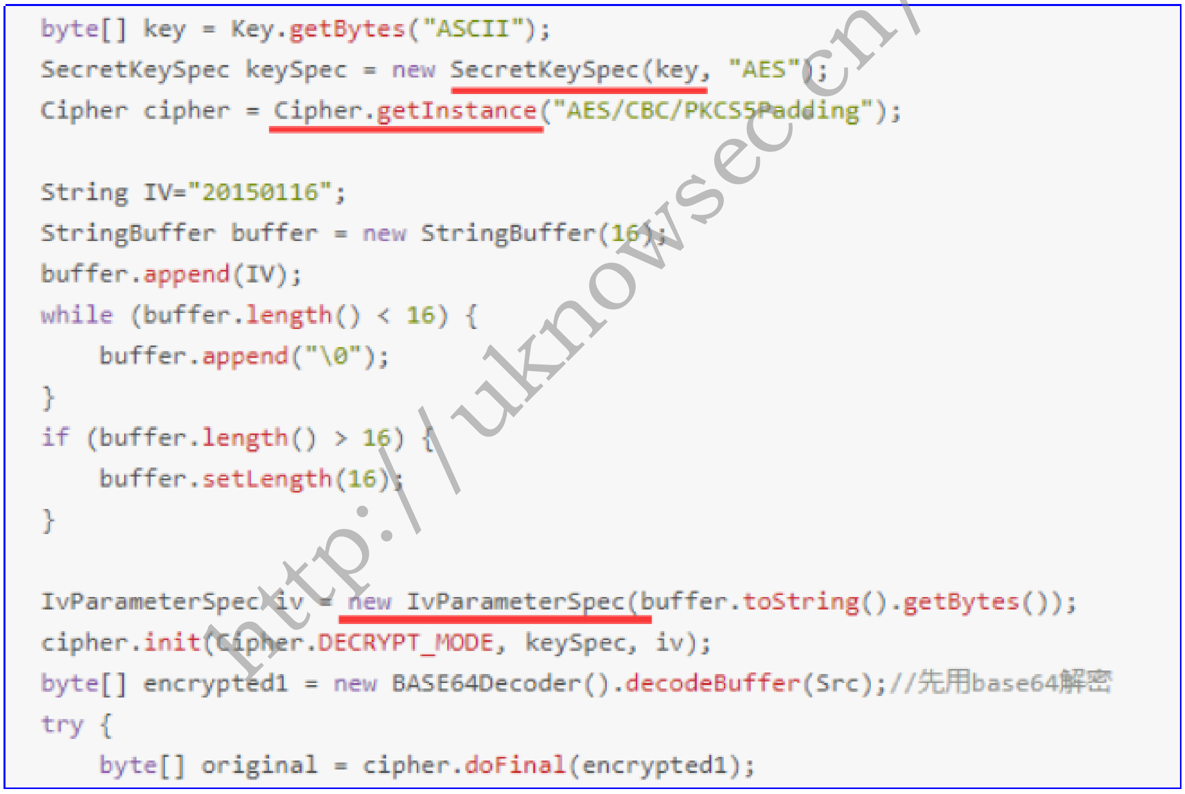
Trace based profiling works by tracing the entry and exit of every method. This captures the execution of all methods, no matter how small, and hence has a high overhead.

触发APP条件事件（比如Click或者刷新）生成.trace文件，这样我们就可以看到整个用堆栈过程。



```
Name                                                                                                   Incl Cpu Time
828 com/bianlidai123/bc/encrypt/Base64.isWhiteSpace (C)Z                                                      0.0%
618 com/bianlidai123/bc/encrypt/Base64.removeWhiteSpace ([C)I                                                 0.1%
362 com/bianlidai123/bc/encrypt/UtiEncrypt.MD5 (Ljava/lang/String;)Ljava/lang/String;                         0.2%
182 com/bianlidai123/bc/encrypt/UtiEncrypt.decryptAES (Ljava/lang/String;)Ljava/lang/String;                  0.8%
    Parents
        76 com/bianlidai123/bc/model/impl/LiCaiImpl.getLiCai (Lcom/bianlidai123/bc/model/params/BaseParams;  100.0%
    Children
        self                                                                                                  0.8%
        181 javax/crypto/Cipher.getInstance (Ljava/lang/String;)Ljavax/crypto/Cipher;                        49.5%
        337 javax/crypto/Cipher.init (ILjava/security/Key;Ljava/security/spec/AlgorithmParameterSpec;)V     25.8%
        457 com/bianlidai123/bc/encrypt/Base64.decode (Ljava/lang/String;)[B                                 15.6%
        539 javax/crypto/Cipher.doFinal ([B)[B                                                                5.4%
        769 java/lang/String.getBytes (Ljava/lang/String;)[B                                                  1.9%
        1492 java/lang/String.<init> ([BLjava/lang/String;)V                                                  0.8%
        1546 javax/crypto/spec/SecretKeySpec.<init> ([BLjava/lang/String;)V                                   0.3%
        javax/crypto/spec/IvParameterSpec.<init> ([B)V                                                        0.0%
231 com/bianlidai123/bc/encrypt/UtiEncrypt.encryptAES (Ljava/lang/String;)Ljava/lang/String;                  0.5%
1534 com/bianlidai123/bc/fragment/LicaiFragment$1.onTouch (Landroid/view/View;Landroid/view/MotionEvent;)Z    0.0%
```

对比AES的解密实现，只需要如下三个点得到，key、模式和iv值就行了。



```
byte[] key = Key.getBytes("ASCII");
SecretKeySpec keySpec = new SecretKeySpec(key, "AES");
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

String IV="20150116";
StringBuffer buffer = new StringBuffer(16);
buffer.append(IV);
while (buffer.length() < 16) {
    buffer.append("\0");
}
if (buffer.length() > 16) {
    buffer.setLength(16);
}

IvParameterSpec iv = new IvParameterSpec(buffer.toString().getBytes());
cipher.init(Cipher.DECRYPT_MODE, keySpec, iv);
byte[] encrypted1 = new BASE64Decoder().decodeBuffer(Src);//先用base64解密
try {
    byte[] original = cipher.doFinal(encrypted1);
```

key值和iv值是构造函数，根据hook构造函数的方法，要用到 `$init`，并且要 `return this.$init(arg1,arg2)` 调用原始的函数实现，同时 `IvParameterSpec` 方法的参数是一个比特数组的重载函数。

Java中比特数组表示为 `[B`，其他类型的数组如下表示。

```
[Z = boolean
[B = byte
[S = short
[I = int
[J = long
[F = float
[D = double
[C = char
[L = any non-primitives(Object)
```

在frida利用如下方式输出比特数组：

```
 for (i=0;i<arg1.length;i++)
    {
        b=(arg1[i]>>>0)&0xff;
        n=b.toString(16);
        hexstr += ("00" + n).slice(-2)+" ";
    }
```

完整jscode:

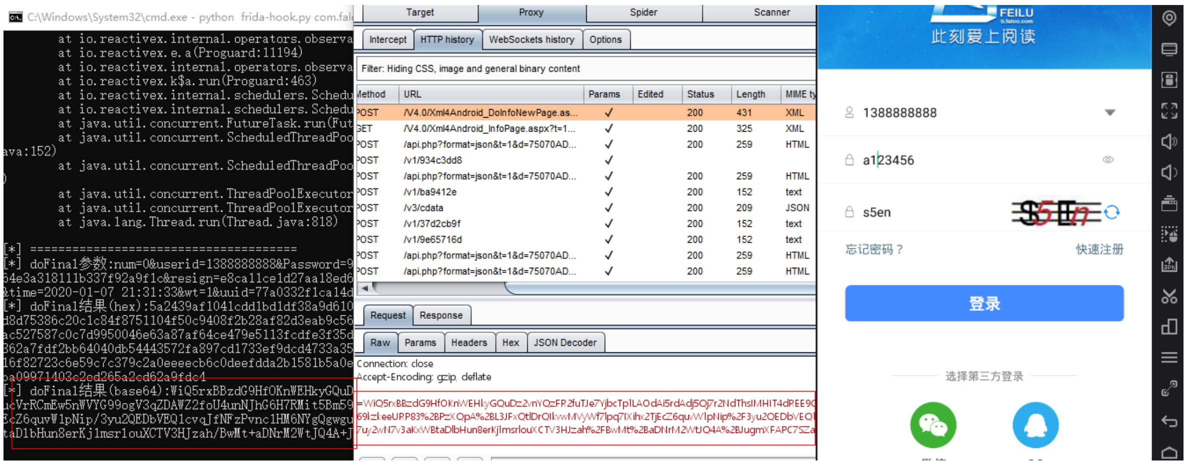
```
Java.perform(function x() {
    // Function to hook is defined here
    var UtiEncrypt = Java.use('com.bianlidai123.bc.encrypt.UtiEncrypt');

    // Whenever button is clicked
    UtiEncrypt.decryptAES.overload('java.lang.String').implementation = function
(arg1) {
        // Show a message to know that the function got called


       var sign=this.decryptAES(arg1);

       send("arg1:"+arg1);
       send("sign:"+sign);
       return sign;

    };

    var Cipher = Java.use('javax.crypto.Cipher');
    Cipher.getInstance.overload('java.lang.String').implementation = function
(arg1) {
         var sign2=this.getInstance(arg1);
         send("Instance:"+arg1);
         return sign2;

    };

    var SecretKeySpec = Java.use('javax.crypto.spec.SecretKeySpec');
    SecretKeySpec.$init.overload('[B', 'java.lang.String').implementation =
function (arg1,arg2) {
     hexstr="";
    for (i=0;i<arg1.length;i++)
    {
        b=(arg1[i]>>>0)&0xff;
        n=b.toString(16);
        hexstr += ("00" + n).slice(-2)+" ";
    }
        send("Key: " + hexstr);
      //send("init1:"+arg1+arg2);
        return this.$init(arg1,arg2);

    };

    var IvParameterSpec = Java.use('javax.crypto.spec.IvParameterSpec');
    IvParameterSpec.$init.overload('[B').implementation = function (arg1) {
    hexstr="";
    for (i=0;i<arg1.length;i++)
    {
        b=(arg1[i]>>>0)&0xff;
        n=b.toString(16);
        hexstr += ("00" + n).slice(-2)+" ";
    }
        send("Iv: " + hexstr);
      //send("init4:"+arg1);
        return this.$init(arg1);

    };

});
```

hook java原生算法同时打印调用堆栈

如下脚本可以hook java原生MD5、MAC、DES、AES、RSA加密算法并打印出调用堆栈，简单暴力。

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

jsCode = """
function showStacks() {
    Java.perform(function() {

send(Java.use("android.util.Log").getStackTraceString(Java.use("java.lang.Except
ion").$new()));
    });
}

function bytesToHex(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 bytesToBase64(e) {
    var base64EncodeChars =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/';
    var r, a, c, h, o, t;
    for (c = e.length, a = 0, r = ''; a < c;) {
        if (h = 255 & e[a++], a == c) {
            r += base64EncodeChars.charAt(h >> 2),
            r += base64EncodeChars.charAt((3 & h) << 4),
            r += '==';
            break
        }
        if (o = e[a++], a == c) {
            r += base64EncodeChars.charAt(h >> 2),
            r += base64EncodeChars.charAt((3 & h) << 4 | (240 & o) >> 4),
            r += base64EncodeChars.charAt((15 & o) << 2),
            r += '=';
            break
        }
        t = e[a++],
        r += base64EncodeChars.charAt(h >> 2),
        r += base64EncodeChars.charAt((3 & h) << 4 | (240 & o) >> 4),
        r += base64EncodeChars.charAt((15 & o) << 2 | (192 & t) >> 6),
        r += base64EncodeChars.charAt(63 & t)
    }
    return r
}
function bytesToString(arr) {
    if (typeof arr === 'string') {
        return arr;
    }
    var str = '',
    _arr = arr;
    for (var i = 0; i < _arr.length; i++) {
        var one = _arr[i].toString(2),
        v = one.match(/^1+?(?=0)/);
        if (v && one.length == 8) {
            var bytesLength = v[0].length;
            var store = _arr[i].toString(2).slice(7 - bytesLength);
            for (var st = 1; st < bytesLength; st++) {
                store += _arr[st + i].toString(2).slice(2);
            }
            str += String.fromCharCode(parseInt(store, 2));
            i += bytesLength - 1;
        } else {
            str += String.fromCharCode(_arr[i]);
        }
    }
    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(hex):" + bytesToHex(result));
        send("doFinal结果(base64):" + 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(hex):" + bytesToHex(result));
        send("doFinal结果(base):" + 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);
        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 (a) {
        showStacks();
        send("======================================");
        send("update:" + bytesToString(a))
        return this.update(a);
    }
    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(hex):" + bytesToHex(result));
        send("digest结果(base64):" + 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(hex):" + bytesToHex(result));
        send("digest结果(base64):" + 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(hex):" + 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(hex):" + bytesToHex(result));
        send("doFinal结果(base64):" + 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(hex):" + bytesToHex(result));
        send("doFinal结果(base64):" + 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.$init.overload('java.math.BigInteger','java.math.BigInteger').i
mplementation = function (a,b) {
        showStacks();
        var result = this.$init(a,b);
        send("======================================");
        //send("RSA密钥:" + bytesToBase64(a));
        send("RSA密钥N:" + a.toString(16));
        send("RSA密钥E:" + b.toString(16));
        return result;
    }
});
""";

fw = open(sys.argv[1],'w+',encoding='utf-8')

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

process = frida.get_remote_device().attach(sys.argv[1])
script= process.create_script(jsCode)
script.on("message", message)
script.load()
sys.stdin.read()
```

`python frida-hook.py com.faloo.BookReader4Android` 脚本加上包名，即可hook所有原生方法。



```
C:\Windows\System32\cmd.exe - python  frida-hook.py com.fal
        at io.reactivex.internal.operators.observa
        at io.reactivex.e.a(Proguard:11194)
        at io.reactivex.internal.operators.observa
        at io.reactivex.k$a.run(Proguard:463)
        at io.reactivex.internal.schedulers.Schedu
        at io.reactivex.internal.schedulers.Schedu
        at java.util.concurrent.FutureTask.run(Fut
        at java.util.concurrent.ScheduledThreadPoo
ava:152)
        at java.util.concurrent.ScheduledThreadPoo
)
        at java.util.concurrent.ThreadPoolExecutor
        at java.util.concurrent.ThreadPoolExecutor
        at java.lang.Thread.run(Thread.java:818)

[*] ======================================
[*] doFinal参数:num=0&userid=1388888888&Password=9
64e3a318111b337f92a9f1c&resign=e8ca11ce1d27aa18ed6
&time=2020-01-07 21:31:33&wt=1&uuid=77a0332f1ca14d
[*] doFinal结果(hex):5a2439af1041cdd1bd1df38a9d610
d8d75386c20c1c84f8751104f50c9408f2b28af82d3eab9c56
ac527587c0c7d9950046e63a87af64ce479e5113fcdfe3f35d
362a7fdf2bb64040db54443572fa897cd1733ef9dcd4733a35
16f82723c6e59c7c379c2a0eeeecb6c0deefdda2b1581b5a0e
ba09971403c2cd265a2cd62a9fdc4
[*] doFinal结果(base64):WiQ5rxBBzdG9HfOKnWEHkyGQuD
ucVrRCmEw5nWVYG99ogV3qZDAWZ2foU4unNJhG6H7RMit5Bm59
EcZ6quvW1pNip/3yu2QEDbVEQ1cvqJfNFzPvnc1HM6NYgQgwgu
taD1bHun8erKj1msr1ouXCTV3HJzah/BwMt+aDNrM2WtJQ4A+J
```

Target | Proxy | Spider | Scanner

Intercept | HTTP history | WebSockets history | Options

Filter: Hiding CSS, image and general binary content

Method	URL	Params	Edited	Status	Length	MIME ty
POST	/V4.0/Xml4Android_DoInfoNewPage.as...	✓		200	431	XML
GET	/V4.0/Xml4Android_InfoPage.aspx?t=1...	✓		200	325	XML
POST	/api.php?format=json&t=1&d=75070AD...	✓		200	259	HTML
POST	/v1/934c3dd8	✓				
POST	/api.php?format=json&t=1&d=75070AD...	✓		200	259	HTML
POST	/v1/ba9412e	✓		200	152	text
POST	/v3/cdata	✓		200	209	JSON
POST	/v1/37d2cb9f	✓		200	152	text
POST	/v1/9e65716d	✓		200	152	text
POST	/api.php?format=json&t=1&d=75070AD...	✓		200	259	HTML
POST	/api.php?format=json&t=1&d=75070AD...	✓		200	259	HTML

Request | Response

Raw | Params | Headers | Hex | JSON Decoder

```
Connection: close
Accept-Encoding: gzip, deflate

=WiQ5rxBBzdG9HfOKnWEHkyGQuDz2vnYOzFR2fuTJe7YjbcTplLAOdAi5rdAdj5Qj7r2NdThsIMHIT4dREE9C
69lzkeeURP83%2BPzXOpA%2BL3JFxQtlDrQIkwMVyWf7lpq7IXihx2TjEcZ6quWlpNip%2F3yu2QEDbVEQ
7uy2wN7v3aKxWBtaDlbHun8erKjlmsrlouXCTV3HJzah%2FBwMt%2BaDNrM2WtJQ4A%2BJugmXFAPC7SZa
```

如图某APP登录处，从 `burp` 里的流量记录，流量被加密了，在hook脚本中也打印了该段加密数据。

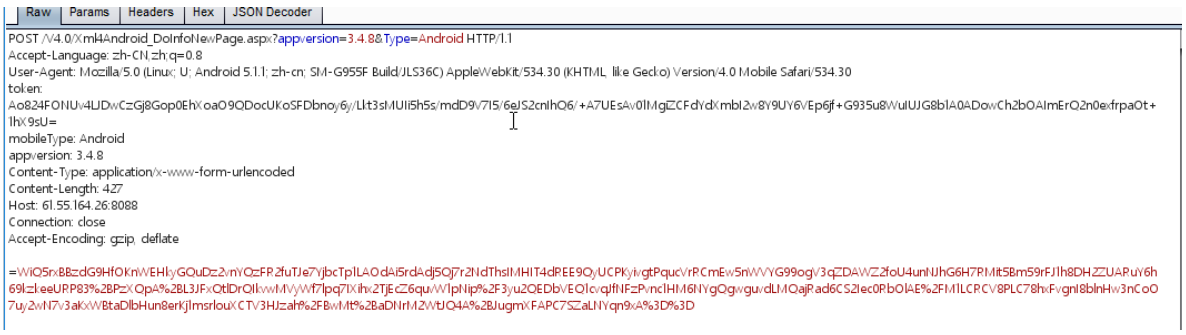
该脚本会在同目录下生成同包名的文件。

```
[*] ======================================
[*] 算法名：AES|Dec密钥:DD240BF148903189BF8DAE0C220C9591
[*] 算法名：AES|Hex密
钥:44443234304246313438393033313839424638444145304332323043393539931
[*] java.lang.Exception
    at javax.crypto.Cipher.getInstance(Native Method)
    at com.faloo.util.AES.encrypt(Proguard:211)
    at com.faloo.util.EncryptUtil._e18(Native Method)
    at com.faloo.util.EncryptUtil.getAESEncrypt(Proguard:150)
    at com.faloo.network.module.b.a(Proguard:49)
    at com.faloo.network.util.e.a(Proguard:174)
    at com.faloo.network.service.a.c.a(Proguard:141)
    at com.faloo.app.activity.LoginPageActivity$13.a(Proguard:941)
    at
io.reactivex.internal.operators.observable.ObservableCreate.b(Proguard:40)
    at io.reactivex.e.a(Proguard:11194)
    at
io.reactivex.internal.operators.observable.ObservableSubscribeOn$a.run(Proguard:
96)
    at io.reactivex.k$a.run(Proguard:463)
    at io.reactivex.internal.schedulers.ScheduledRunnable.run(Proguard:66)
    at io.reactivex.internal.schedulers.ScheduledRunnable.call(Proguard:57)
    at java.util.concurrent.FutureTask.run(FutureTask.java:237)
    at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$201(
ScheduledThreadPoolExecutor.java:152)
    at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(Schedul
edThreadPoolExecutor.java:265)
    at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1112)
    at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:587)
    at java.lang.Thread.run(Thread.java:818)

[*] ======================================
[*] 模式填充:AES/CBC/PKCS5Padding
[*] java.lang.Exception
    at javax.crypto.spec.IvParameterSpec.<init>(Native Method)
    at com.faloo.util.AES.encrypt(Proguard:212)
    at com.faloo.util.EncryptUtil._e18(Native Method)
    at com.faloo.util.EncryptUtil.getAESEncrypt(Proguard:150)
    at com.faloo.network.module.b.a(Proguard:49)
    at com.faloo.network.util.e.a(Proguard:174)
    at com.faloo.network.service.a.c.a(Proguard:141)
    at com.faloo.app.activity.LoginPageActivity$13.a(Proguard:941)
    at
io.reactivex.internal.operators.observable.ObservableCreate.b(Proguard:40)
    at io.reactivex.e.a(Proguard:11194)
    at
io.reactivex.internal.operators.observable.ObservableSubscribeOn$a.run(Proguard:
96)
    at io.reactivex.k$a.run(Proguard:463)
    at io.reactivex.internal.schedulers.ScheduledRunnable.run(Proguard:66)
    at io.reactivex.internal.schedulers.ScheduledRunnable.call(Proguard:57)
    at java.util.concurrent.FutureTask.run(FutureTask.java:237)
    at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$201(
ScheduledThreadPoolExecutor.java:152)
    at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(Schedul
edThreadPoolExecutor.java:265)
    at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1112)
    at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:587)
    at java.lang.Thread.run(Thread.java:818)

[*] ======================================
[*] iv向量:
[*] iv向量(hex):00000000000000000000000000000000
[*] java.lang.Exception
    at javax.crypto.Cipher.doFinal(Native Method)
    at com.faloo.util.AES.encrypt(Proguard:213)
    at com.faloo.util.EncryptUtil._e18(Native Method)
    at com.faloo.util.EncryptUtil.getAESEncrypt(Proguard:150)
    at com.faloo.network.module.b.a(Proguard:49)
    at com.faloo.network.util.e.a(Proguard:174)
    at com.faloo.network.service.a.c.a(Proguard:141)
    at com.faloo.app.activity.LoginPageActivity$13.a(Proguard:941)
    at
io.reactivex.internal.operators.observable.ObservableCreate.b(Proguard:40)
    at io.reactivex.e.a(Proguard:11194)
    at
io.reactivex.internal.operators.observable.ObservableSubscribeOn$a.run(Proguard:
96)
    at io.reactivex.k$a.run(Proguard:463)
    at io.reactivex.internal.schedulers.ScheduledRunnable.run(Proguard:66)
    at io.reactivex.internal.schedulers.ScheduledRunnable.call(Proguard:57)
    at java.util.concurrent.FutureTask.run(FutureTask.java:237)
    at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$201(
ScheduledThreadPoolExecutor.java:152)
    at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(Schedul
edThreadPoolExecutor.java:265)
    at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1112)
```

```
    at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:587)
    at java.lang.Thread.run(Thread.java:818)

[*] =======================================
[*] doFinal参
数:num=0&userid=1388888888&Password=9c9c5f3bff756cf5395265b6c5f0d8f0&tid=2&ts=15
78400865589&nonce=fb4f89a9464e3a318111b337f92a9f1c&resign=e8ca11ce1d27aa18ed6840
617b8cbcf7&pwdcode=YTEyMzQ1Ng==
&time=2020-01-07
21:31:33&wt=1&uuid=77a0332f1ca14daf9337f941bcc86e70&appversion=3.4.8&Type=Androi
d&xp=0
[*] doFinal结果
(hex):5a2439af1041cdd1bd1df38a9d6107932190b83cf6be7610cc54767ee4c97bb6236dc4e9d4
b00e7408b9add01d8f9423eebd8d75386c20c1c84f8751104f50c9408f2b28af82d3eab9c56b4429
84c399d65581bdf68815dea6430166767e8538ba7349846e87ed1322b79066e7dac527587c0c7d99
50046e63a87af64ce479e5113fcdfe3f35d0a40f8bdc917142d943ad0224bf0315c967fb969abb21
78a1c764e311c67aaaebd6d69362a7fdf2bb64040db54443572fa897cd1733ef9dcd4733a3588108
3082ebdd2cc41a8d169de824b621e73445b3a5004fccd4b091095f0f2c2efc87116f82723c6e59c7
c379c2a0eeeecb6c0deefdda2b1581b5a0e56c7ba7f1eaca8f59acae5a2e5c24d5dc72736a1fc1c0
cb7e68336b3365ad250e00f89ba09971403c2ed265a2cd62a9fdc4
[*] doFinal结果
(base64):WiQ5rxBBzdG9HfOKnWEHkyGQuDz2vnYQzFR2fuTJe7YjbcTp1LAOdAi5rdAdj5Qj7r2NdTh
sIMHIT4dREE9QyUCPKyivgtPqucVrRCmEw5nWVYG99ogV3qZDAWZ2foU4unNJhG6H7RMit5Bm59rFJ1h
8DH2ZUARuY6h69kzkeeURP83+PzXQpA+L3JFxQtlDrQIkvwMVyWf7lpq7IXihx2TjEcZ6quvW1pNip/3
yu2QEDbVEQ1cvqJfNFzPvnc1HM6NYgQgwguvdLMQajRad6CS2Iec0RbOlAE/M1LCRCV8PLC78hxFvgnI
8blnHw3nCoO7uy2wN7v3aKxWBtaDlbHun8erKj1msrlouXCTV3HJzah/BwMt+aDNrM2WtJQ4A+JugmXF
APC7SZaLNYqn9xA==
```

如下数据是最终加密的结果，即发给服务器的数据。

```
WiQ5rxBBzdG9HfOKnWEHkyGQuDz2vnYQzFR2fuTJe7YjbcTp1LAOdAi5rdAdj5Qj7r2NdThsIMHIT4dR
EE9QyUCPKyivgtPqucVrRCmEw5nWVYG99ogV3qZDAWZ2foU4unNJhG6H7RMit5Bm59rFJ1h8DH2ZUARu
Y6h69kzkeeURP83+PzXQpA+L3JFxQtlDrQIkvwMVyWf7lpq7IXihx2TjEcZ6quvW1pNip/3yu2QEDbVE
Q1cvqJfNFzPvnc1HM6NYgQgwguvdLMQajRad6CS2Iec0RbOlAE/M1LCRCV8PLC78hxFvgnI8blnHw3nC
oO7uy2wN7v3aKxWBtaDlbHun8erKj1msrlouXCTV3HJzah/BwMt+aDNrM2WtJQ4A+JugmXFAPC7SZaLN
Yqn9xA==
```



```
POST /V4.0/Xml4Android_DoInfoNewPage.aspx?appversion=3.4.8&Type=Android HTTP/1.1
Accept-Language: zh-CN,zh;q=0.8
User-Agent: Mozilla/5.0 (Linux; U; Android 5.1.1; zh-cn; SM-G955F Build/JLS36C) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Mobile Safari/534.30
token:
Ao824FONUv4UDwCzGj8Gop0EhXoaO9QDocUKoSFDbnoy6y/Lkt3sMUIi5h5s/mdD9V7I5/6eJS2cnIhQ6/+A7UEsAv0lMgZCFdYdXmbI2w8Y9UY6VEp6jf+G935u8WuIUJG8bIA0ADowCh2bOAImErQ2n0exfrpaOt+
1hX9sU=
mobileType: Android
appversion: 3.4.8
Content-Type: application/x-www-form-urlencoded
Content-Length: 427
Host: 61.55.164.26:8088
Connection: close
Accept-Encoding: gzip, deflate

=WiQ5rxBBzdG9HfOKnWEHkyGQuDz2vnYQzFR2fuTJe7YjbcTp1LAOdAi5rdAdj5Qj7r2NdThsIMHIT4dREE9QyUCPKyivgtPqucVrRCmEw5nWVYG99ogV3qZDAWZ2foU4unNJhG6H7RMit5Bm59rFJ1h8DH2ZUARuY6h
69kzkeeURP83%2BPzXQpA%2BL3JFxQtlDrQIkvwMVyWf7lpq7IXihx2TjEcZ6quvW1pNip%2F3yu2QEDbVEQ1cvqJfNFzPvnc1HM6NYgQgwguvdLMQajRad6CS2Iec0RbOlAE%2FM1LCRCV8PLC78hxFvgnI8blnHw3nCoO
7uy2wN7v3aKxWBtaDlbHun8erKj1msrlouXCTV3HJzah%2FBwMt%2BaDNrM2WtJQ4A%2BJugmXFAPC7SZaLNYqn9xA%3D%3D
```

这里hook到了加密算法和密钥

```
[*] 算法名：AES|Dec密钥:DD240BF148903189BF8DAE0C220C9591
[*] 算法名：AES|Hex密
钥:44443234304246313438393033313839424638444145304332323043393539353931
```

偏移模式

```
模式填充:AES/CBC/PKCS5Padding
```

iv向量

```
iv向量(hex):00000000000000000000000000000000
```

Password加密过程分析

输出中可以看到其加密之前的数据为

```
num=0&userid=1388888888&Password=9c9c5f3bff756cf5395265b6c5f0d8f0&tid=2&ts=15784
00865589&nonce=fb4f89a9464e3a318111b337f92a9f1c&resign=e8ca11ce1d27aa18ed6840617
b8cbcf7&pwdcode=YTEyMzQ1Ng==
&time=2020-01-07
21:31:33&wt=1&uuid=77a0332f1ca14daf9337f941bcc86e70&appversion=3.4.8&Type=Androi
d&xp=0
```

数据中userid为用户名，Password为密码，这里的密码是密文 `9c9c5f3bff756cf5395265b6c5f0d8f0`，这里并不知道是什么密文。此密文并非输入的密码的md5值。在输出中我们可以直接搜索这一段密文。

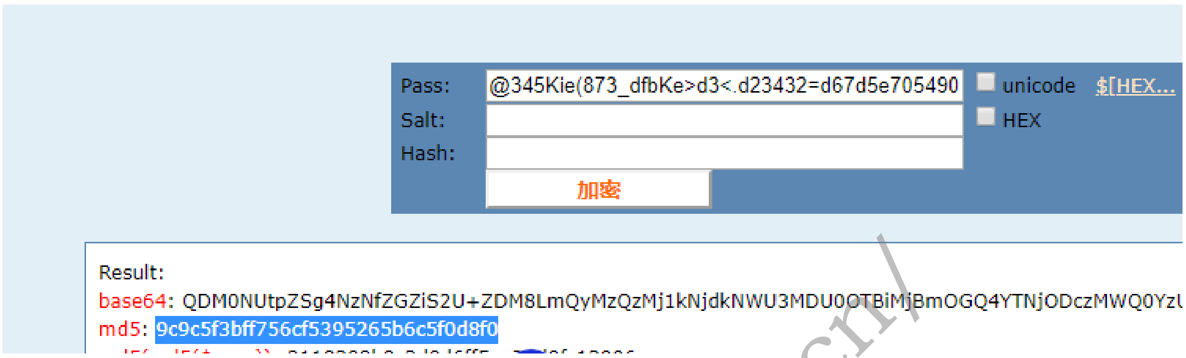
搜索到`9c9c5f3bff756cf5395265b6c5f0d8f0`为如下输出结果。

```
[*] ========================================
[*] update:@345Kie(873_dfbKe>d3<.d23432=d67d5e705490b20f8d8a3c8731d4c535
[*] java.lang.Exception
    at java.security.MessageDigest.digest(Native Method)
    at java.security.MessageDigest.digest(MessageDigest.java:278)
    at java.security.MessageDigest.digest(Native Method)
    at com.faloo.network.util.MD5.MD5(Proguard:22)
    at com.faloo.util.EncryptUtil._e16(Native Method)
    at com.faloo.util.EncryptUtil.EncryptPwd(Proguard:23)
    at com.faloo.network.service.a.c.a(Proguard:119)
    at com.faloo.app.activity.LoginPageActivity$13.a(Proguard:941)
    at
io.reactivex.internal.operators.observable.ObservableCreate.b(Proguard:40)
    at io.reactivex.e.a(Proguard:11194)
    at
io.reactivex.internal.operators.observable.ObservableSubscribeOn$a.run(Proguard:
96)
    at io.reactivex.k$a.run(Proguard:463)
    at io.reactivex.internal.schedulers.ScheduledRunnable.run(Proguard:66)
    at io.reactivex.internal.schedulers.ScheduledRunnable.call(Proguard:57)
    at java.util.concurrent.FutureTask.run(FutureTask.java:237)
    at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$201(
ScheduledThreadPoolExecutor.java:152)
    at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(Schedul
edThreadPoolExecutor.java:265)
    at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1112)
    at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:587)
    at java.lang.Thread.run(Thread.java:818)
```

```
[*] ======================================
[*] digest结果(hex):9c9c5f3bff756cf5395265b6c5f0d8f0
[*] digest结果(base64):nJxfO/91bPU5UmW2xfDY8A==
[*] digest结果(hex):9c9c5f3bff756cf5395265b6c5f0d8f0
[*] digest结果(base64):nJxfO/91bPU5UmW2xfDY8A==
```

在这个过程中输入的为`@345Kie(873_dfbKe>d3<.d23432=d67d5e705490b20f8d8a3c8731d4c535`输出的`9c9c5f3bff756cf5395265b6c5f0d8f0`

通过md5加密对比如下图，发现此过程为md5加密过程。



Pass: @345Kie(873_dfbKe>d3<.d23432=d67d5e705490

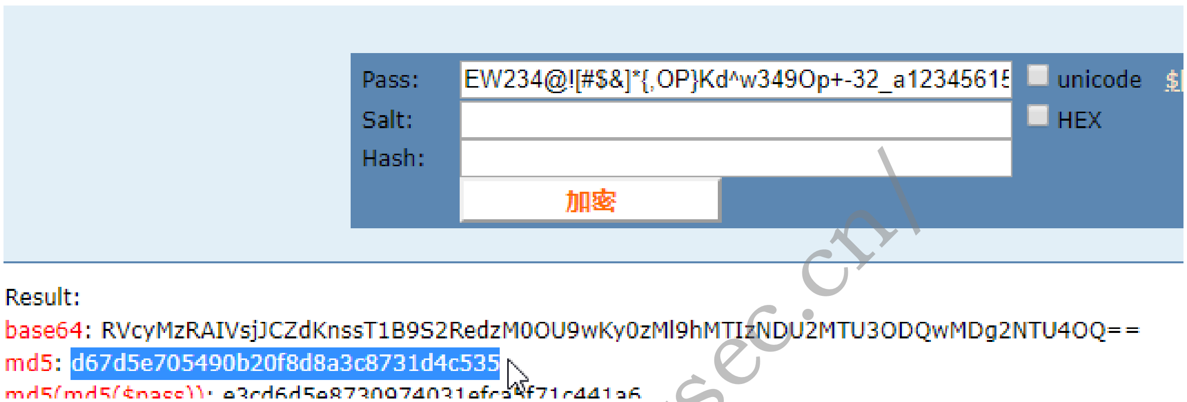
Result:
base64: QDM0NUtpZSg4NzNfZGZiS2U+ZDM8LmQyMzQzMj1kNjdkNWU3MDU0OTBiMjBmOGQ4YTNjODczMWQ0YzU
md5: 9c9c5f3bff756cf5395265b6c5f0d8f0

`@345Kie(873_dfbKe>d3<.d23432=d67d5e705490b20f8d8a3c8731d4c535`中，前一段`@345Kie(873_dfbKe>d3<.d23432=`并没有搜索到，可能为固定值拼接，通过多此测试发现确实为固定值拼接。故直接搜索后32位`d67d5e705490b20f8d8a3c8731d4c535`跟到如下输出过程。

```
[*] ======================================
[*] update:EW234@![#$&]*{,OP}Kd^w349Op+-32_a1234561578400865589
[*] java.lang.Exception
    at java.security.MessageDigest.digest(Native Method)
    at java.security.MessageDigest.digest(MessageDigest.java:278)
    at java.security.MessageDigest.digest(Native Method)
    at com.faloo.network.util.MD5.MD5(Proguard:22)
    at com.faloo.util.EncryptUtil._e16(Native Method)
    at com.faloo.util.EncryptUtil.EncryptPwd(Proguard:23)
    at com.faloo.network.service.a.c.a(Proguard:119)
    at com.faloo.app.activity.LoginPageActivity$13.a(Proguard:941)
    at
io.reactivex.internal.operators.observable.ObservableCreate.b(Proguard:40)
    at io.reactivex.e.a(Proguard:11194)
    at
io.reactivex.internal.operators.observable.ObservableSubscribeOn$a.run(Proguard:
96)
    at io.reactivex.k$a.run(Proguard:463)
    at io.reactivex.internal.schedulers.ScheduledRunnable.run(Proguard:66)
    at io.reactivex.internal.schedulers.ScheduledRunnable.call(Proguard:57)
    at java.util.concurrent.FutureTask.run(FutureTask.java:237)
    at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$201(
ScheduledThreadPoolExecutor.java:152)
    at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(Schedul
edThreadPoolExecutor.java:265)
    at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1112)
```

```
    at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:587)
    at java.lang.Thread.run(Thread.java:818)

[*] ======================================
[*] digest结果(hex):d67d5e705490b20f8d8a3c8731d4c535
[*] digest结果(base64):1n1ecFSQsg+NijyHMdTFNQ==
[*] digest结果(hex):d67d5e705490b20f8d8a3c8731d4c535
[*] digest结果(base64):1n1ecFSQsg+NijyHMdTFNQ==
```

如上输入为`EW234@![#$&]*{,OP}Kd^w349Op+-32_a1234561578400865589`输出为`d67d5e705490b20f8d8a3c8731d4c535`。这里`EW234@![#$&]*{,OP}Kd^w349Op+-32_`同样为固定值，`a123456`为我们输入的密码，`1578400865589`为时间戳。此过程为MD5加密



Result:
base64: RVcyMzRAIVsjJCZdKnssT1B9S2RedzM0OU9wKy0zMl9hMTIzNDU2MTU3ODQwMDg2NTU4OQ==
md5: d67d5e705490b20f8d8a3c8731d4c535
md5(md5($pass)): e3cd6d5e8730974031efca3f71c441a6

至此我们可以获取到

```
num=0&userid=1388888888&Password=9c9c5f3bff756cf5395265b6c5f0d8f0&tid=2&ts=15784
00865589&nonce=fb4f89a9464e3a318111b337f92a9f1c&resign=e8ca11ce1d27aa18ed6840617
b8cbcf7&pwdcode=YTEyMzQ1Ng==
&time=2020-01-07
21:31:33&wt=1&uuid=77a0332f1ca14daf9337f941bcc86e70&appversion=3.4.8&Type=Androi
d&xp=0
```

aes加密前的数据中password的加密过程：

大致流程为：

1. string1 = `EW234@![#$&]*{,OP}Kd^w349Op+-32_` + `密码` +时间戳，对string1进行md5加密
2. 对md5(string1)前拼接@345Kie(873_dfbKe>d3<.d23432=，再进行一次MD5加密得到最后的password加密值。

once加密过程分析

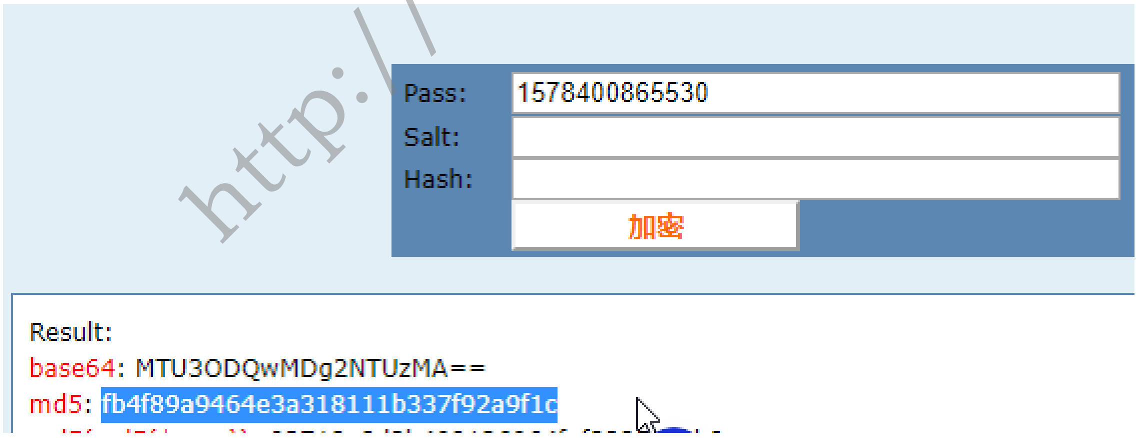
```
num=0&userid=1388888888&Password=9c9c5f3bff756cf5395265b6c5f0d8f0&tid=2&ts=15784
00865589&nonce=fb4f89a9464e3a318111b337f92a9f1c&resign=e8ca11ce1d27aa18ed6840617
b8cbcf7&pwdcode=YTEyMzQ1Ng==
&time=2020-01-07
21:31:33&wt=1&uuid=77a0332f1ca14daf9337f941bcc86e70&appversion=3.4.8&Type=Androi
d&xp=0
```

如上数据中的ts为时间戳即加密用到的时间戳，客户端将此时间戳发给服务端，因为在密码加密中用到了这个时间戳，所以服务器进行密码对比的时候也要用到这个时间戳，所以这个时间戳是一一对应的。

once值`fb4f89a9464e3a318111b337f92a9f1c`同样的方法去搜索

```
[*] update:1578400865530
[*] java.lang.Exception
    at java.security.MessageDigest.digest(Native Method)
    at org.faloo.app.pay.a.a(Proguard:40)
    at com.faloo.app.activity.LoginPageActivity.e(Proguard:483)
    at com.faloo.app.activity.LoginPageActivity.b(Proguard:111)
    at com.faloo.app.activity.LoginPageActivity$3.a(Proguard:531)
    at com.faloo.app.activity.LoginPageActivity$3.onNext(Proguard:517)
    at
io.reactivex.internal.operators.observable.ObservableObserveOn$ObserveOnObserver
.drainNormal(Proguard:200)
    at
io.reactivex.internal.operators.observable.ObservableObserveOn$ObserveOnObserver
.run(Proguard:252)
    at io.reactivex.android.b.b$b.run(Proguard:109)
    at android.os.Handler.handleCallback(Handler.java:739)
    at android.os.Handler.dispatchMessage(Handler.java:95)
    at android.os.Looper.loop(Looper.java:135)
    at android.app.ActivityThread.main(ActivityThread.java:5293)
    at java.lang.reflect.Method.invoke(Native Method)
    at java.lang.reflect.Method.invoke(Method.java:372)
    at
com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:903)
    at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:698)

[*] ======================================
[*] digest结果(hex):fb4f89a9464e3a318111b337f92a9f1c
[*] digest结果(base64):+0+JqUZOOjGBEbM3+SqfHA==
```

得到如上过程，输入`1578400865530`输出once值。同样为md5加密。



Pass: 1578400865530
Salt:
Hash:

Result:
base64: MTU3ODQwMDg2NTUzMA==
md5: fb4f89a9464e3a318111b337f92a9f1c

此时间戳并非在加密过程中用到，可能在后端并未做校验。

resign加密过程分析

```
num=0&userid=1388888888&Password=9c9c5f3bff756cf5395265b6c5f0d8f0&tid=2&ts=15784
00865589&nonce=fb4f89a9464e3a318111b337f92a9f1c&resign=e8ca11ce1d27aa18ed6840617
b8cbcf7&pwdcode=YTEyMzQ1Ng==
&time=2020-01-07
21:31:33&wt=1&uuid=77a0332f1ca14daf9337f941bcc86e70&appversion=3.4.8&Type=Androi
d&xp=0
```

这个数据中还有一个`e8ca11ce1d27aa18ed6840617b8cbcf7`为resign签名值。

```
[*] ========================================
[*] update:a1f*(DV<>ME29p08adsfKQ@N>FEP*(F&G)&B)R@PVDbvnTPFPSDFQ>QM@o9i8t5P_)
(SGB?9c9c5f3bff756cf5395265b6c5f0d8f0fb4f89a9464e3a318111b337f92a9f1c
[*] java.lang.Exception
    at java.security.MessageDigest.digest(Native Method)
    at java.security.MessageDigest.digest(MessageDigest.java:278)
    at java.security.MessageDigest.digest(Native Method)
    at com.faloo.network.util.MD5.MD5(Proguard:22)
    at com.faloo.util.EncryptUtil._e14(Native Method)
    at com.faloo.util.EncryptUtil.EncryptRePwd(Proguard:46)
    at com.faloo.network.service.a.c.a(Proguard:120)
    at com.faloo.app.activity.LoginPageActivity$13.a(Proguard:941)
    at
io.reactivex.internal.operators.observable.ObservableCreate.b(Proguard:40)
    at io.reactivex.e.a(Proguard:11194)
    at
io.reactivex.internal.operators.observable.ObservableSubscribeOn$a.run(Proguard:
96)
    at io.reactivex.k$a.run(Proguard:463)
    at io.reactivex.internal.schedulers.ScheduledRunnable.run(Proguard:66)
    at io.reactivex.internal.schedulers.ScheduledRunnable.call(Proguard:57)
    at java.util.concurrent.FutureTask.run(FutureTask.java:237)
    at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$201(
ScheduledThreadPoolExecutor.java:152)
    at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(Schedul
edThreadPoolExecutor.java:265)
    at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1112)
    at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:587)
    at java.lang.Thread.run(Thread.java:818)

[*] ========================================
[*] digest结果(hex):e8ca11ce1d27aa18ed6840617b8cbcf7
[*] digest结果(base64):6MoRzh0nqhjtaEBhe4y89w==
[*] digest结果(hex):e8ca11ce1d27aa18ed6840617b8cbcf7
[*] digest结果(base64):6MoRzh0nqhjtaEBhe4y89w==
```

该签名与`a1f*(DV<>ME29p08adsfKQ@N>FEP*(F&G)&B)R@PVDbvnTPFPSDFQ>QM@o9i8t5P_)(SGB?9c9c5f3bff756cf5395265b6c5f0d8f0fb4f89a9464e3a318111b337f92a9f1c`有关。此过程也为MD5加密。其中一段为9c9c5f3bff756cf5395265b6c5f0d8f0为password值。`fb4f89a9464e3a318111b337f92a9f1c`为once值。前面的一段`a1f*(DV<>ME29p08adsfKQ@N>FEP*(F&G)&B)R@PVDbvnTPFPSDFQ>QM@o9i8t5P_)(SGB?`为固定值。

所以签名值resign为固定值+password+once然后进行md5加密。

而最后的值`uuid=77a0332f1ca14daf9337f941bcc86e70`可能为用户名标识或者手机设备识别码。

```
num=0&userid=1388888888&Password=9c9c5f3bff756cf5395265b6c5f0d8f0&tid=2&ts=15784
00865589&nonce=fb4f89a9464e3a318111b337f92a9f1c&resign=e8ca11ce1d27aa18ed6840617
b8cbcf7&pwdcode=YTEyMzQ1Ng==
&time=2020-01-07
21:31:33&wt=1&uuid=77a0332f1ca14daf9337f941bcc86e70&appversion=3.4.8&Type=Androi
d&xp=0
```

pwdcode值为`YTEyMzQ1Ng==`为密码的base64。

自此如上加密数据所有部分都通过hook输出内容进行了分析。

RSA加密过程分析

在hook输出内容中还有，RSA加密过程。

```
[*] ======================================
[*] RSA密
钥:MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCDiI/dCs429FC75NYnF82omzzAweej2VdpdaKP3
DL0D/s3Hg7cnVTGBh6yRrKYI9cvBKorxdrCEaW0SXZYBH5nvmCg8qyzO8jBj08ISiukEQuqG2oS0L2sb
cQl0MV7rExsyO0vlPpND7klBWikAIO1UfZW1ab/EWit1XkaXCr6nQIDAQAB
[*] java.lang.Exception
    at javax.crypto.Cipher.getInstance(Native Method)
    at com.faloo.util.SignUtils.encrypt(Proguard:104)
    at com.faloo.util.EncryptUtil._e8(Native Method)
    at com.faloo.util.EncryptUtil.getRSAEncrypt(Proguard:166)
    at com.faloo.network.module.b.d(Proguard:39)
    at com.faloo.network.util.e.a(Proguard:101)
    at com.faloo.network.service.a.c.a(Proguard:141)
    at com.faloo.app.activity.LoginPageActivity$13.a(Proguard:941)
    at
io.reactivex.internal.operators.observable.ObservableCreate.b(Proguard:40)
    at io.reactivex.e.a(Proguard:11194)
    at
io.reactivex.internal.operators.observable.ObservableSubscribeOn$a.run(Proguard:
96)
    at io.reactivex.k$a.run(Proguard:463)
    at io.reactivex.internal.schedulers.ScheduledRunnable.run(Proguard:66)
    at io.reactivex.internal.schedulers.ScheduledRunnable.call(Proguard:57)
    at java.util.concurrent.FutureTask.run(FutureTask.java:237)
    at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$201(
ScheduledThreadPoolExecutor.java:152)
    at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(Schedul
edThreadPoolExecutor.java:265)
    at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1112)
    at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:587)
    at java.lang.Thread.run(Thread.java:818)

[*] ======================================
[*] 模式填充:RSA/ECB/PKCS1Padding
[*] java.lang.Exception
    at javax.crypto.Cipher.doFinal(Native Method)
    at com.faloo.util.SignUtils.encrypt(Proguard:106)
    at com.faloo.util.EncryptUtil._e8(Native Method)
    at com.faloo.util.EncryptUtil.getRSAEncrypt(Proguard:166)
    at com.faloo.network.module.b.d(Proguard:39)
    at com.faloo.network.util.e.a(Proguard:101)
    at com.faloo.network.service.a.c.a(Proguard:141)
    at com.faloo.app.activity.LoginPageActivity$13.a(Proguard:941)
    at
io.reactivex.internal.operators.observable.ObservableCreate.b(Proguard:40)
    at io.reactivex.e.a(Proguard:11194)
    at
io.reactivex.internal.operators.observable.ObservableSubscribeOn$a.run(Proguard:
96)
    at io.reactivex.k$a.run(Proguard:463)
    at io.reactivex.internal.schedulers.ScheduledRunnable.run(Proguard:66)
    at io.reactivex.internal.schedulers.ScheduledRunnable.call(Proguard:57)
    at java.util.concurrent.FutureTask.run(FutureTask.java:237)
    at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$201(
ScheduledThreadPoolExecutor.java:152)
    at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(Schedul
edThreadPoolExecutor.java:265)
    at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1112)
    at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:587)
    at java.lang.Thread.run(Thread.java:818)

[*] ======================================
[*] doFinal参数:DD240BF148903189BF8DAE0C220C9591
[*] doFinal结果
(hex):028f36e0538d52fe0b243c02cc68fc1a8a741215e868ef500e87142a84850db9e8cbacbf2e
4b77b0c5088b9879b3f99d0fd57b239ffa7894b672722143affe03b504b00bf4d4c82264215d61d5
e66c8db0f18f5463a544a7a8dff86f77e6ef16b885091bc6f5034003a300a1d9b38022612b4369f4
7b17eba5a3adfb5857f6c5
[*] doFinal结果
(base64):Ao824FONUv4LJDwCzGj8Gop0EhXoaO9QDocUKoSFDbnoy6y/Lkt3sMUIi5h5s/mdD9V7I5/
6eJS2cnIhQ6/+A7UEsAv01MgiZCFdYdXmbI2w8Y9UY6VEp6jf+G935u8WuIUJG8b1A0ADowCh2bOAImE
rQ2n0exfrpaOt+1hX9sU=
```

如上为RSA加密过程公钥为 `MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCDiI/dCs429FC75NYnF82omzzAweej2VdpdaKP3DL0D/s3Hg7cnVTGBh6yRrKYI9cvBKorxdrCEaW0SXZYBH5nvmCg8qyzO8jBj08ISiukEQuqG2oS0L2sbcQl0MV7rExsyO0vlPpND7klBWikAIO1UfZW1ab/EWit1XkaXCr6nQIDAQAB`

模式填充: `RSA/ECB/PKCS1Padding`

加密数据为： `DD240BF148903189BF8DAE0C220C9591`

加密结果为： `Ao824FONUv4LJDwCzGj8Gop0EhXoaO9QDocUKoSFDbnoy6y/Lkt3sMUIi5h5s/mdD9V7I5/6eJS2cnIhQ6/+A7UEsAv01MgiZCFdYdXmbI2w8Y9UY6VEp6jf+G935u8WuIUJG8b1A0ADowCh2bOAImErQ2n0exfrpaOt+1hX9sU=`

这个过程是为了加密AES密钥，因为RSA加密的加密数据长度是有限的。

> RSA一次能加密的明文长度与密钥长度成正比：

> len_in_byte(raw_data) = len_in_bit(key)/8 -11，如 1024bit 的密钥，一次能加密的内容长度为 1024/8 -11 = 117 byte。
>
> 所以非对称加密一般都用于加密对称加密算法的密钥，而不是直接加密内容

因为AES是对称加密

> 对称加密就是指，**加密和解密使用同一个密钥的加密方式。**
>
> 发送方使用密钥将明文数据加密成密文，然后发送出去，接收方收到密文后，使用同一个密钥将密文解密成明文读取。
>
> 优点：**加密计算量小、速度块，适合对大量数据进行加密的场景。**

所以通常流量数据为AES，但是AES存在安全问题就是，AES的密钥要是被截获了就可以任意解密数据。

所以目前APP中的加密套路就是:

客户端：流量数据通过AES加密发送给服务端，同时AES密钥随机生成通过RSA公钥加密发服务端

服务端：服务端接收到RSA加密后的AES密钥通过RSA私钥进行解密得到AES密钥，然后通过AES密钥解密流量数据进行验证。

frida hook 工具篇

Brida

Brida简介

Brida是BurpSuite的一个插件，它可以将Burp和Frida结合起来使用，可以在 BurpSuite中直接调用目标应用程序中的加/解密函数，这样就可以根据你的需求修改移动端APP与服务器的通信流量。而不用去逆向它，从而节省测试人员的精力。

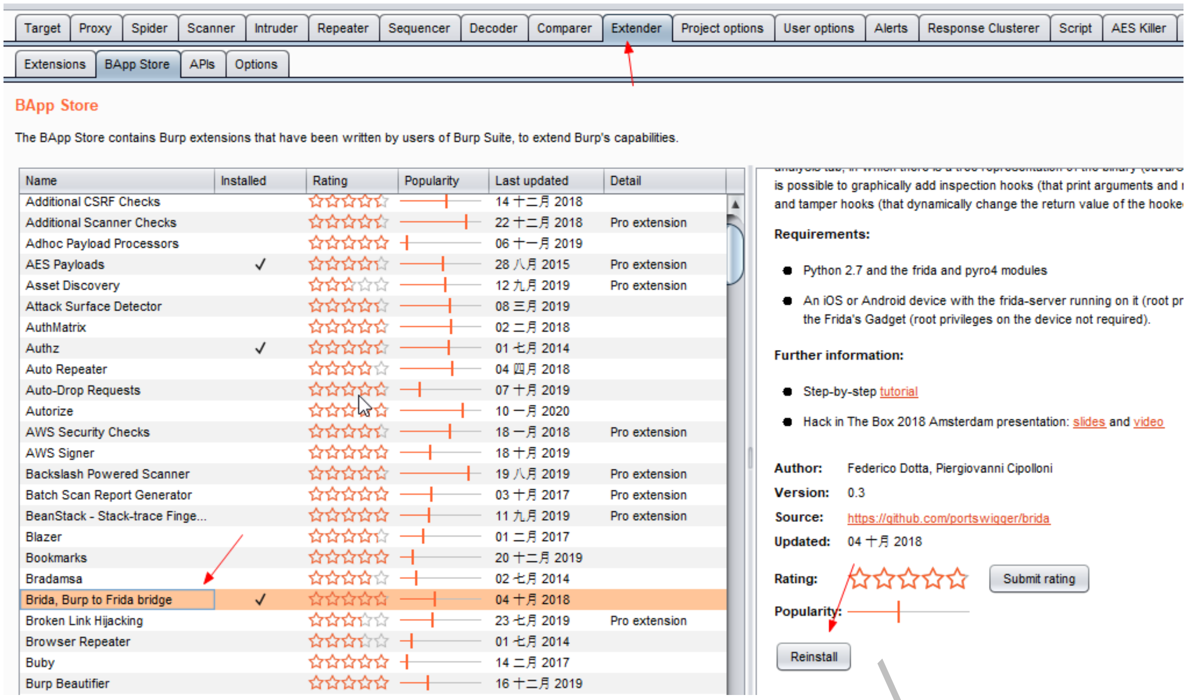
https://github.com/federicodotta/Brida

Brida安装

Brida目前只支持python 2.7。

```
pip install frida
pip install frida-tools
pip install pyro4
```

Brida可以直接从BurpSuite中安装



Target | Proxy | Spider | Scanner | Intruder | Repeater | Sequencer | Decoder | Comparer | Extender | Project options | User options | Alerts | Response Clusterer | Script | AES Killer

Extensions | BApp Store | APIs | Options

BApp Store

The BApp Store contains Burp extensions that have been written by users of Burp Suite, to extend Burp's capabilities.

Name	Installed	Rating	Popularity	Last updated	Detail
Additional CSRF Checks				14 十二月 2018	
Additional Scanner Checks				22 十二月 2018	Pro extension
Adhoc Payload Processors				06 十一月 2019	
AES Payloads	✓			28 八月 2015	Pro extension
Asset Discovery				12 九月 2019	Pro extension
Attack Surface Detector				08 三月 2019	
AuthMatrix				02 二月 2018	
Authz	✓			01 七月 2014	
Auto Repeater				04 四月 2018	
Auto-Drop Requests				07 十月 2019	
Autorize				10 一月 2020	
AWS Security Checks				18 一月 2018	Pro extension
AWS Signer				18 十月 2019	
Backslash Powered Scanner				19 八月 2019	Pro extension
Batch Scan Report Generator				03 十月 2017	Pro extension
BeanStack - Stack-trace Finge...				11 九月 2019	Pro extension
Blazer				01 二月 2017	
Bookmarks				20 十二月 2019	
Bradamsa				02 七月 2014	
Brida, Burp to Frida bridge	✓			04 十月 2018	
Broken Link Hijacking				23 七月 2019	Pro extension
Browser Repeater				01 七月 2014	
Buby				14 二月 2017	
Burp Beautifier				16 十二月 2019	

is possible to graphically add inspection hooks (that print arguments and
and tamper hooks (that dynamically change the return value of the hooke

Requirements:

- Python 2.7 and the frida and pyro4 modules
- An iOS or Android device with the frida-server running on it (root pr
  the Frida's Gadget (root privileges on the device not required).

Further information:

- Step-by-step tutorial
- Hack in The Box 2018 Amsterdam presentation: slides and video

Author: Federico Dotta, Piergiovanni Cipolloni
Version: 0.3
Source: https://github.com/portswigger/brida
Updated: 04 十月 2018

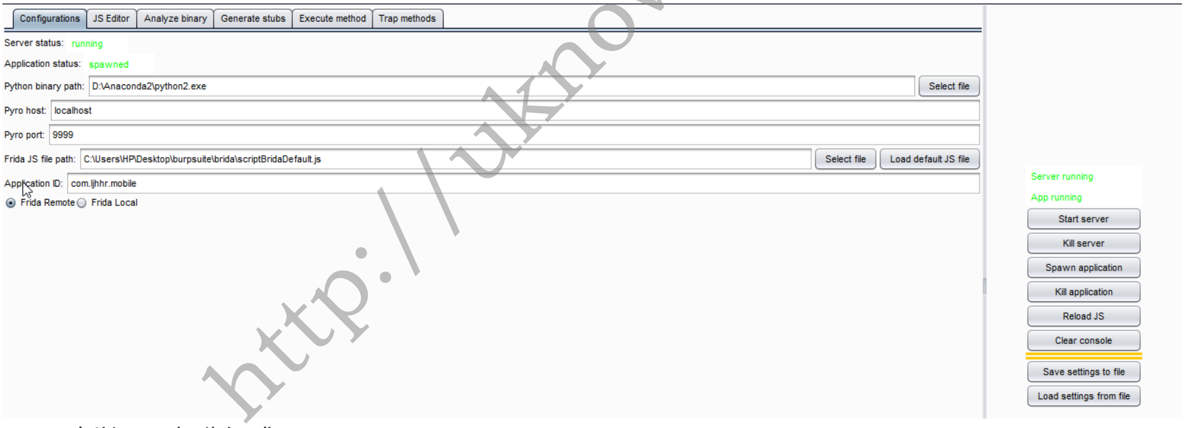
Rating: Submit rating
Popularity:

Reinstall

Brida功能模块

Brida控制台

填好配置项后先点击`Start Server`然后在点击`Spawn application`。



Configurations | JS Editor | Analyze binary | Generate stubs | Execute method | Trap methods

Server status: running
Application status: spawned
Python binary path: D:\Anaconda2\python2.exe [Select file]
Pyro host: localhost
Pyro port: 9999
Frida JS file path: C:\Users\HP\Desktop\burpsuite\brida\scriptBridaDefault.js [Select file] [Load default JS file]
Application ID: com.ljhhr.mobile
[✓ Frida Remote] Frida Local

Server running
App running

Brida由以下三部分组成：

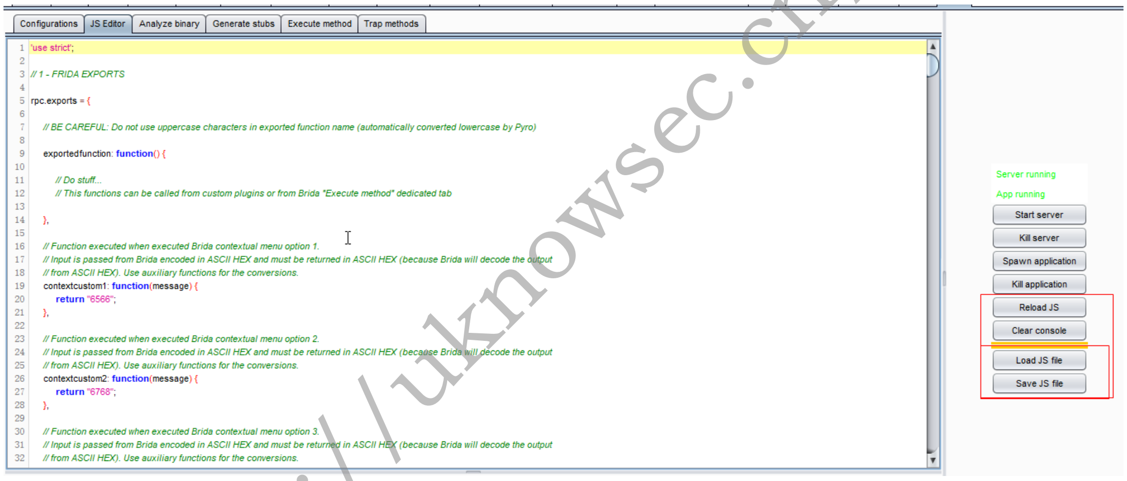
- Brida.jar为Burpsuite插件；
- bridaServicePyro是用于Frida适配到burpsuite上的python脚本，这一部分存储在插件中，在执行brida过程中复制到缓存文件夹中；
- script.js是要注入到目标应用程序的javascript脚本，它会通过Frida带有的rpc.exports功能将信息返回到拓展程序中，同时该script.js脚本会被Frida注入到我们在 Brida中指定的进程中所以我们可以直接使用 Frida的API。

Brida的几个配置参数：

配置参数	说明
Python binary path	即Python可执行程序路径，用于启动Pyro服务
Pyro host, Pyro port	即Pyro 服务的主机以及端口，可以保持默认
Frida JS file path	需要注入的Frida脚本存放的位置
Application ID	目标进程名(APP的包名)
Frida Remote"/"Frida Loca	如果您使用的是Frida USB操作模式，则必须选择" Frida Local"。如果使用端口转发模式，则必须选择" Frida Remote"。

JS编辑器

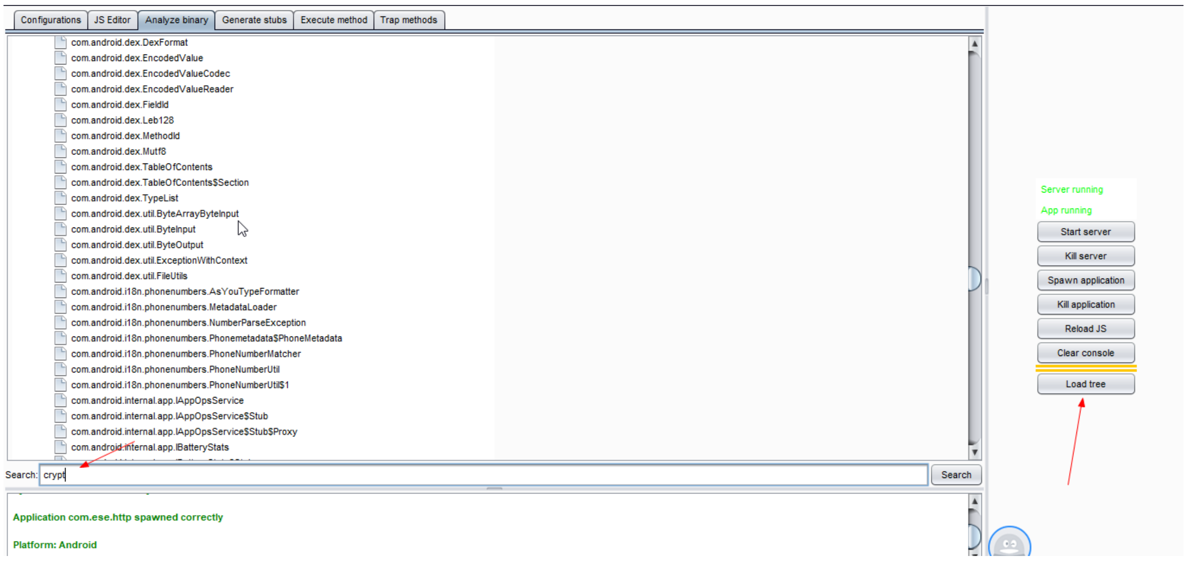
可以实时对hook脚本进行编辑。



```
'use strict';

// 1 - FRIDA EXPORTS

rpc.exports = {

    // BE CAREFUL: Do not use uppercase characters in exported function name (automatically converted lowercase by Pyro)

    exportedfunction: function() {

        // Do stuff...
        // This functions can be called from custom plugins or from Brida "Execute method" dedicated tab

    },

    // Function executed when executed Brida contextual menu option 1.
    // Input is passed from Brida encoded in ASCII HEX and must be returned in ASCII HEX (because Brida will decode the output
    // from ASCII HEX). Use auxiliary functions for the conversions.
    contextcustom1: function(message) {
        return "6566";
    },

    // Function executed when executed Brida contextual menu option 2.
    // Input is passed from Brida encoded in ASCII HEX and must be returned in ASCII HEX (because Brida will decode the output
    // from ASCII HEX). Use auxiliary functions for the conversions.
    contextcustom2: function(message) {
        return "6768";
    },

    // Function executed when executed Brida contextual menu option 3.
    // Input is passed from Brida encoded in ASCII HEX and must be returned in ASCII HEX (because Brida will decode the output
    // from ASCII HEX). Use auxiliary functions for the conversions.
```

Analyze Binary

切换到Analyze Binary，点击Load tree，然后可能会卡一会，因为在加载类列表，加载完点开Java，可以看到这个进程里的所有类，可在下面的搜索框直接搜crypt来找加解密类。



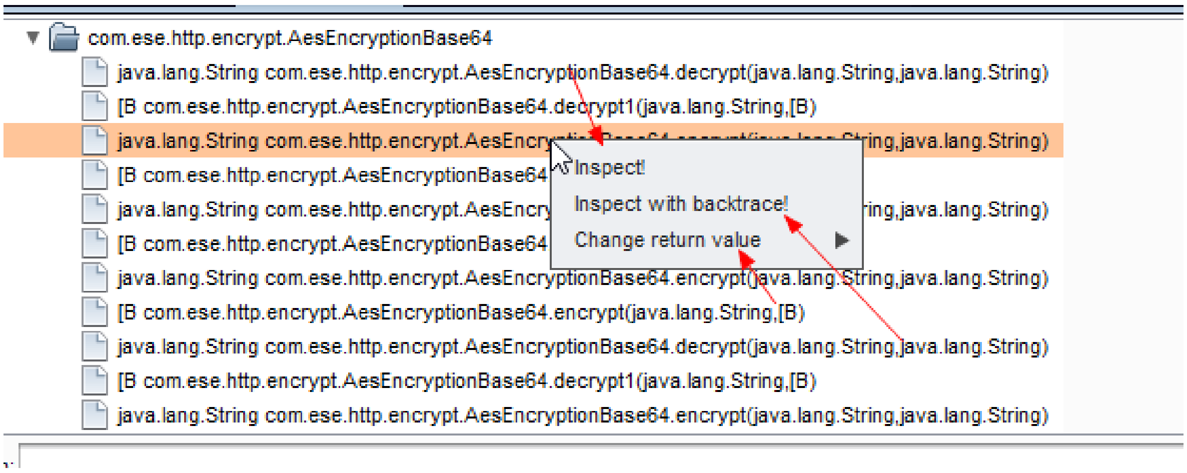
```
com.android.dex.DexFormat
com.android.dex.EncodedValue
com.android.dex.EncodedValueCodec
com.android.dex.EncodedValueReader
com.android.dex.FieldId
com.android.dex.Leb128
com.android.dex.MethodId
com.android.dex.Mutf8
com.android.dex.TableOfContents
com.android.dex.TableOfContents$Section
com.android.dex.TypeList
com.android.dex.util.ByteArrayByteInput
com.android.dex.util.ByteInput
com.android.dex.util.ByteOutput
com.android.dex.util.ExceptionWithContext
com.android.dex.util.FileUtils
com.android.i18n.phonenumbers.AsYouTypeFormatter
com.android.i18n.phonenumbers.MetadataLoader
com.android.i18n.phonenumbers.NumberParseException
com.android.i18n.phonenumbers.Phonemetadata$PhoneMetadata
com.android.i18n.phonenumbers.PhoneNumberMatcher
com.android.i18n.phonenumbers.PhoneNumberUtil
com.android.i18n.phonenumbers.PhoneNumberUtil$1
com.android.internal.app.IAppOpsService
com.android.internal.app.IAppOpsService$Stub
com.android.internal.app.IAppOpsService$Stub$Proxy
com.android.internal.app.IBatteryStats
```

Search: crypt

```
Application com.ese.http spawned correctly

Platform: Android
```

其功能和TraceView 相似，我们在操作APP的时候，他会打印出所有加载的类和方法。

点开java下拉找到app包名通过一个一个插桩 然后进行提交测试 看响应框是否会有信息。

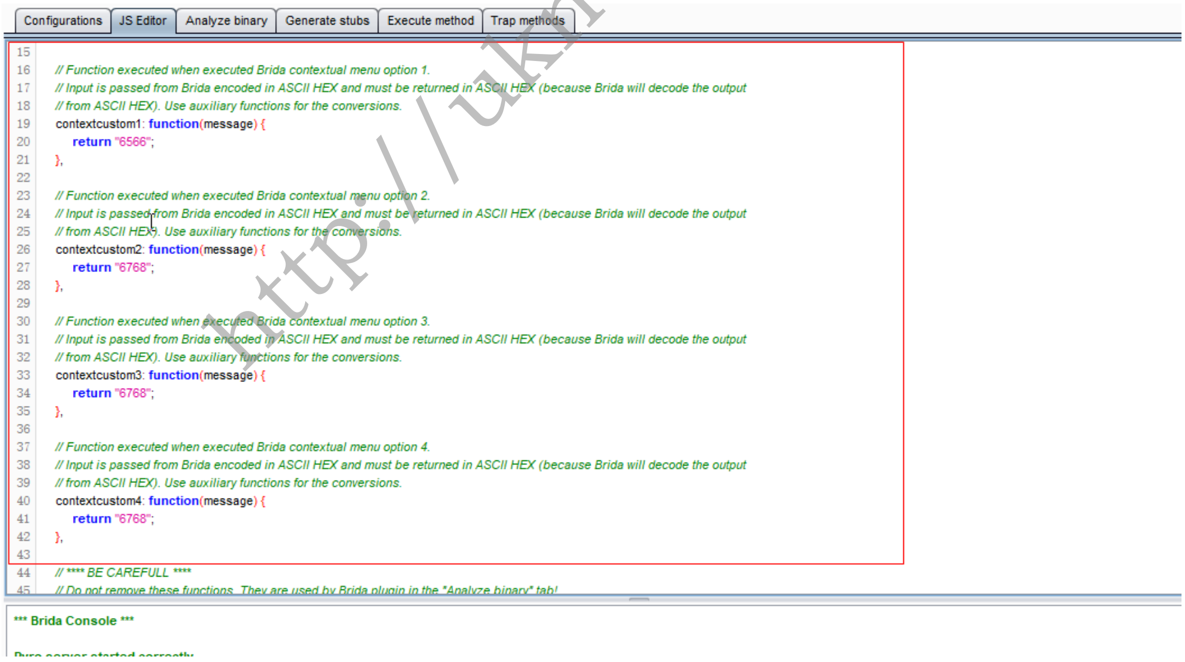


```
com.ese.http.encrypt.AesEncryptionBase64
    java.lang.String com.ese.http.encrypt.AesEncryptionBase64.decrypt(java.lang.String,java.lang.String)
    [B com.ese.http.encrypt.AesEncryptionBase64.decrypt1(java.lang.String,[B)
    java.lang.String com.ese.http.encrypt.AesEncry                               ring,java.lang.String)
    [B com.ese.http.encrypt.AesEncryptionBase64
    java.lang.String com.ese.http.encrypt.AesEncry                               ring,java.lang.String)
    [B com.ese.http.encrypt.AesEncryptionBase64
    java.lang.String com.ese.http.encrypt.AesEncryptionBase64.encrypt(java.lang.String,java.lang.String)
    [B com.ese.http.encrypt.AesEncryptionBase64.encrypt(java.lang.String,[B)
    java.lang.String com.ese.http.encrypt.AesEncryptionBase64.decrypt(java.lang.String,java.lang.String)
    [B com.ese.http.encrypt.AesEncryptionBase64.decrypt1(java.lang.String,[B)
    java.lang.String com.ese.http.encrypt.AesEncryptionBase64.encrypt(java.lang.String,java.lang.String)
```

通过插桩可以看到此方法前后的输入值与返回值，同时可以通过`change return value`修改方法返回值。

Execute method

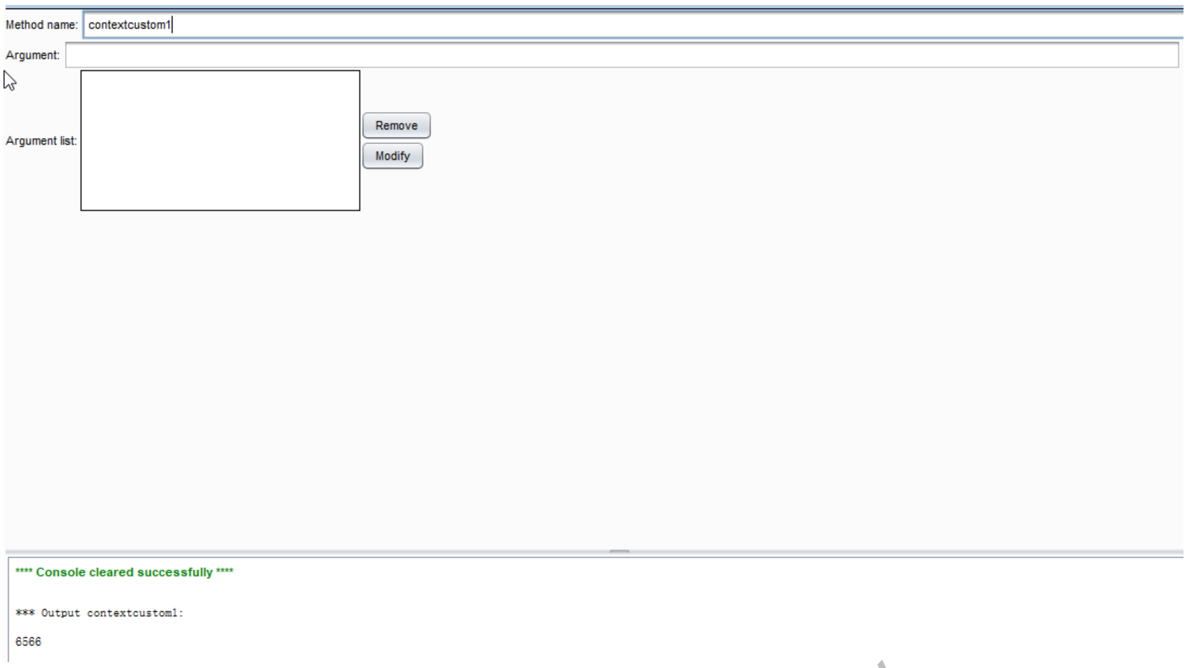
在通用js脚本中的`rpc.exports`里帮写了四个`contextcustom`，这四个是给右键菜单预留的，`contextcustom1`、`contextcustom2`会出现在`repeater`等模块中`request`的右键菜单，`contextcustom2`、`contextcustom3`则会出现在`response`的右键菜单。主要就是为了实现手动加解密的功能。这四个函数接收的参数都是hex形式的，所以返回的时候也要转成hex再传出去。



```

// Function executed when executed Brida contextual menu option 1.
// Input is passed from Brida encoded in ASCII HEX and must be returned in ASCII HEX (because Brida will decode the output
// from ASCII HEX). Use auxiliary functions for the conversions.
contextcustom1: function(message) {
    return "6566";
},

// Function executed when executed Brida contextual menu option 2.
// Input is passed from Brida encoded in ASCII HEX and must be returned in ASCII HEX (because Brida will decode the output
// from ASCII HEX). Use auxiliary functions for the conversions.
contextcustom2: function(message) {
    return "6768";
},

// Function executed when executed Brida contextual menu option 3.
// Input is passed from Brida encoded in ASCII HEX and must be returned in ASCII HEX (because Brida will decode the output
// from ASCII HEX). Use auxiliary functions for the conversions.
contextcustom3: function(message) {
    return "6768";
},

// Function executed when executed Brida contextual menu option 4.
// Input is passed from Brida encoded in ASCII HEX and must be returned in ASCII HEX (because Brida will decode the output
// from ASCII HEX). Use auxiliary functions for the conversions.
contextcustom4: function(message) {
    return "6768";
},

// **** BE CAREFULL ****
// Do not remove these functions. They are used by Brida plugin in the "Analyze binary" tab!
```

```
*** Brida Console ***

Pyro server started correctly
```

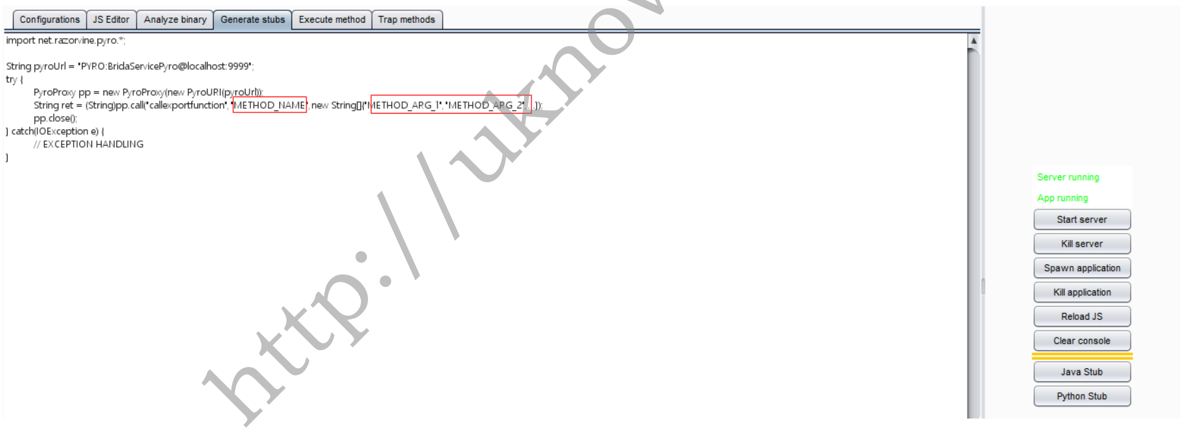
例如：我们掉模块中调用函数contextcustom1返回值为上面代码中的6566。



```
**** Console cleared successfully ****

*** Output contextcustom1:

6566
```

Generate stubs

这里可以生成java/python代码，`METHOD_NAME` 你要调用hook脚本的函数，即如上的contextcustom1，另外一处标红为参数列表。



```
import net.razorvine.pyro.*;

String pyroUrl = "PYRO:BridaServicePyro@localhost:9999";
try {
        PyroProxy pp = new PyroProxy(new PyroURI(pyroUrl));
        String ret = (String)pp.call("callexportfunction", "METHOD_NAME", new String[]{"METHOD_ARG_1","METHOD_ARG_2",...});
        pp.close();
} catch(IOException e) {
        // EXCEPTION HANDLING
}
```

我们要实现的功能只是要让Repeater, Intruder and Scanner模块能对数据进行自动加/解密。

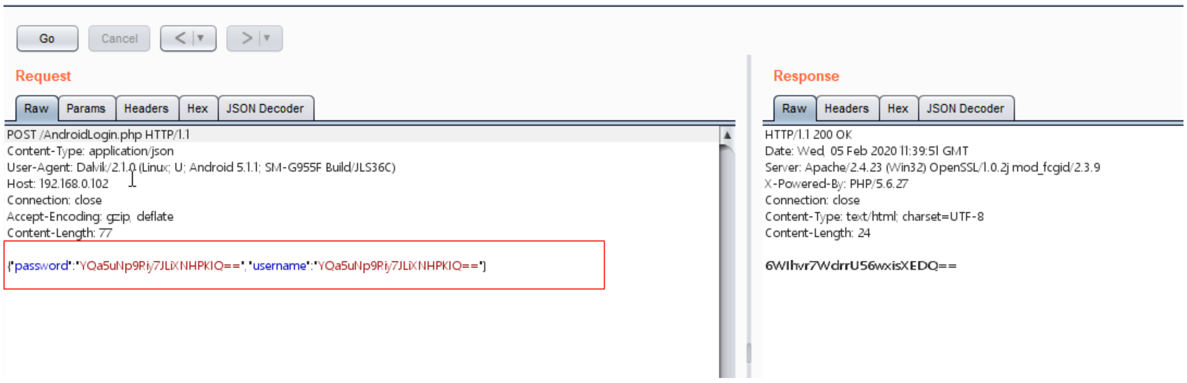
如下代码是brida官方文档给出的通用代码，我们只需修改调用的函数名和参数并对相关的加密解密数据进行处理即可。

```
package burp;

import java.io.PrintWriter;
import java.util.Arrays;
import org.apache.commons.lang3.ArrayUtils;
import net.razorvine.pyro.PyroProxy;
import net.razorvine.pyro.PyroURI;

public class BurpExtender implements IBurpExtender, IHttpListener {
  private PrintWriter stdout;
  private PrintWriter stderr;
  private IBurpExtenderCallbacks callbacks;
  private IExtensionHelpers helpers;
  public void registerExtenderCallbacks(IBurpExtenderCallbacks callbacks) {
    // Set the name of the extension
    callbacks.setExtensionName("Brida Demo Search Plugin");
    // Initialize stdout and stderr (configurable from the Extension pane)
    stdout = new PrintWriter(callbacks.getStdout(), true);
    stderr = new PrintWriter(callbacks.getStderr(), true);
    // Save references to useful objects
    this.callbacks = callbacks;
    this.helpers = callbacks.getHelpers();
    // Register ourselves as an HttpListener, in this way all requests and
responses will be forwarded to us
    callbacks.registerHttpListener(this);
  }

  public void processHttpMessage(int toolFlag, boolean messageIsRequest,
IHttpRequestResponse messageInfo) {

    // Process only Repeater, Scanner and Intruder requests
    if(toolFlag == IBurpExtenderCallbacks.TOOL_SCANNER ||
       toolFlag == IBurpExtenderCallbacks.TOOL_REPEATER ||
       toolFlag == IBurpExtenderCallbacks.TOOL_INTRUDER) {

      // Modify "test" parameter of Repeater requests
      if(messageIsRequest) {
        // Get request bytes
        byte[] request = messageInfo.getRequest();
        // Get a IRequestInfo object, useful to work with the request
        IRequestInfo requestInfo = helpers.analyzeRequest(request);
        // Get "test" parameter
        IParameter contentParameter = helpers.getRequestParameter(request,
"content");
        if(contentParameter != null) {
          String urlDecodedContentParameterValue =
helpers.urlDecode(contentParameter.getValue());
          String ret = "";
          // Ask Brida to encrypt our attack vector
          String pyroUrl = "PYRO:BridaServicePyro@localhost:9999";
          try {
            PyroProxy pp = new PyroProxy(new PyroURI(pyroUrl));
            ret = (String)pp.call("callexportfunction","encryptrequest",new
String[]{urlDecodedContentParameterValue});
            pp.close();
          } catch(Exception e) {
            stderr.println(e.toString());
            StackTraceElement[] exceptionElements = e.getStackTrace();
            for(int i=0; i< exceptionElements.length; i++) {
              stderr.println(exceptionElements[i].toString());
            }
          }
          // Create the new parameter
          IParameter newTestParameter =
helpers.buildParameter(contentParameter.getName(), helpers.urlEncode(ret),
contentParameter.getType());
          // Create the new request with the updated parameter
          byte[] newRequest = helpers.updateParameter(request,
newTestParameter);
```

```
          // Update the messageInfo object with the modified request (otherwise
the request remains the old one)
          messageInfo.setRequest(newRequest);
        }

      // Response
      } else {
        // Get request bytes in order to check if the request contain "content"
parameter
        byte[] request = messageInfo.getRequest();
        IRequestInfo requestInfo = helpers.analyzeRequest(request);
        IParameter contentParameter = helpers.getRequestParameter(request,
"content");
        if(contentParameter != null) {
          // Get response bytes
          byte[] response = messageInfo.getResponse();
          // Get a IResponseInfo object, useful to work with the request
          IResponseInfo responseInfo = helpers.analyzeResponse(response);
          // Get the offset of the body
          int bodyOffset = responseInfo.getBodyOffset();
          // Get the body (byte array and String)
          byte[] body = Arrays.copyOfRange(response, bodyOffset,
response.length);
          String bodyString = helpers.bytesToString(body);
          String ret = "";
          // Ask Brida to decrypt the response
          String pyroUrl = "PYRO:BridaServicePyro@localhost:9999";
          try {
            PyroProxy pp = new PyroProxy(new PyroURI(pyroUrl));
            ret = (String)pp.call("callexportfunction","decryptresponse",new
String[]{bodyString});
            pp.close();
          } catch(Exception e) {
            stderr.println(e.toString());
            StackTraceElement[] exceptionElements = e.getStackTrace();
            for(int i=0; i< exceptionElements.length; i++) {
              stderr.println(exceptionElements[i].toString());
            }
          }
          // Update the messageInfo object with the modified request (otherwise
the request remains the old one)
          byte[] newResponse = ArrayUtils.addAll(Arrays.copyOfRange(response, 0,
bodyOffset),ret.getBytes());
          messageInfo.setResponse(newResponse);
        }
      }
    }
  }
}
```

实战

如图app对数据进行了加密。


Request

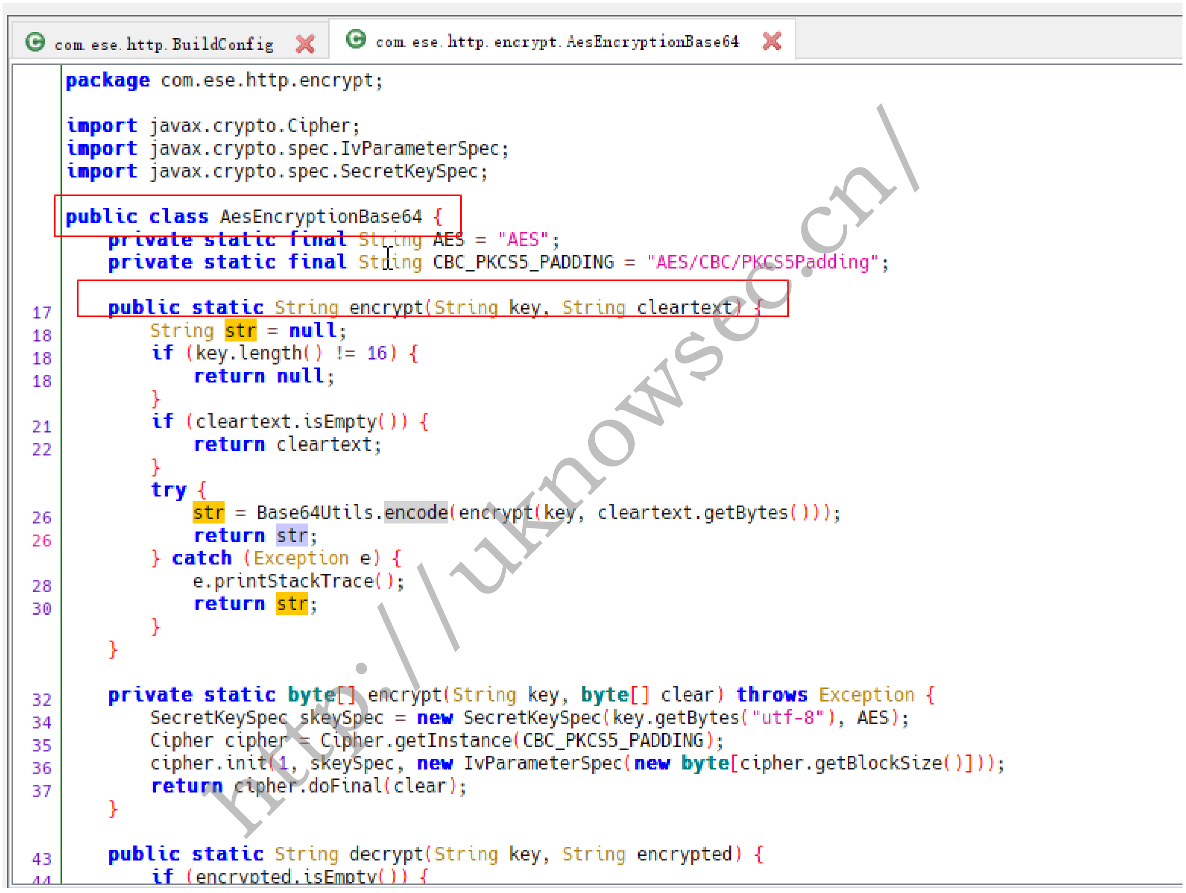
```
POST /AndroidLogin.php HTTP/1.1
Content-Type: application/json
User-Agent: Dalvik/2.1.0 (Linux; U; Android 5.1.1; SM-G955F Build/JLS36C)
Host: 192.168.0.102
Connection: close
Accept-Encoding: gzip, deflate
Content-Length: 77

{"password":"YQa5uNp9Riy7JLiXNHPKIQ==","username":"YQa5uNp9Riy7JLiXNHPKIQ=="}
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 05 Feb 2020 11:39:51 GMT
Server: Apache/2.4.23 (Win32) OpenSSL/1.0.2j mod_fcgid/2.3.9
X-Powered-By: PHP/5.6.27
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 24

6WIhvr7WdrrU56wxisXEDQ==
```

这里我们可以通过直接分析APK或者通过Hook来看他是使用的什么加密，且调用的是那个类的那个方法。

逆向APP可以知道调用的类和方法。

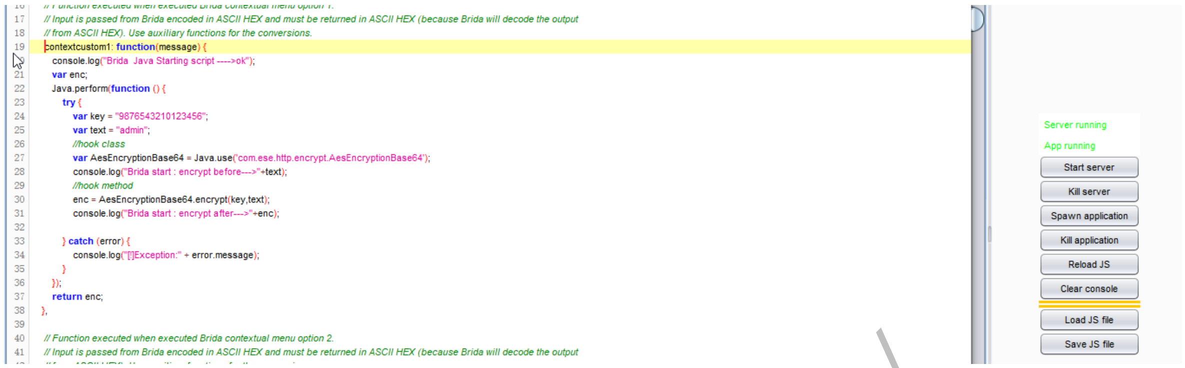


```
package com.ese.http.encrypt;

import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

public class AesEncryptionBase64 {
    private static final String AES = "AES";
    private static final String CBC_PKCS5_PADDING = "AES/CBC/PKCS5Padding";

    public static String encrypt(String key, String cleartext) {
        String str = null;
        if (key.length() != 16) {
            return null;
        }
        if (cleartext.isEmpty()) {
            return cleartext;
        }
        try {
            str = Base64Utils.encode(encrypt(key, cleartext.getBytes()));
            return str;
        } catch (Exception e) {
            e.printStackTrace();
            return str;
        }
    }

    private static byte[] encrypt(String key, byte[] clear) throws Exception {
        SecretKeySpec skeySpec = new SecretKeySpec(key.getBytes("utf-8"), AES);
        Cipher cipher = Cipher.getInstance(CBC_PKCS5_PADDING);
        cipher.init(1, skeySpec, new IvParameterSpec(new byte[cipher.getBlockSize()]));
        return cipher.doFinal(clear);
    }

    public static String decrypt(String key, String encrypted) {
        if (encrypted.isEmpty()) {
```

同时得到加密密钥为： `9876543210123456`

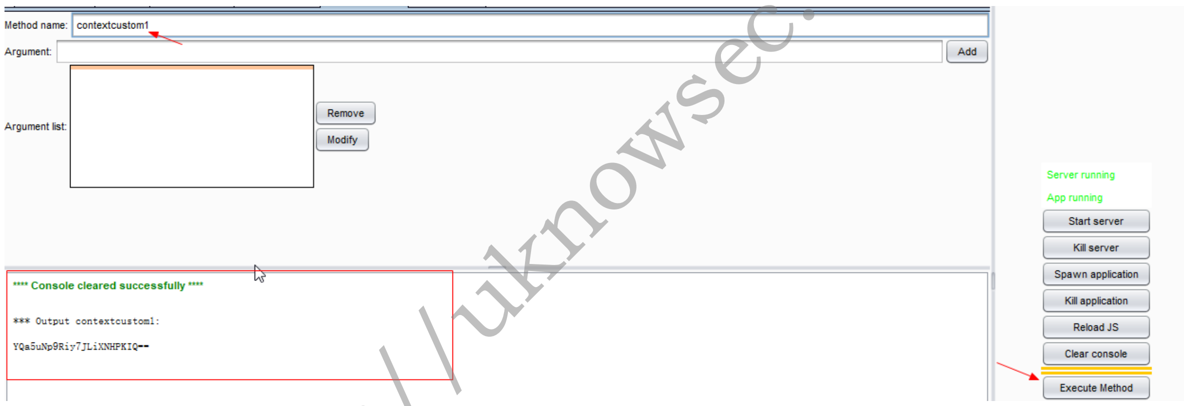
我们可以利用Execute method模块进行函数调用测试。修改contextcustom1

```
contextcustom1: function(message) {
        console.log("Brida  Java Starting script ---->ok");
        var enc;
        Java.perform(function () {
            try {
                var key = "9876543210123456";
                var text = "admin";
                //hook class
                var AesEncryptionBase64 =
Java.use('com.ese.http.encrypt.AesEncryptionBase64');
                console.log("Brida start : encrypt before--->"+text);
                //hook method
                enc = AesEncryptionBase64.encrypt(key,text);
                console.log("Brida start : encrypt after--->"+enc);

            } catch (error) {
                console.log("[!]Exception:" + error.message);
            }
        });
        return enc;
    },
```



```
// Function executed when executed Brida contextual menu option 1.
// Input is passed from Brida encoded in ASCII HEX and must be returned in ASCII HEX (because Brida will decode the output
// from ASCII HEX). Use auxiliary functions for the conversions.
contextcustom1: function(message) {
  console.log("Brida  Java Starting script ---->ok");
  var enc;
  Java.perform(function () {
    try {
      var key = "9876543210123456";
      var text = "admin";
      //hook class
      var AesEncryptionBase64 = Java.use("com.ese.http.encrypt.AesEncryptionBase64");
      console.log("Brida start : encrypt before--->"+text);
      //hook method
      enc = AesEncryptionBase64.encrypt(key,text);
      console.log("Brida start : encrypt after--->"+enc);

    } catch (error) {
      console.log("[!]Exception:" + error.message);
    }
  });
  return enc;
},

// Function executed when executed Brida contextual menu option 2.
// Input is passed from Brida encoded in ASCII HEX and must be returned in ASCII HEX (because Brida will decode the output
```

如上是hook `com.ese.http.encrypt.AesEncryptionBase64` 类的 `encrypt` 方法，并传入key值和加密内容。



Method name: contextcustom1
Argument:
Argument list:

```
**** Console cleared successfully ****

*** Output contextcustom1:

YQa5uNp9Riy7JLiXNHPKIQ==
```

可得到加密后的密文。

4个方法与请求数据包与返回数据包相互一一对应：

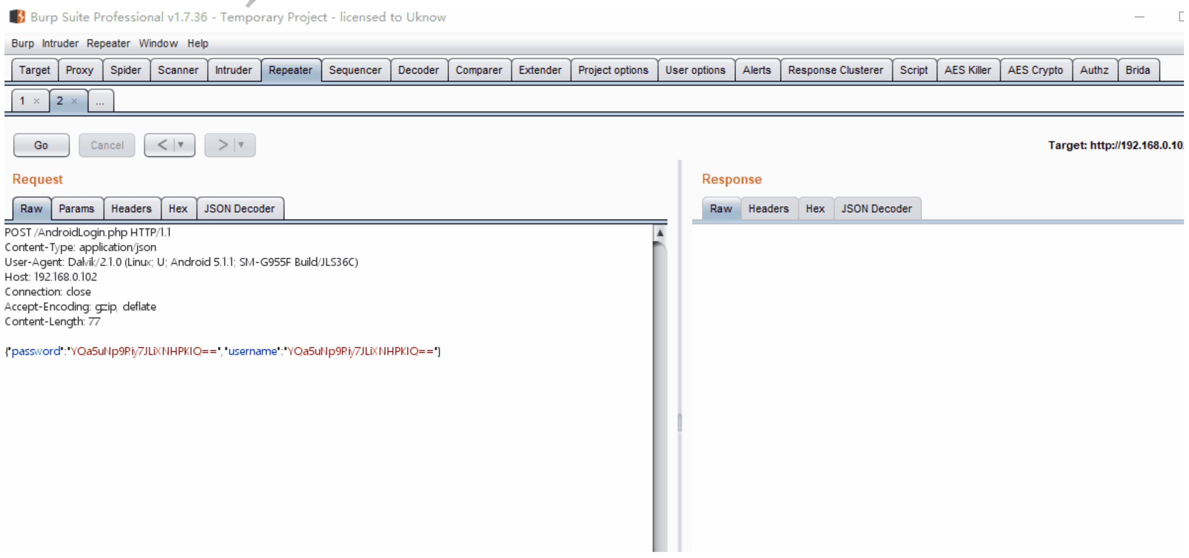
- Brida Custom 1：通过右键菜单进行访问，它会调用contextcustom1 JS脚本；
- Brida Custom 2：通过右键菜单进行访问，它会调用contextcustom2 JS脚本；
- Brida Custom 3：通过右键菜单进行访问，它会调用contextcustom3 JS脚本；
- Brida Custom 4：通过右键菜单进行访问，它会调用contextcustom4 JS脚本。

编写加密解密脚本：（函数接收的参数和返回的数据都是以 16进制编码的，所以我们使用时要先对他们进行16进制解码，然后返回的时候在进行16进制编码。）

```
    //AesEncryptionBase64 encrypt
    contextcustom1: function (message) {
        console.log("Brida start :0--->" + message);
        var data = hexToString(message)
        console.log("Brida start :1--->" + data);
        var enc;
        Java.perform(function () {
            try {
                var key = "9876543210123456";
                var text = data;
                //hook class
                var AesEncryptionBase64 =
Java.use('com.ese.http.encrypt.AesEncryptionBase64');
                console.log("Brida start : AesEncryptionBase64 ---> success");
                console.log("Brida start : encrypt before--->"+text);
                //hook method
                enc = AesEncryptionBase64.encrypt(key,text);
                console.log("Brida start : encrypt after--->"+enc);

            } catch (error) {
                console.log("[!]Exception:" + error.message);
            }
        });
        return stringToHex(enc);
    },

    //AesEncryptionBase64 decrypt
    contextcustom2: function (message) {
        console.log("Brida start :0--->" + message);
        var data = hexToString(message)
        console.log("Brida start :1--->" + data);
        var text;
        Java.perform(function () {
            try {
                var key = "9876543210123456";
                var enc = data;
                //hook class
                var AesEncryptionBase64 =
Java.use('com.ese.http.encrypt.AesEncryptionBase64');
                console.log("Brida start : AesEncryptionBase64 ---> success");
                console.log("Brida start : decrypt before--->"+enc);
                //hook method
                text = AesEncryptionBase64.decrypt(key,enc);
                console.log("Brida start : decrypt after--->"+text);
            } catch (error) {
                console.log("[!]Exception:" + error.message);
            }
        });
        console.log("Brida start : decrypt after--->"+stringToHex(text));
        return stringToHex(text);
    },

    //AesEncryptionBase64 encrypt
    contextcustom3: function (message) {
        console.log("Brida start :0--->" + message);
        var data = hexToString(message)
        console.log("Brida start :1--->" + data);
        var enc;
        Java.perform(function () {
            try {
                var key = "9876543210123456";
                var text = data;
                //hook class
                var AesEncryptionBase64 =
Java.use('com.ese.http.encrypt.AesEncryptionBase64');
                console.log("Brida start : AesEncryptionBase64 ---> success");
                console.log("Brida start : encrypt before--->"+text);
                //hook method
                enc = AesEncryptionBase64.encrypt(key,text);
                console.log("Brida start : encrypt after--->"+enc);

            } catch (error) {
                console.log("[!]Exception:" + error.message);
            }
        });
        return stringToHex(enc);
    },

    //AesEncryptionBase64 decrypt
    contextcustom4: function (message) {
        console.log("Brida start :0--->" + message);
        var data = hexToString(message)
        console.log("Brida start :1--->" + data);
        var text;
        Java.perform(function () {
            try {
                var key = "9876543210123456";
                var enc = data;
                //hook class
                var AesEncryptionBase64 =
Java.use('com.ese.http.encrypt.AesEncryptionBase64');
                console.log("Brida start : AesEncryptionBase64 ---> success");
                console.log("Brida start : decrypt before--->"+enc);
                //hook method
                text = AesEncryptionBase64.decrypt(key,enc);
                console.log("Brida start : decrypt after--->"+text);
            } catch (error) {
                console.log("[!]Exception:" + error.message);
            }
        });
        console.log("Brida start : decrypt after--->"+stringToHex(text));
        return stringToHex(text);
    }
```

替换脚本里的contextcustom函数。

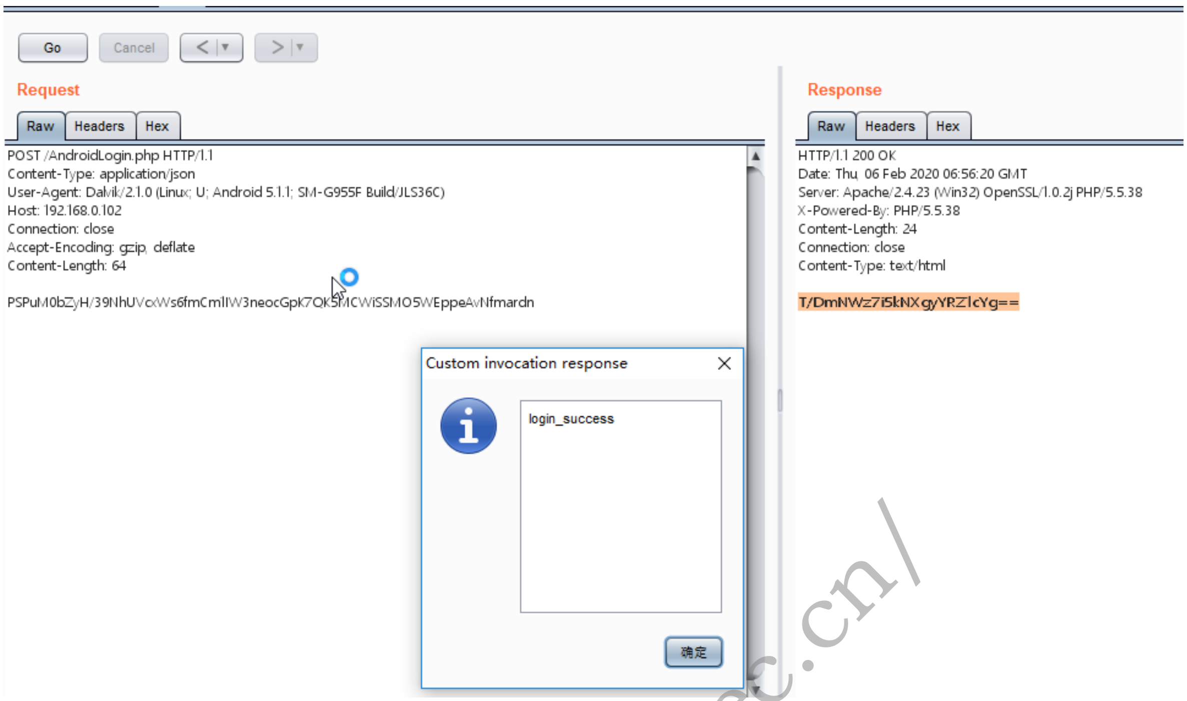
这样就能在burpsuite中进行右键菜单转换了。


Burp Suite Professional v1.7.36 - Temporary Project - licensed to Uknow
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts Response Clusterer Script AES Killer AES Crypto Authz Brida
1 × 2 × ...
Go Cancel
Target: http://192.168.0.10
Request
Raw Params Headers Hex JSON Decoder
Response
Raw Headers Hex JSON Decoder

```
POST /AndroidLogin.php HTTP/1.1
Content-Type: application/json
User-Agent: Dalvik/2.1.0 (Linux; U; Android 5.1.1; SM-G955F Build/JLS36C)
Host: 192.168.0.102
Connection: close
Accept-Encoding: gzip, deflate
Content-Length: 77

{"password":"YQa5uNp9Riy7JLiXNHPKIQ==","username":"YQa5uNp9Riy7JLiXNHPKIQ=="}
```

接下来实现自定义插件实现在`reperter`、`scanner`、`intruder`模板中自动加密请求包和解密返回包。

为了让处理加解密数据更方便，所以就改变了一下服务端加解密的方式。



```
POST /AndroidLogin.php HTTP/1.1
Content-Type: application/json
User-Agent: Dalvik/2.1.0 (Linux; U; Android 5.1.1; SM-G955F Build/JLS36C)
Host: 192.168.0.102
Connection: close
Accept-Encoding: gzip, deflate
Content-Length: 64

PSPuM0bZyH/39NhUVcxWs6fmCmlIW3neocGpK7QK5MCWiSSMO5WEppeAvNfmardn
```

```
HTTP/1.1 200 OK
Date: Thu, 06 Feb 2020 06:56:20 GMT
Server: Apache/2.4.23 (Win32) OpenSSL/1.0.2j PHP/5.5.38
X-Powered-By: PHP/5.5.38
Content-Length: 24
Connection: close
Content-Type: text/html

T/DmNWz7i5kNXgyYRZlcYg==
```

```
login_success
```

这里我用的python编写插件。所以需要安装jython

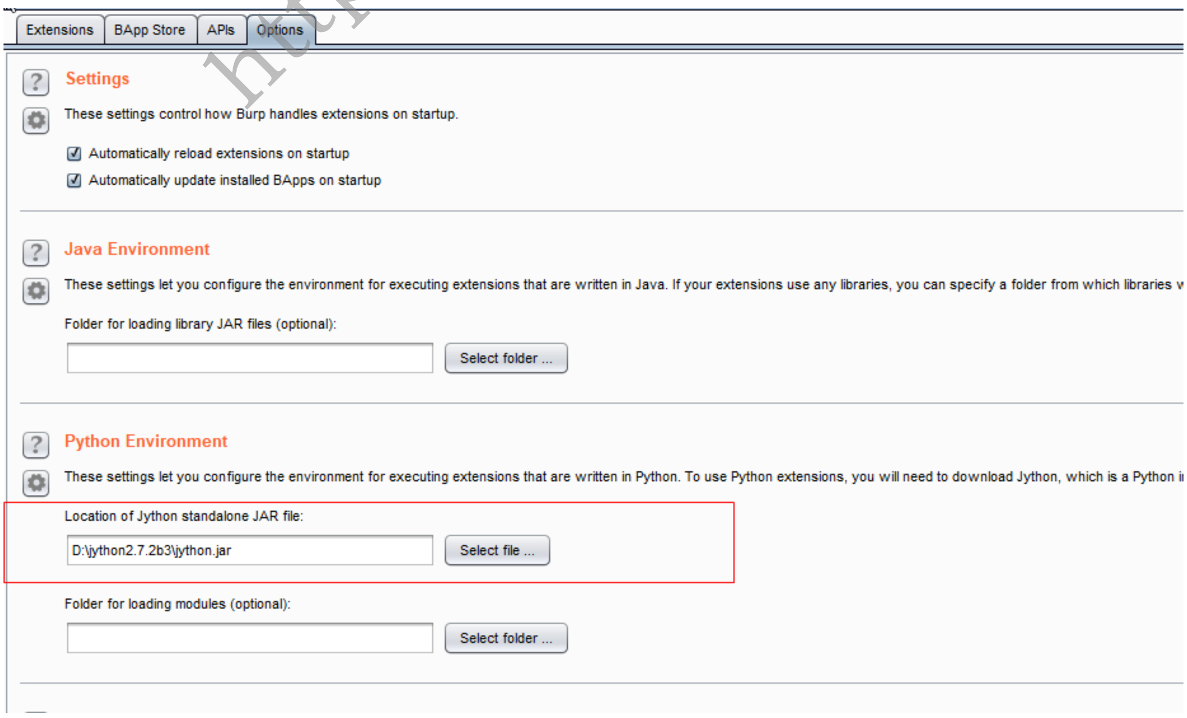
推荐安装最新版的`jython-installer-2.7.2b3.jar`

https://repo1.maven.org/maven2/org/python/jython-installer/2.7.2b3/jython-installer-2.7.2b3.jar

同时使用jython自带的pip安装Pyro4

```
pip install pyro4
```

burp中配置最新安装好pyro4的路径。



Settings

These settings control how Burp handles extensions on startup.

☑ Automatically reload extensions on startup

☑ Automatically update installed BApps on startup

Java Environment

These settings let you configure the environment for executing extensions that are written in Java. If your extensions use any libraries, you can specify a folder from which libraries w

Folder for loading library JAR files (optional):

Python Environment

These settings let you configure the environment for executing extensions that are written in Python. To use Python extensions, you will need to download Jython, which is a Python i

Location of Jython standalone JAR file:

D:\jython2.7.2b3\jython.jar

Folder for loading modules (optional):

然后用python编写burp插件。

申明一个类，继承于`IBurpExtender`和`IHttpListener`：

```
`class` `BurpExtender(IBurpExtender, IHttpListener)`
```

重写`registerExtenderCallbacks`和`processHttpMessage`：

```
def registerExtenderCallbacks(self, callbacks):
        self._callbacks = callbacks
        self._helpers = callbacks.getHelpers()
        self._callbacks.setExtensionName("fuck encrypt by Uknow!")
        callbacks.registerHttpListener(self)
```

```
    def processHttpMessage(self, toolFlag, messageIsRequest, messageInfo):
        # tool https://portswigger.net/burp/extender/api/constant-
values.html#burp.IBurpExtenderCallbacks
        if toolFlag == 64 or toolFlag == 16 or toolFlag == 32: # TOOL_REPEATER
  TOOL_SCANNER     TOOL_INTRUDER
            request = messageInfo.getRequest()
            analyzedRequest = self._helpers.analyzeRequest(request)
            headers = analyzedRequest.getHeaders()
            if not messageIsRequest:
                response = messageInfo.getResponse()
                analyzedResponse = self._helpers.analyzeResponse(response)
                messageInfo.setResponse(self.decrypt(analyzedResponse,
response))
            # elif toolFlag != 4:
            else:
                messageInfo.setRequest(self.encrypt(analyzedRequest, request))
```

对着toolFlag一顿if是为了过滤Burp的模块，判断他是从哪过来的，这里是过滤了三个：`reperter`、`scanner`、`intruder`，抓包过来的无需处理，如果你处理了那APP就不能正常收发数据了。`self.decrypt`和`self.encrypt`就是去跟`Brida`开的端口交换数据，处理加解密：

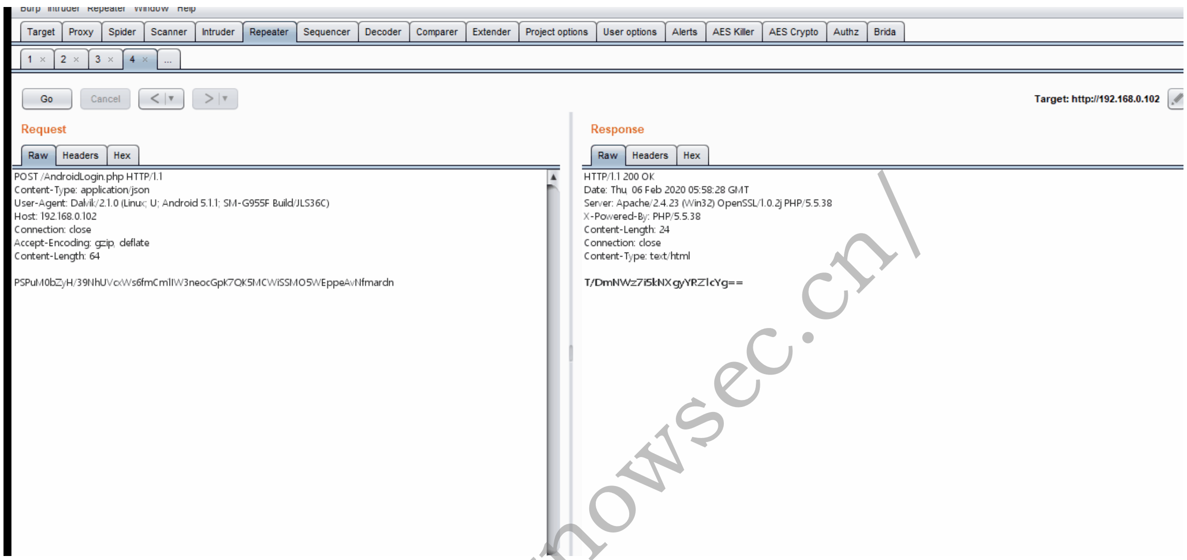
```
def decrypt(self, RequestOrResponse,raw):
    uri = 'PYRO:BridaServicePyro@localhost:9999'
    pp = Pyro4.Proxy(uri)
    body = raw[RequestOrResponse.getBodyOffset():]
    newbody = body.tostring()
    args = []
    args.append(newbody.encode('hex'))
    ret = pp.callexportfunction('contextcustom4',args)
    ret = self._helpers.bytesToString(ret).decode('hex')
    return self._helpers.buildHttpMessage(RequestOrResponse.getHeaders(), ret)

def encrypt(self, RequestOrResponse,raw):
    uri = 'PYRO:BridaServicePyro@localhost:9999'
    pp = Pyro4.Proxy(uri)
    body = raw[RequestOrResponse.getBodyOffset():]
    newbody = body.tostring()
    args = []
    args.append(newbody.encode('hex'))
    ret = pp.callexportfunction('contextcustom1',args)
    ret = self._helpers.bytesToString(ret).decode('hex')
    return self._helpers.buildHttpMessage(RequestOrResponse.getHeaders(), ret)
```

用`callexportfunction`来调用你刚才js脚本里`rpc.exports`里的函数，参数是函数名和参数列表。

完整代码如下：

```
from burp import IBurpExtender
from burp import IHttpListener
from burp import IHttpRequestResponse
from burp import IResponseInfo
from burp import IProxyListener
import Pyro4


print 'uknowsec.cn'

class BurpExtender(IBurpExtender,IHttpListener,IHttpRequestResponse,
IProxyListener):
    def registerExtenderCallbacks(self, callbacks):
        self._callbacks = callbacks
        self._helpers = callbacks.getHelpers()
        self._callbacks.setExtensionName("fuck encrypt by Uknow!")
        callbacks.registerHttpListener(self)

    def decrypt(self, RequestOrResponse,raw):
        uri = 'PYRO:BridaServicePyro@localhost:9999'
        pp = Pyro4.Proxy(uri)
        body = raw[RequestOrResponse.getBodyOffset():]
        newbody = body.tostring()
        args = []
        args.append(newbody.encode('hex'))
        ret = pp.callexportfunction('contextcustom4',args)
        ret = self._helpers.bytesToString(ret).decode('hex')
        return self._helpers.buildHttpMessage(RequestOrResponse.getHeaders(),
ret)

    def encrypt(self, RequestOrResponse,raw):
        uri = 'PYRO:BridaServicePyro@localhost:9999'
        pp = Pyro4.Proxy(uri)
        body = raw[RequestOrResponse.getBodyOffset():]
        newbody = body.tostring()
        args = []
        args.append(newbody.encode('hex'))
        ret = pp.callexportfunction('contextcustom1',args)
        ret = self._helpers.bytesToString(ret).decode('hex')
        return self._helpers.buildHttpMessage(RequestOrResponse.getHeaders(),
ret)

    def processHttpMessage(self, toolFlag, messageIsRequest, messageInfo):
        # tool https://portswigger.net/burp/extender/api/constant-
values.html#burp.IBurpExtenderCallbacks
        if toolFlag == 64 or toolFlag == 16 or toolFlag == 32: # TOOL_REPEATER
  TOOL_SCANNER     TOOL_INTRUDER
            request = messageInfo.getRequest()
            analyzedRequest = self._helpers.analyzeRequest(request)
            headers = analyzedRequest.getHeaders()
            if not messageIsRequest:
                response = messageInfo.getResponse()
                analyzedResponse = self._helpers.analyzeResponse(response)
                messageInfo.setResponse(self.decrypt(analyzedResponse,
response))
            # elif toolFlag != 4:
            else:
                messageInfo.setRequest(self.encrypt(analyzedRequest, request))
```

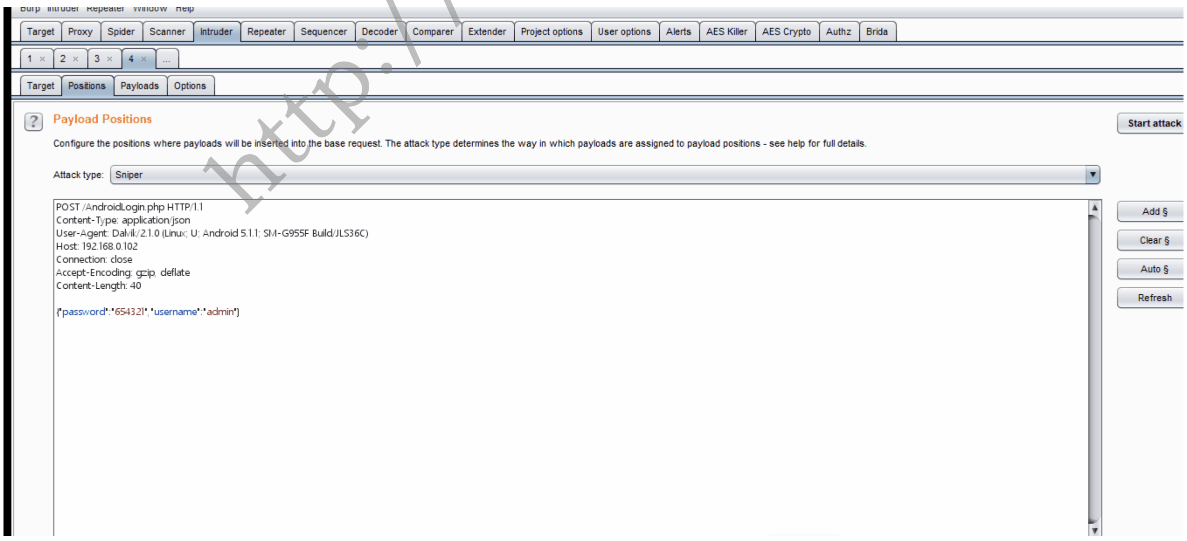
然后加载就可以在`reperter`、`scanner`、`intruder`模块中实现自动加密解密了。操作如下图。



```
POST /AndroidLogin.php HTTP/1.1
Content-Type: application/json
User-Agent: Dalvik/2.1.0 (Linux; U; Android 5.1.1; SM-G955F Build/JLS36C)
Host: 192.168.0.102
Connection: close
Accept-Encoding: gzip, deflate
Content-Length: 64

PSPuM0bZyH/39NhUVcxWs6fmCmlIW3neocGpK7QK5MCWiSSMO5WEppeAvNfmardn
```

```
HTTP/1.1 200 OK
Date: Thu, 06 Feb 2020 05:58:28 GMT
Server: Apache/2.4.23 (Win32) OpenSSL/1.0.2j PHP/5.5.38
X-Powered-By: PHP/5.5.38
Content-Length: 24
Connection: close
Content-Type: text/html

T/DmNWz7i5kNXgyYRZ1cYg==
```

比如如上场景可直接进行爆破。



Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Sniper

```
POST /AndroidLogin.php HTTP/1.1
Content-Type: application/json
User-Agent: Dalvik/2.1.0 (Linux; U; Android 5.1.1; SM-G955F Build/JLS36C)
Host: 192.168.0.102
Connection: close
Accept-Encoding: gzip, deflate
Content-Length: 40

{"password":"654321","username":"admin"}
```

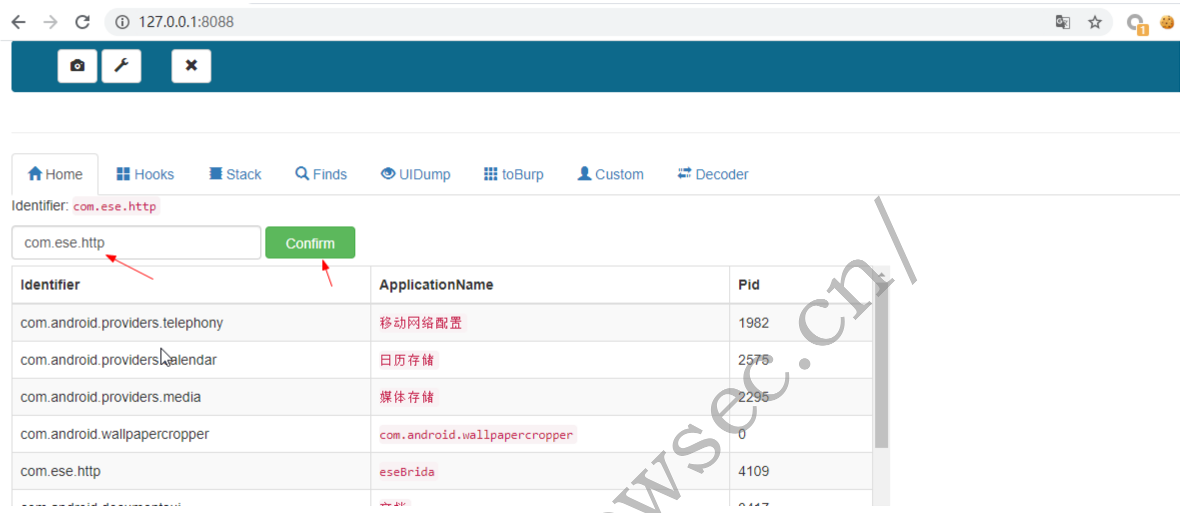
自动以插件也可以使用Java编写，可以根据文章前文给出的官方文档给出的示例进行修改。

lxhToolHTTPDecrypt

https://github.com/lyxhh/lxhToolHTTPDecrypt

HTTP Decrypt 提供了Finds Hooks模块，可以在不逆向不脱壳的情况下快速的找到APP所使用的加解密算法，而toBurp模块提供了直接使用APP内的方法进行加解密，而不需自己动手敲代码，对于整体POST加密更是提供了自动化加解密功能，可以实现Burp一条龙，Burp Scanner，Intruder自动加解密。

1. python3 app.py
2. Android_frida_server 运行
3. 转发frida端口。
4. 打开HTTP Decrypt页面，如果在Start界面出现应用包名列表信息则可正常使用其他功能，如果不行，刷新一下看看控制台出现的信息。



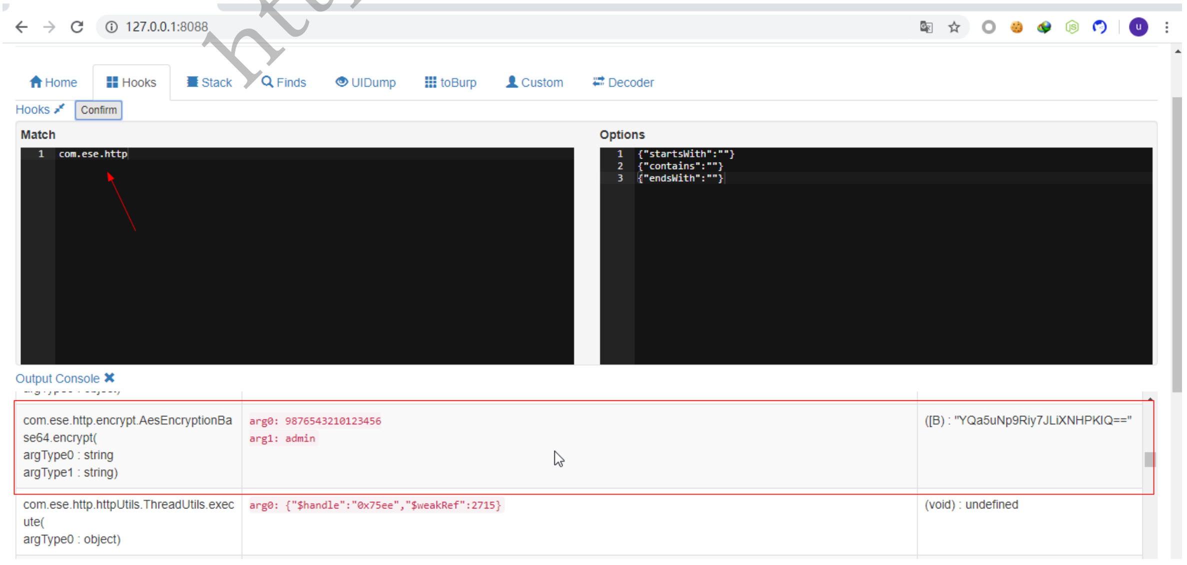
Identifier	ApplicationName	Pid
com.android.providers.telephony	移动网络配置	1982
com.android.providers.calendar	日历存储	2575
com.android.providers.media	媒体存储	2295
com.android.wallpapercropper	com.android.wallpapercropper	0
com.ese.http	eseBrida	4109

Hooks

填写字符串，将类名与你填写的字符串匹配，并Hooks类下的所有方法，hook多个类名，回车换行。

在APP中进行操作，尽量进行多次操作，让脚本hook到所有方法。

如下图，这里找到了加密方法为 `com.ese.http.encrypt.AesEncryptionBase64.encrypt`



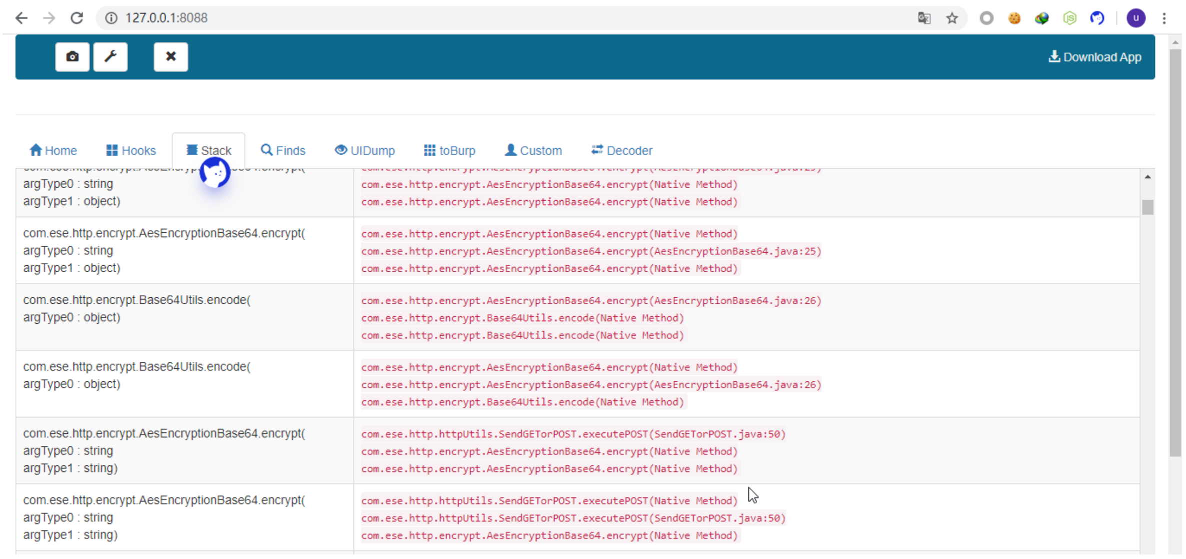
Match:
```
com.ese.http
```

Options:
```
{"startsWith":""}
{"contains":""}
{"endsWith":""}
```

```
com.ese.http.encrypt.AesEncryptionBa    arg0: 9876543210123456    ([B) : "YQa5uNp9Riy7JLiXNHPKIQ=="
se64.encrypt(                           arg1: admin
argType0 : string
argType1 : string)

com.ese.http.httpUtils.ThreadUtils.exec arg0: {"$handle":"0x75ee","$weakRef":2715}    (void) : undefined
ute(
argType0 : object)
```

Stack

像Brida一样也可以显示Hooks打印的堆栈。



argType0 : string argType1 : object)	com.ese.http.encrypt.AesEncryptionBase64.encrypt(Native Method) com.ese.http.encrypt.AesEncryptionBase64.encrypt(Native Method)

com.ese.http.encrypt.AesEncryptionBase64.encrypt(
argType0 : string
argType1 : object)
	com.ese.http.encrypt.AesEncryptionBase64.encrypt(Native Method)
	com.ese.http.encrypt.AesEncryptionBase64.encrypt(AesEncryptionBase64.java:25)
	com.ese.http.encrypt.AesEncryptionBase64.encrypt(Native Method)

com.ese.http.encrypt.Base64Utils.encode(
argType0 : object)
	com.ese.http.encrypt.AesEncryptionBase64.encrypt(AesEncryptionBase64.java:26)
	com.ese.http.encrypt.Base64Utils.encode(Native Method)
	com.ese.http.encrypt.Base64Utils.encode(Native Method)

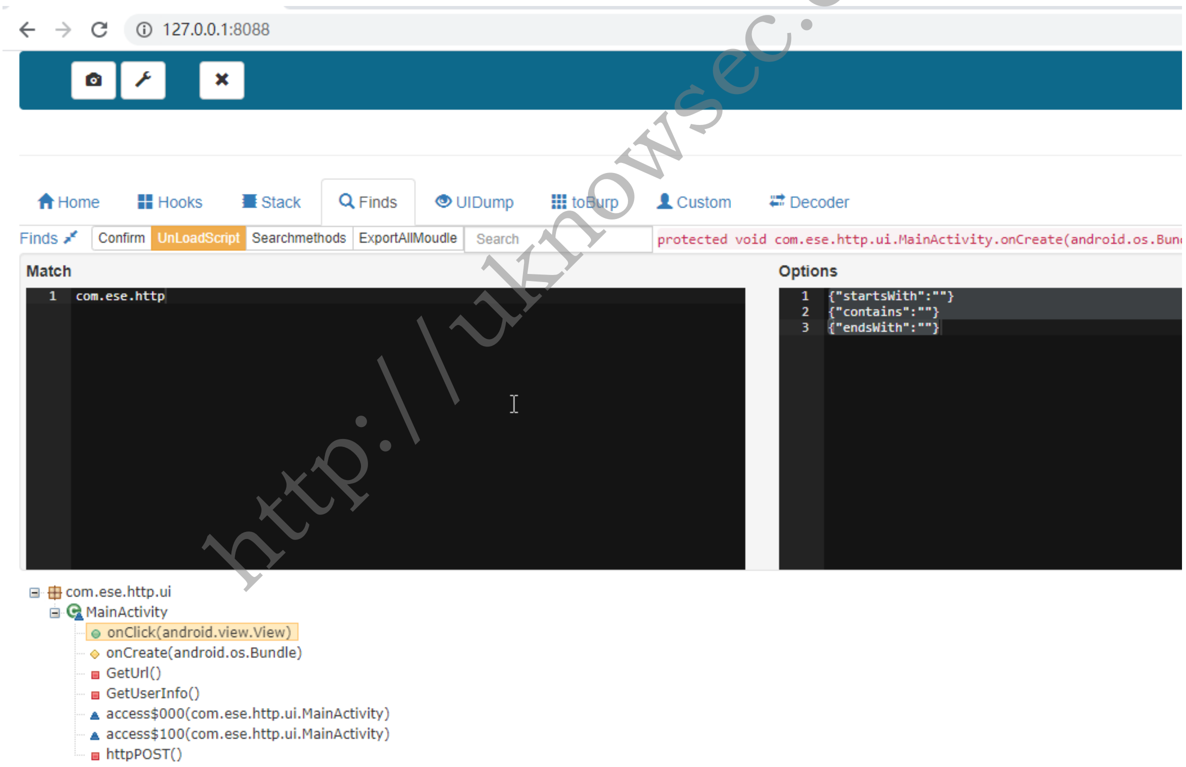
com.ese.http.encrypt.Base64Utils.encode(
argType0 : object)
	com.ese.http.encrypt.AesEncryptionBase64.encrypt(Native Method)
	com.ese.http.encrypt.AesEncryptionBase64.encrypt(AesEncryptionBase64.java:26)
	com.ese.http.encrypt.Base64Utils.encode(Native Method)

com.ese.http.encrypt.AesEncryptionBase64.encrypt(
argType0 : string
argType1 : string)
	com.ese.http.httpUtils.SendGETorPOST.executePOST(SendGETorPOST.java:50)
	com.ese.http.encrypt.AesEncryptionBase64.encrypt(Native Method)
	com.ese.http.encrypt.AesEncryptionBase64.encrypt(Native Method)

com.ese.http.encrypt.AesEncryptionBase64.encrypt(
argType0 : string
argType1 : string)
	com.ese.http.httpUtils.SendGETorPOST.executePOST(Native Method)
	com.ese.http.httpUtils.SendGETorPOST.executePOST(SendGETorPOST.java:50)
	com.ese.http.encrypt.AesEncryptionBase64.encrypt(Native Method)

Finds

根据字符串，查找类，匹配到类，将类下的方法都打印出来。多个查找回车换行继续写。提供了过滤机制。



```
protected void com.ese.http.ui.MainActivity.onCreate(android.os.Bun
```

Match:

```
com.ese.http
```

Options:

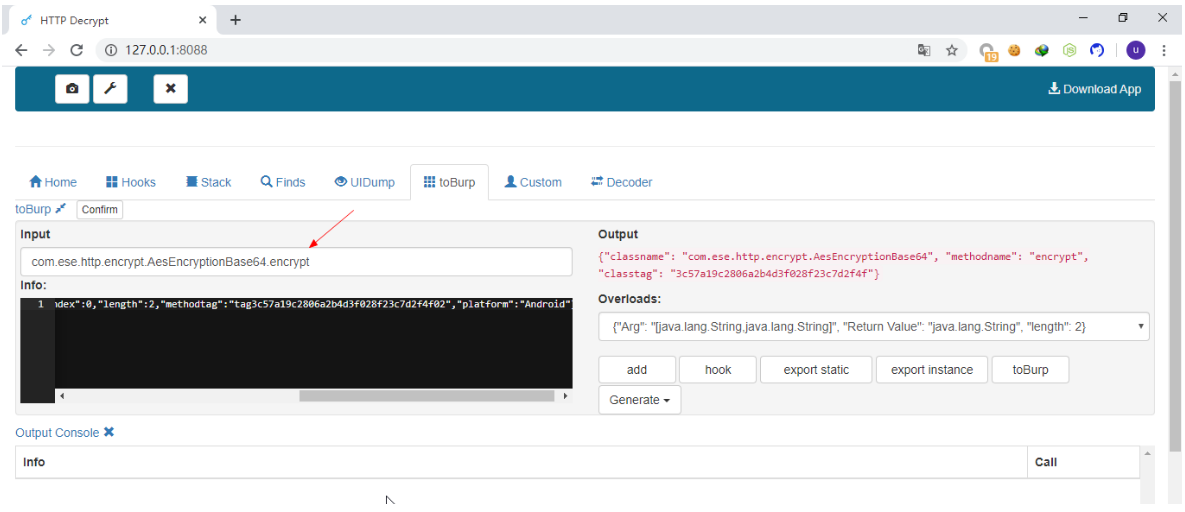
```
{"startsWith":""}
{"contains":""}
{"endsWith":""}
```

```
com.ese.http.ui
  MainActivity
    onClick(android.view.View)
    onCreate(android.os.Bundle)
    GetUrl()
    GetUserInfo()
    access$000(com.ese.http.ui.MainActivity)
    access$100(com.ese.http.ui.MainActivity)
    httpPOST()
```

匹配的不是很准确，可能是我姿势有问题，但是我们可以自己去反编译或者通过hook来判断哪个是加解密方法。

toBurp

添入我们得到加解密方法。



toBurp Confirm

Input

com.ese.http.encrypt.AesEncryptionBase64.encrypt

Info:

```
dex":0,"length":2,"methodtag":"tag3c57a19c2806a2b4d3f028f23c7d2f4f02","platform":"Android"
```

Output

```
{"classname": "com.ese.http.encrypt.AesEncryptionBase64", "methodname": "encrypt",
"classtag": "3c57a19c2806a2b4d3f028f23c7d2f4f"}
```

Overloads:

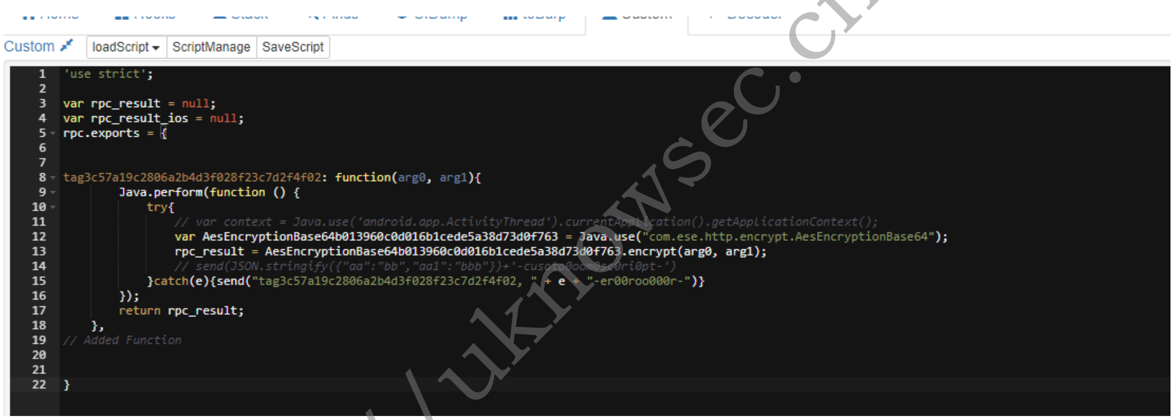
{"Arg": "[java.lang.String,java.lang.String]", "Return Value": "java.lang.String", "length": 2}

Output Console

Info	Call

然后点击Confirm按钮，之后再点击Add按钮，将方法的信息添加到左边的info里面去，因为加解密方法有两个参数，无法使用HTTPDecrypt自动的生成的脚本（自动生成的脚本只能适应一个参数），所以我们需要自己写一些代码，接下来我们点击Generate export static script按钮。（因为方法类型是静态的所以选择static按钮，动态的选择instance，如果你很熟悉frida，点哪个都无所谓。）

HTTPDecrypt会将一些代码生成在Custom选项卡中，如下：



Custom loadScript ScriptManage SaveScript

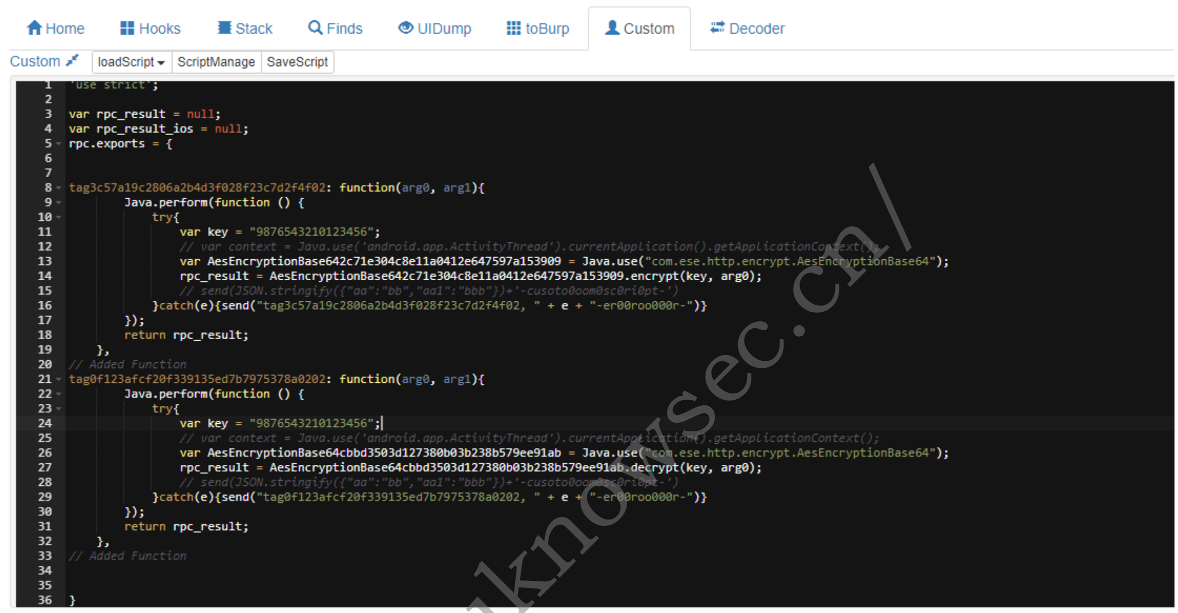
```
'use strict';

var rpc_result = null;
var rpc_result_ios = null;
rpc.exports = {


tag3c57a19c2806a2b4d3f028f23c7d2f4f02: function(arg0, arg1){
        Java.perform(function () {
            try{
                // var context = Java.use('android.app.ActivityThread').currentApplication().getApplicationContext();
                var AesEncryptionBase64b013960c0d016b1cede5a38d73d0f763 = Java.use("com.ese.http.encrypt.AesEncryptionBase64");
                rpc_result = AesEncryptionBase64b013960c0d016b1cede5a38d73d0f763.encrypt(arg0, arg1);
                // send(JSON.stringify({"aa":"bb","aa1":"bbb"})+'-cusoto0oom0sc0ri0pt-')
            }catch(e){send("tag3c57a19c2806a2b4d3f028f23c7d2f4f02, " + e + "-er00roo000r-")}
        });
        return rpc_result;
    },
// Added Function


}
```

像Brida修改代码，添加key值。encrypt方法的第一个参数是一个固定值key，通过反编译和hook分析可以得到。第二个参数为加密内容。

```
'use strict';

var rpc_result = null;
var rpc_result_ios = null;
rpc.exports = {



tag3c57a19c2806a2b4d3f028f23c7d2f4f02: function(arg0, arg1){
        Java.perform(function () {
            try{
                var key = "9876543210123456";
                // var context =
Java.use('android.app.ActivityThread').currentApplication().getApplicationContex
t();
                var AesEncryptionBase64bc9333b9adb0ceb7cc6c929d900e3365 =
Java.use("com.ese.http.encrypt.AesEncryptionBase64");
                rpc_result =
AesEncryptionBase64bc9333b9adb0ceb7cc6c929d900e3365.encrypt(key, arg0);
                // send(JSON.stringify({"aa":"bb","aa1":"bbb"})+'-
cusoto0oom0sc0ri0pt-')
```

```
            }catch(e){send("tag3c57a19c2806a2b4d3f028f23c7d2f4f02, " + e + "-
er00roo000r-")}
        });
        return rpc_result;
    },
// Added Function


}
```

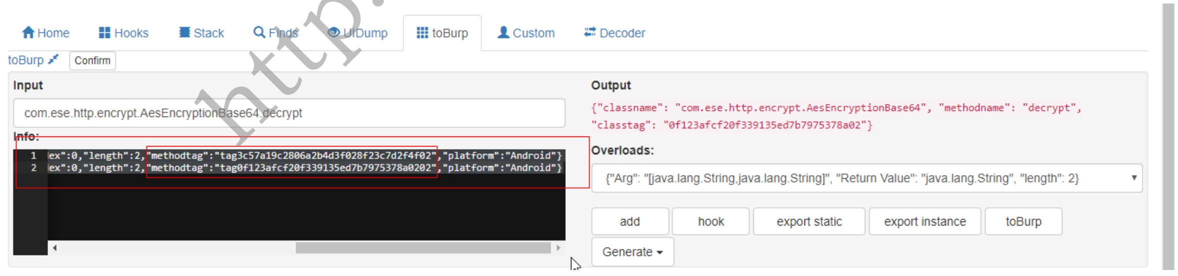
同样的方法修改解密方法。如图



```
'use strict';

var rpc_result = null;
var rpc_result_ios = null;
rpc.exports = {


tag3c57a19c2806a2b4d3f028f23c7d2f4f02: function(arg0, arg1){
        Java.perform(function () {
            try{
                var key = "9876543210123456";
                // var context = Java.use('android.app.ActivityThread').currentApplication().getApplicationContext();
                var AesEncryptionBase642c71e304c8e11a0412e647597a153909 = Java.use("com.ese.http.encrypt.AesEncryptionBase64");
                rpc_result = AesEncryptionBase642c71e304c8e11a0412e647597a153909.encrypt(key, arg0);
                // send(JSON.stringify({"aa":"bb","aa1":"bbb"})+'-cusoto0oom0sc0ri0pt-')
            }catch(e){send("tag3c57a19c2806a2b4d3f028f23c7d2f4f02, " + e + "-er00roo000r-")}
        });
        return rpc_result;
    },
// Added Function
tag0f123afcf20f339135ed7b7975378a0202: function(arg0, arg1){
        Java.perform(function () {
            try{
                var key = "9876543210123456";
                // var context = Java.use('android.app.ActivityThread').currentApplication().getApplicationContext();
                var AesEncryptionBase64cbbd3503d127380b03b238b579ee91ab = Java.use("com.ese.http.encrypt.AesEncryptionBase64");
                rpc_result = AesEncryptionBase64cbbd3503d127380b03b238b579ee91ab.decrypt(key, arg0);
                // send(JSON.stringify({"aa":"bb","aa1":"bbb"})+'-cusoto0oom0sc0ri0pt-')
            }catch(e){send("tag0f123afcf20f339135ed7b7975378a0202, " + e + "-er00roo000r-")}
        });
        return rpc_result;
    },
// Added Function


}
```

loadscript加载脚本。

这样我们可以得到两个方法名，和brida中的contextcustom一样。



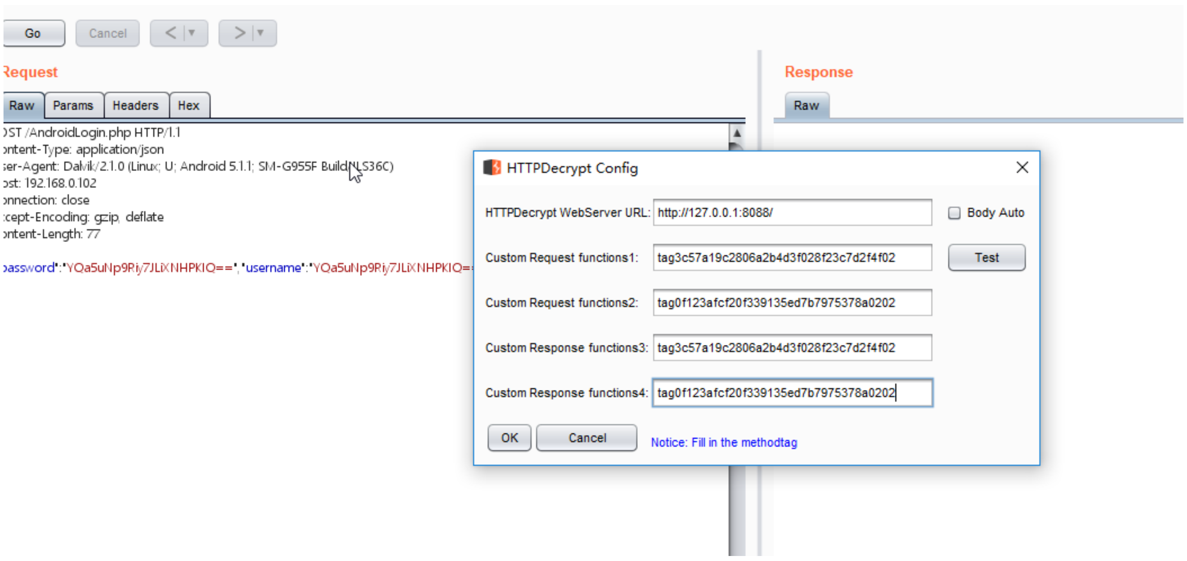
com.ese.http.encrypt.AesEncryptionBase64.decrypt

```
ex":0,"length":2,"methodtag":"tag3c57a19c2806a2b4d3f028f23c7d2f4f02","platform":"Android"}
ex":0,"length":2,"methodtag":"tag0f123afcf20f339135ed7b7975378a0202","platform":"Android"}
```

```
{"classname": "com.ese.http.encrypt.AesEncryptionBase64", "methodname": "decrypt",
"classtag": "0f123afcf20f339135ed7b7975378a02"}
```

{"Arg": "[java.lang.String,java.lang.String]", "Return Value": "java.lang.String", "length": 2}

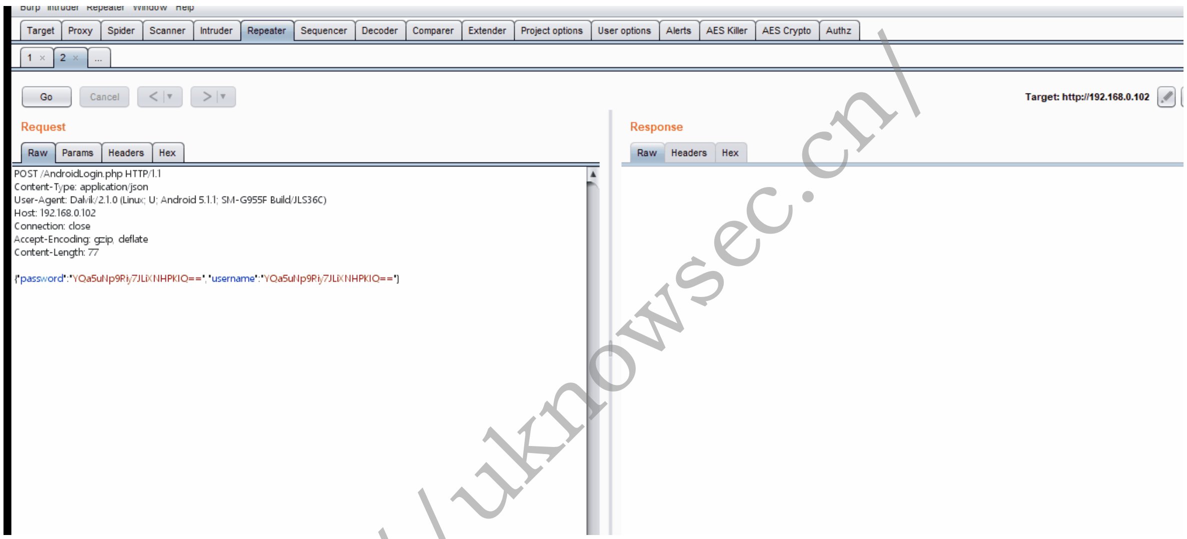
然后我们将这两个方法名添入burp插件配置项里



```
OST /AndroidLogin.php HTTP/1.1
ontent-Type: application/json
ser-Agent: Dalvik/2.1.0 (Linux; U; Android 5.1.1; SM-G955F Build/NLS36C)
ost: 192.168.0.102
onnection: close
ccept-Encoding: gzip, deflate
ontent-Length: 77

password":"YQa5uNp9Riy7JLiXNHPKIQ==","username":"YQa5uNp9Riy7JLiXNHPKIQ==
```

HTTPDecrypt WebServer URL: http://127.0.0.1:8088/ Body Auto
Custom Request functions1: tag3c57a19c2806a2b4d3f028f23c7d2f4f02
Custom Request functions2: tag0f123afcf20f339135ed7b7975378a0202
Custom Response functions3: tag3c57a19c2806a2b4d3f028f23c7d2f4f02
Custom Response functions4: tag0f123afcf20f339135ed7b7975378a0202
Notice: Fill in the methodtag

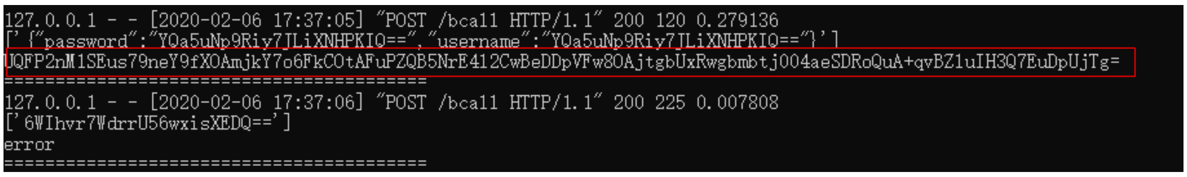
同样就可以通过右键菜单栏进行加解密了。



```
POST /AndroidLogin.php HTTP/1.1
Content-Type: application/json
User-Agent: Dalvik/2.1.0 (Linux; U; Android 5.1.1; SM-G955F Build/JLS36C)
Host: 192.168.0.102
Connection: close
Accept-Encoding: gzip, deflate
Content-Length: 77

{"password":"YQa5uNp9Riy7JLiXNHPKIQ==","username":"YQa5uNp9Riy7JLiXNHPKIQ=="}
```

他也具有自动加解密的功能，但是目前只能对整个请求包进行加解密不能匹配到需要加解密的数据。

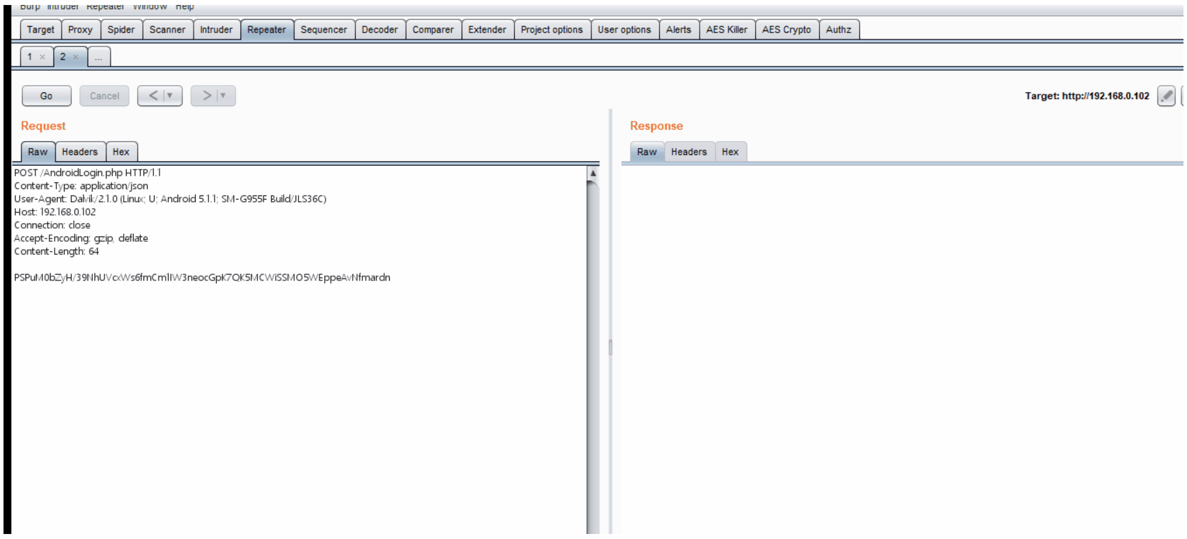
通过burp发给服务端的数据可以看到，在看起自动加解密后他把整个请求body进行了加密，跟服务端验证不符合，所以存在一定的局限性。



```
127.0.0.1 - - [2020-02-06 17:37:05] "POST /bcall HTTP/1.1" 200 120 0.279136
['{"password":"YQa5uNp9Riy7JLiXNHPKIQ==","username":"YQa5uNp9Riy7JLiXNHPKIQ=="}']
UQFP2nM1SEus79neY9fXOAmjkY7o6FkCOtAFuPZQB5NrE412CwBeDDpVFw80AjtgbUxRwgbmbtj004aeSDRoQuA+qvBZ1uIH3Q7EuDpUjTg=
========================================
127.0.0.1 - - [2020-02-06 17:37:06] "POST /bcall HTTP/1.1" 200 225 0.007808
['6WIhvr7WdrrU56wxisXEDQ==']
error
========================================
```

通过修改服务端代码，和请求包格式。

我们改成客户端把整个请求包body进行加密，给后端验证的方法。体验一下自动加解密。


Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts AES Killer AES Crypto Authz
Go
Cancel
Target: http://192.168.0.102
Request
Raw Headers Hex
Response
Raw Headers Hex

```
POST /AndroidLogin.php HTTP/1.1
Content-Type: application/json
User-Agent: Dalvik/2.1.0 (Linux; U; Android 5.1.1; SM-G955F Build/JLS36C)
Host: 192.168.0.102
Connection: close
Accept-Encoding: gzip, deflate
Content-Length: 64

PSPuM0bZyH/39NhUVcxWs6fmCmlIW3neocGpK7QK5MCWiSSMO5WEppeAvNfmardn
```

对比总结

- Brida的插桩功能好像有点不好使，lxhToolHTTPDecrypt的hook功能对APP进行多次操作后，效果还是挺好的。但定位加解密方法还是要人工的去分析，工具只是辅助作用。
- Brida和lxhToolHTTPDecrypt的右键菜单功能其实是相同的原理，绑定函数去调用函数进行加解密得到返回值
- Brida灵活性强一点，可以自定义编写插件。但是这也是比较麻烦的一点，对于不熟悉BURP插件编写的使用者是一个难点，lxhToolHTTPDecrypt方便一点，他不需要自己去写插件对数据进行处理，但是他只能对整个body进行自动加解密，存在一定局限性。而在Brida中可以在编写脚本的过程中对需要加解密的内容进行灵活的操作，lxhToolHTTPDecrypt目前还在不断的更新，可能作者后面会解决这个问题。