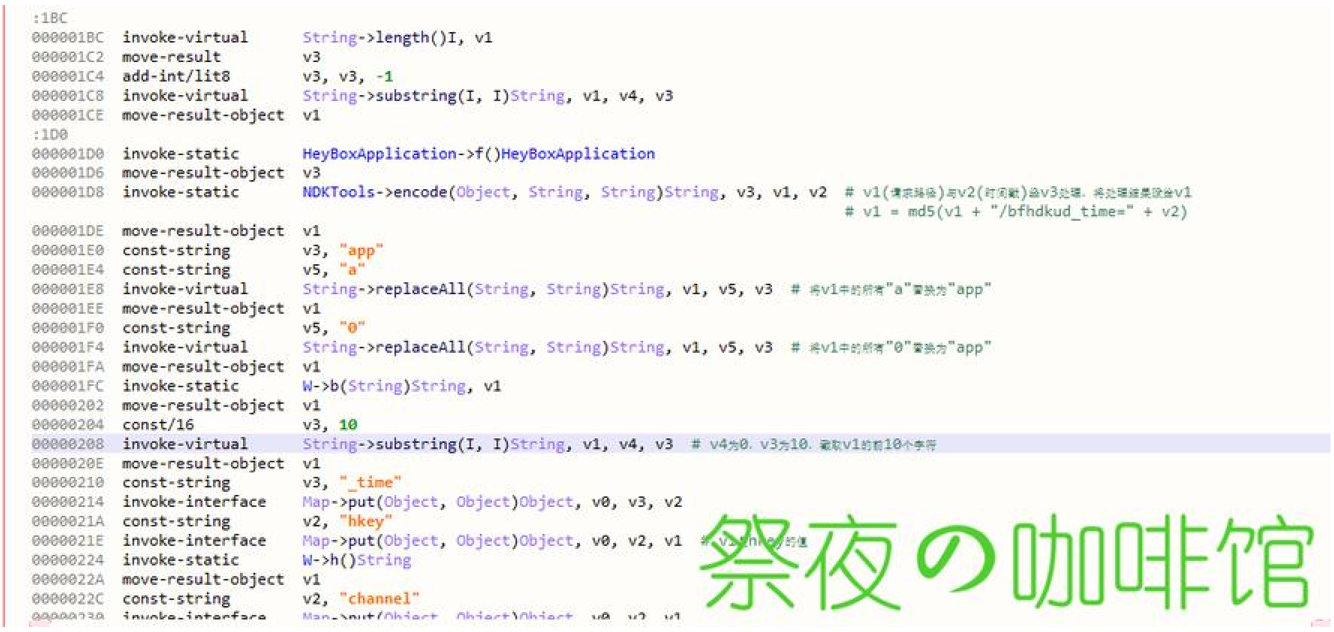
[转]Python 反反爬虫 – Frida 破解某盒子 hkey 反爬虫算法

作者：Jireh

原文链接：https://ld246.com/article/1596542896563

来源网站：链滴



```
:1BC
000001BC  invoke-virtual        String->length()I, v1
000001C2  move-result           v3
000001C4  add-int/lit8          v3, v3, -1
000001C8  invoke-virtual        String->substring(I, I)String, v1, v4, v3
000001CE  move-result-object    v1
:1D0
000001D0  invoke-static         HeyBoxApplication->f()HeyBoxApplication
000001D6  move-result-object    v3
000001D8  invoke-static         NDKTools->encode(Object, String, String)String, v3, v1, v2  # v1(请求路径)与v2(时间戳)及v3处理，将处理结果赋给v1
                                                                                             # v1 = md5(v1 + "/bfhdkud_time=" + v2)
000001DE  move-result-object    v1
000001E0  const-string          v3, "app"
000001E4  const-string          v5, "a"
000001E8  invoke-virtual        String->replaceAll(String, String)String, v1, v5, v3  # 将v1中的所有"a"替换为"app"
000001EE  move-result-object    v1
000001F0  const-string          v5, "0"
000001F4  invoke-virtual        String->replaceAll(String, String)String, v1, v5, v3  # 将v1中的所有"0"替换为"app"
000001FA  move-result-object    v1
000001FC  invoke-static         W->b(String)String, v1
00000202  move-result-object    v1
00000204  const/16              v3, 10
00000208  invoke-virtual        String->substring(I, I)String, v1, v4, v3  # v4为0. v3为10. 截取v1的前10个字符
0000020E  move-result-object    v1
00000210  const-string          v3, "_time"
00000214  invoke-interface      Map->put(Object, Object)Object, v0, v3, v2
0000021A  const-string          v2, "hkey"
0000021E  invoke-interface      Map->put(Object, Object)Object, v0, v2, v1  # v1为hkey的值
00000224  invoke-static         W->h()String
0000022A  move-result-object    v1
0000022C  const-string          v2, "channel"
00000230  invoke-interface      Map->put(Object, Object)Object, v0, v2, v1
```

[转]Python反反爬虫 – Frida破解某盒子hkey反爬虫算法

<div class="vditor-linkcard vditor-tooltipped vditor-tooltipped__n" aria-label="点击跳转到博端访问原文 https://www.jysafe.cn/4353.air">
    <a href="https://www.jysafe.cn/4353.air" class="fn__flex" target="_blank">
        <span class="vditor-linkcard__image" style="background-image: url(&quot;https://thidqq.qlogo.cn/g?b=oidb&k=fHgY8KqzfeElaJBPcAPTZQ&s=100&t=1574343121&quot;);"></span>
        <span class="vditor-linkcard__info">
            <span class="vditor-linkcard__title">
                # 祭夜の咖啡馆
</span>
<span class="vditor-linkcard__abstract">教程, 记录, Android, 猿之力</span>
<span class="vditor-linkcard__site">本博文转自 [[记录]Python反反爬虫 – Frida破解某盒子hkey爬虫算法]
</span>
</span>
</a>
</div>

前言

这盒子有个抽奖功能，但是中奖率感人~

故，hack!

用到的工具：

1.抓包软件：NetKeeper[安卓端]

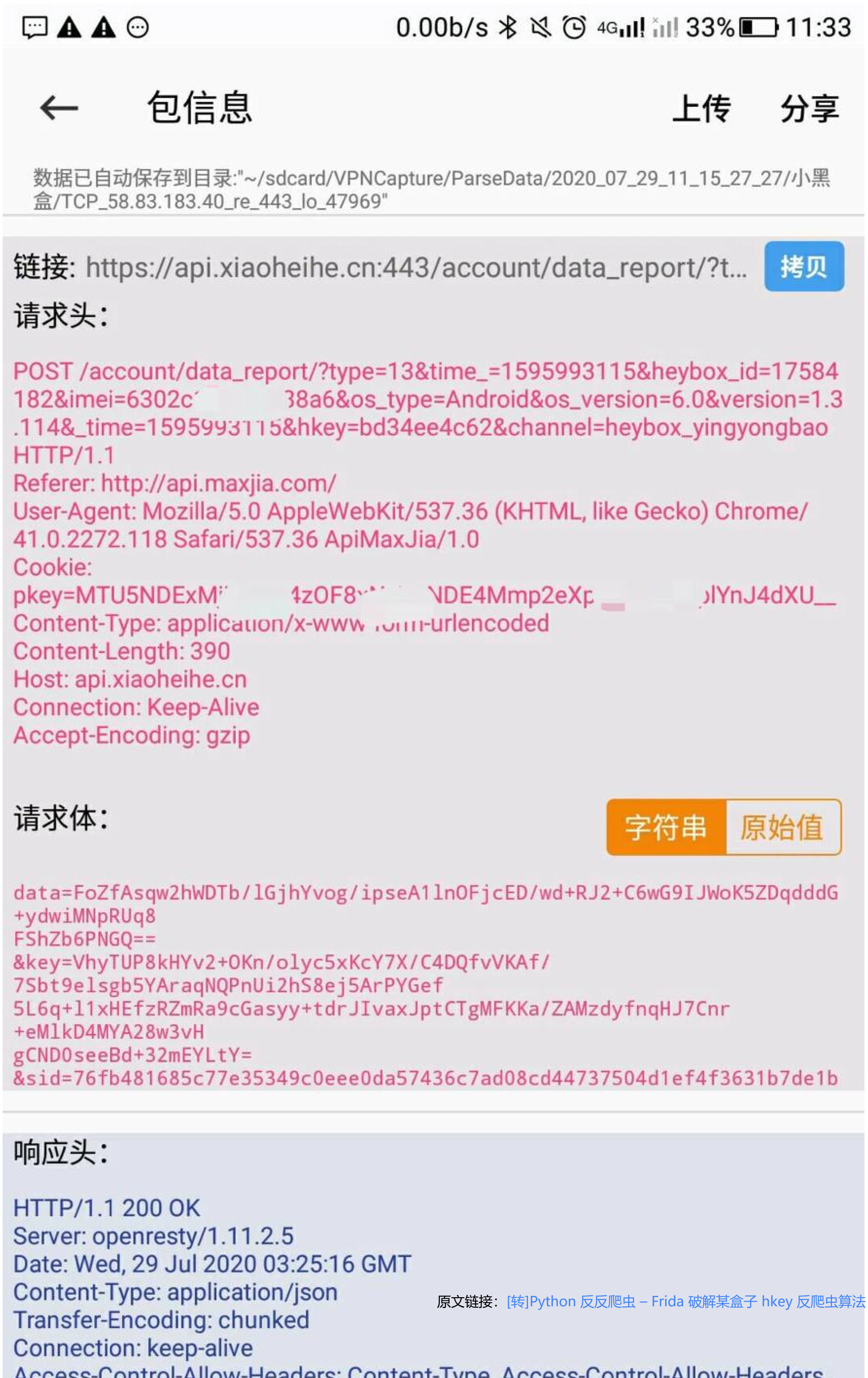
2.jeb分析工具[PC端]

3.ida分析工具[PC端]

4.Frida [Python模块以及服务端]

分析处理

先抓个包包吧



数据已自动保存到目录:"~/sdcard/VPNCapture/ParseData/2020_07_29_11_15_27_27/小黑盒/TCP_58.83.183.40_re_443_lo_47969"

链接: https://api.xiaoheihe.cn:443/account/data_report/?t...

请求头:

```
POST /account/data_report/?type=13&time_=1595993115&heybox_id=17584
182&imei=6302c[redacted]38a6&os_type=Android&os_version=6.0&version=1.3
.114&_time=1595993115&hkey=bd34ee4c62&channel=heybox_yingyongbao
HTTP/1.1
Referer: http://api.maxjia.com/
User-Agent: Mozilla/5.0 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
41.0.2272.118 Safari/537.36 ApiMaxJia/1.0
Cookie:
pkey=MTU5NDExM[redacted]4zOF8[redacted]NDE4Mmp2eXp[redacted]lYnJ4dXU__
Content-Type: application/x-www-form-urlencoded
Content-Length: 390
Host: api.xiaoheihe.cn
Connection: Keep-Alive
Accept-Encoding: gzip
```

请求体:

```
data=FoZfAsqw2hWDTb/lGjhYvog/ipseA1lnOFjcED/wd+RJ2+C6wG9IJWoK5ZDqdddG
+ydwiMNpRUq8
FShZb6PNGQ==
&key=VhyTUP8kHYv2+OKn/olyc5xKcY7X/C4DQfvVKAf/
7Sbt9elsgb5YAraqNQPnUi2hS8ej5ArPYGef
5L6q+l1xHEfzRZmRa9cGasyy+tdrJIvaxJptCTgMFKKa/ZAMzdyfnqHJ7Cnr
+eMlkD4MYA28w3vH
gCND0seeBd+32mEYLtY=
&sid=76fb481685c77e35349c0eee0da57436c7ad08cd44737504d1ef4f3631b7de1b
```

响应头:

```
HTTP/1.1 200 OK
Server: openresty/1.11.2.5
Date: Wed, 29 Jul 2020 03:25:16 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Connection: keep-alive
Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers
```

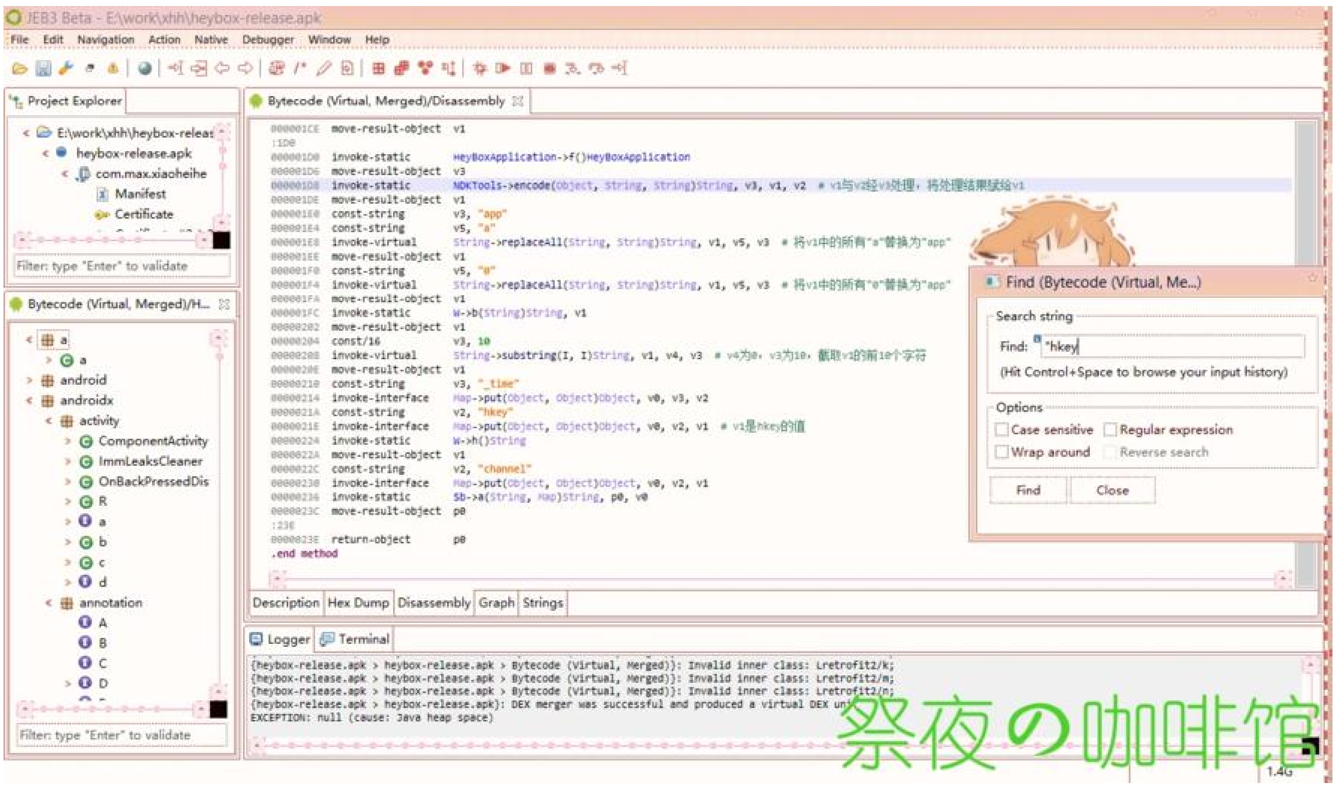
主要请求部分如下：

POST /account/data_report/?type=13&time_=1595993115&heybox_id=17584182&imei=xxxx
&os_type=Android
&os_version=6.0&version=1.3.114&_time=1595993115&hkey=bd34ee4c62
&channel=heybox_yingyongbao HTTP/1.1

经过测试hkey会随time_值变动而变动

jeb分析

搜寻hkey


JEB3 Beta - E:\work\xhh\heybox-release.apk
File Edit Navigation Action Native Debugger Window Help
Project Explorer
E:\work\xhh\heybox-releas
heybox-release.apk
com.max.xiaoheihe
Manifest
Certificate
Filter: type "Enter" to validate
Bytecode (Virtual, Merged)/H...
a
a
android
androidx
activity
ComponentActivity
ImmLeaksCleaner
OnBackPressedDis
R
a
b
c
d
annotation
A
B
C
D
Filter: type "Enter" to validate
Bytecode (Virtual, Merged)/Disassembly
Description Hex Dump Disassembly Graph Strings
Logger Terminal
Find (Bytecode (Virtual, Me...)
Search string
Find: "hkey
(Hit Control+Space to browse your input history)
Options
Case sensitive
Regular expression
Wrap around
Reverse search
Find
Close
祭夜の咖啡馆
1.4G

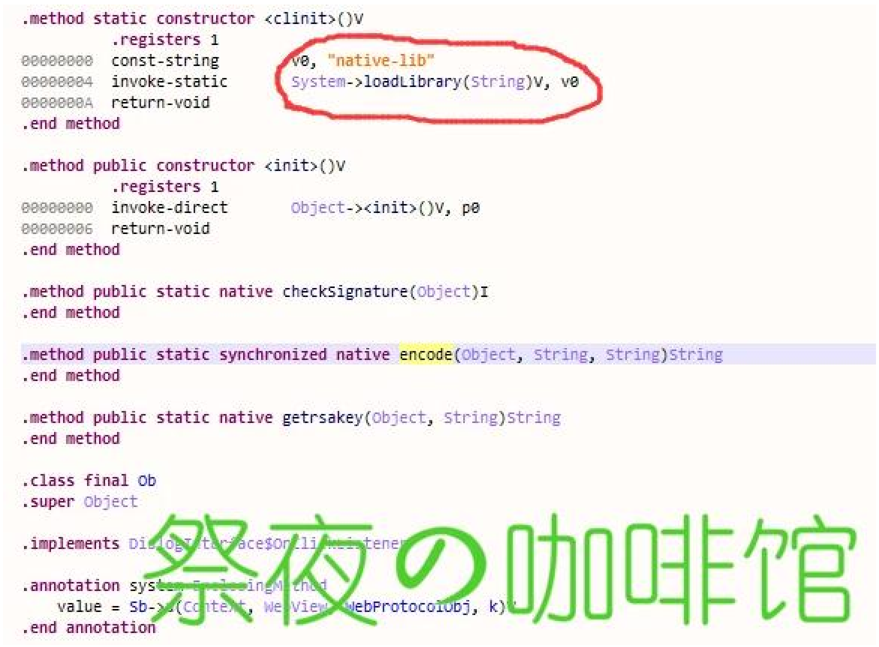
```
000001CE  move-result-object  v1
:1D0
000001D0  invoke-static       HeyBoxApplication->f()HeyBoxApplication
000001D6  move-result-object  v3
000001D8  invoke-static       NDKTools->encode(Object, String, String)String, v3, v1, v2  # v1与v2经v3处理，将处理结果赋给v1
000001DE  move-result-object  v1
000001E0  const-string        v3, "app"
000001E4  const-string        v5, "a"
000001E8  invoke-virtual      String->replaceAll(String, String)String, v1, v5, v3  # 将v1中的所有"a"替换为"app"
000001EE  move-result-object  v1
000001F0  const-string        v5, "0"
000001F4  invoke-virtual      String->replaceAll(String, String)String, v1, v5, v3  # 将v1中的所有"0"替换为"app"
000001FA  move-result-object  v1
000001FC  invoke-static       W->b(String)String, v1
00000202  move-result-object  v1
00000204  const/16            v3, 10
00000208  invoke-virtual      String->substring(I, I)String, v1, v4, v3  # v4为0, v3为10, 截取v1的前10个字符
0000020E  move-result-object  v1
00000210  const-string        v3, "_time"
00000214  invoke-interface    Map->put(Object, Object)Object, v0, v3, v2
0000021A  const-string        v2, "hkey"
0000021E  invoke-interface    Map->put(Object, Object)Object, v0, v2, v1  # v1是hkey的值
00000224  invoke-static       W->h()String
0000022A  move-result-object  v1
0000022C  const-string        v2, "channel"
00000230  invoke-interface    Map->put(Object, Object)Object, v0, v2, v1
00000236  invoke-static       Sb->a(String, Map)String, p0, v0
0000023C  move-result-object  p0
:23E
0000023E  return-object       p0
.end method
```

```
{heybox-release.apk > heybox-release.apk > Bytecode (Virtual, Merged)}: Invalid inner class: Lretrofit2/k;
{heybox-release.apk > heybox-release.apk > Bytecode (Virtual, Merged)}: Invalid inner class: Lretrofit2/m;
{heybox-release.apk > heybox-release.apk > Bytecode (Virtual, Merged)}: Invalid inner class: Lretrofit2/n;
{heybox-release.apk > heybox-release.apk}: DEX merger was successful and produced a virtual DEX un
EXCEPTION: null (cause: Java heap space)
```

v1与v2的值来源似乎不好处理，

为了减少脑细胞与头发的消耗，可以尝试从NDKTools->encode方法入手。

但是，到达encode位置后（Ctrl+双击），

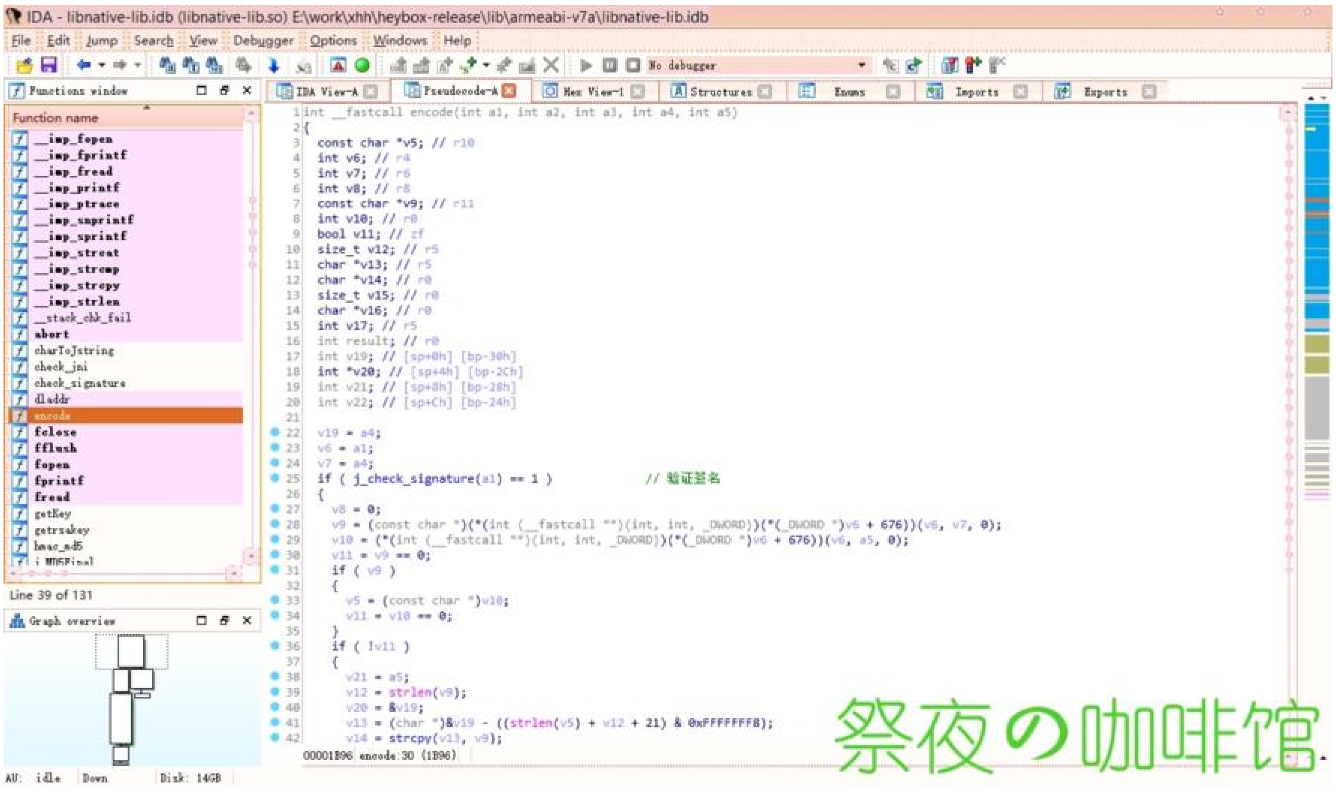


```
.method static constructor <clinit>()V
          .registers 1
00000000  const-string        v0, "native-lib"
00000004  invoke-static       System->loadLibrary(String)V, v0
0000000A  return-void
.end method

.method public constructor <init>()V
          .registers 1
00000000  invoke-direct       Object-><init>()V, p0
00000006  return-void
.end method

.method public static native checkSignature(Object)I
.end method

.method public static synchronized native encode(Object, String, String)String
.end method

.method public static native getrsakey(Object, String)String
.end method

.class final Ob
.super Object

.implements DialogInterface$OnClickListener

.annotation system-ReleasingMethod
    value = Sb->a(Context, WebView, WebProtocolObj, k)V
.end annotation
```

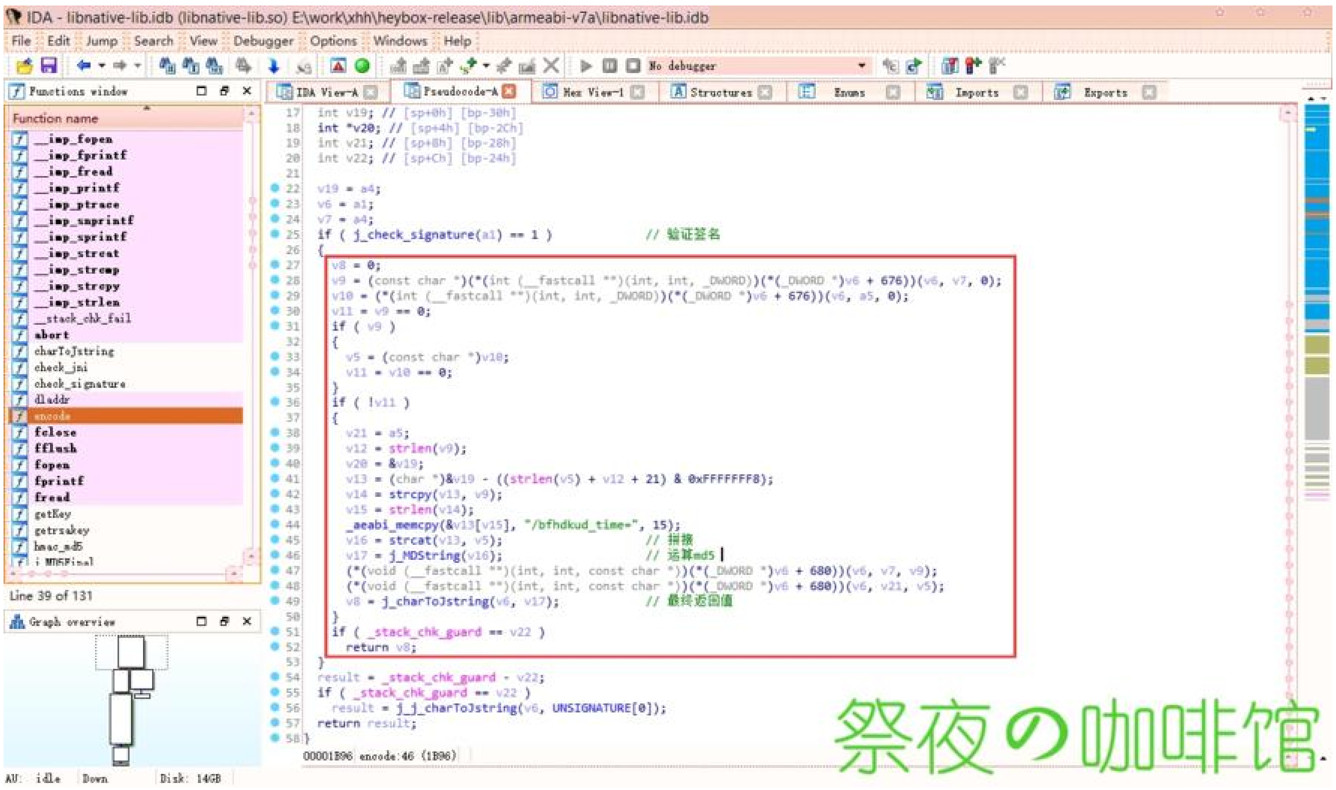
啊，这得上ida了

ida分析

进入ida的encode中，



```
int __fastcall encode(int a1, int a2, int a3, int a4, int a5)
{
  const char *v5; // r10
  int v6; // r4
  int v7; // r6
  int v8; // r8
  const char *v9; // r11
  int v10; // r0
  bool v11; // zf
  size_t v12; // r5
  char *v13; // r5
  char *v14; // r0
  size_t v15; // r0
  char *v16; // r0
  int v17; // r5
  int result; // r0
  int v19; // [sp+0h] [bp-30h]
  int *v20; // [sp+4h] [bp-2Ch]
  int v21; // [sp+8h] [bp-28h]
  int v22; // [sp+Ch] [bp-24h]

  v19 = a4;
  v6 = a1;
  v7 = a4;
  if ( j_check_signature(a1) == 1 )                    // 验证签名
  {
    v8 = 0;
    v9 = (const char *)(*(int (__fastcall **)(int, int, _DWORD))(*(_DWORD *)v6 + 676))(v6, v7, 0);
    v10 = (*(int (__fastcall **)(int, int, _DWORD))(*(_DWORD *)v6 + 676))(v6, a5, 0);
    v11 = v9 == 0;
    if ( v9 )
    {
      v5 = (const char *)v10;
      v11 = v10 == 0;
    }
    if ( !v11 )
    {
      v21 = a5;
      v12 = strlen(v9);
      v20 = &v19;
      v13 = (char *)&v19 - ((strlen(v5) + v12 + 21) & 0xFFFFFFF8);
      v14 = strcpy(v13, v9);
00001B96 encode:30 (1B96)
```

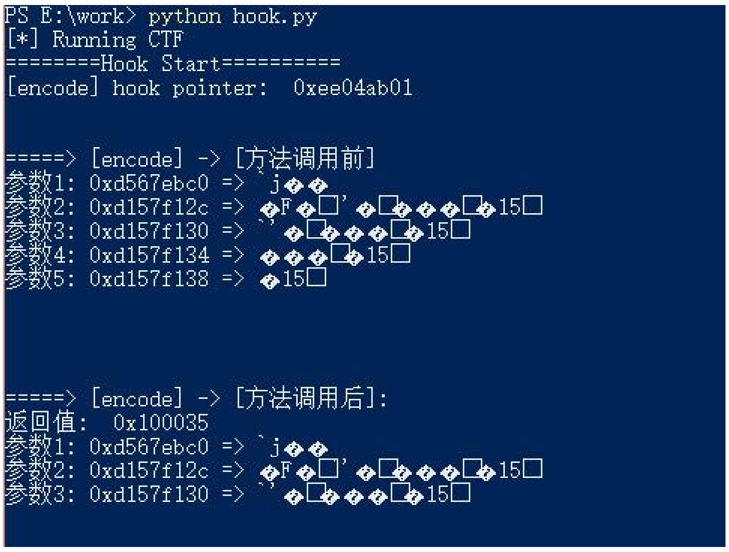


```
  int v19; // [sp+0h] [bp-30h]
  int *v20; // [sp+4h] [bp-2Ch]
  int v21; // [sp+8h] [bp-28h]
  int v22; // [sp+Ch] [bp-24h]

  v19 = a4;
  v6 = a1;
  v7 = a4;
  if ( j_check_signature(a1) == 1 )                 // 验证签名
  {
    v8 = 0;
    v9 = (const char *)(*(int (__fastcall **)(int, int, _DWORD))(*(_DWORD *)v6 + 676))(v6, v7, 0);
    v10 = (*(int (__fastcall **)(int, int, _DWORD))(*(_DWORD *)v6 + 676))(v6, a5, 0);
    v11 = v9 == 0;
    if ( v9 )
    {
      v5 = (const char *)v10;
      v11 = v10 == 0;
    }
    if ( !v11 )
    {
      v21 = a5;
      v12 = strlen(v9);
      v20 = &v19;
      v13 = (char *)&v19 - ((strlen(v5) + v12 + 21) & 0xFFFFFFF8);
      v14 = strcpy(v13, v9);
      v15 = strlen(v14);
      _aeabi_memcpy(&v13[v15], "/bfhdkud_time=", 15);
      v16 = strcat(v13, v5);                        // 拼接
      v17 = j_MDString(v16);                        // 运算md5
      (*(void (__fastcall **)(int, int, const char *))(*(_DWORD *)v6 + 680))(v6, v7, v9);
      (*(void (__fastcall **)(int, int, const char *))(*(_DWORD *)v6 + 680))(v6, v21, v5);
      v8 = j_charToJstring(v6, v17);                // 最终返回值
    }
    if ( _stack_chk_guard == v22 )
      return v8;
  }
  result = _stack_chk_guard - v22;
  if ( _stack_chk_guard == v22 )
    result = j_j_charToJstring(v6, UNSIGNATURE[0]);
  return result;
}
00001B96 encode:46 (1B96)
```

看似没什么大问题，来hook：

```
console.log("========Hook Start==========")

String.prototype.format = function () {
    var values = arguments;
    return this.replace(/\{(\d+)\}/g, function (match, index) {
        if (values.length > index) {
            return values[index];
        } else {
            return "";
        }
    });
}

var JNI_LOAD_POINTER = Module.getExportByName('libnative-lib.so', 'JNI_OnLoad'); // 首先到 JNI_OnLoad方法的地址
var BASE_ADDR = parseInt(JNI_LOAD_POINTER) - parseInt('0x1C6C'); // 用程序运行中JNI_OnLod的绝对地址减去它的相对地址得到基址


// encode
Java.perform(function() {
    var hookpointer = '0x' + parseInt(BASE_ADDR + parseInt('0x1B00')).toString(16) // 获取要hook方法的地址
    var pointer = new NativePointer(hookpointer) // 根据方法地址构建NativePointer
    console.log('[encode] hook pointer: ', pointer)

    var arg0, arg1, arg2, arg3
    Interceptor.attach(pointer, {
        onEnter: function(args) {
            arg0 = args[0]
            arg1 = args[1]
            arg2 = args[2]
            arg3 = args[3]
            console.log('\n')
            console.log('=====> [encode] -> [方法调用前]')
            console.log('参数1: {0} => {1}'.format(arg0, Memory.readCString(arg0)))
            console.log('参数2: {0} => {1}'.format(arg1, Memory.readCString(arg1)))
            console.log('参数3: {0} => {1}'.format(arg2, Memory.readCString(arg2)))
            console.log('参数4: {0} => {1}'.format(arg3, Memory.readCString(arg3)))
            console.log('参数5: {0} => {1}'.format(args[4], Memory.readCString(args[4])))
            console.log('\n')
        },
        onLeave: function(retval) {
            console.log('\n')
            console.log('=====> [encode] -> [方法调用后]:')
            console.log('返回值: ', retval)
            console.log('参数1: {0} => {1}'.format(retval, Memory.readCString(retval)))
            console.log('\n')
        }
    }
  )
})
```

但是，还是出了些问题：



```
PS E:\work> python hook.py
[*] Running CTF
========Hook Start==========
[encode] hook pointer:  0xee04ab01


=====> [encode] -> [方法调用前]
参数1: 0xd567ebc0 => `j��
参数2: 0xd157f12c => �F�□'�□����□�15□
参数3: 0xd157f130 => `'�□����□�15□
参数4: 0xd157f134 => ���□�15□
参数5: 0xd157f138 => �15□



=====> [encode] -> [方法调用后]:
返回值:  0x100035
参数1: 0xd567ebc0 => `j��
参数2: 0xd157f12c => �F�□'�□����□�15□
参数3: 0xd157f130 => `'�□����□�15□
```

额，我不觉得肉眼能看出这是什么东西

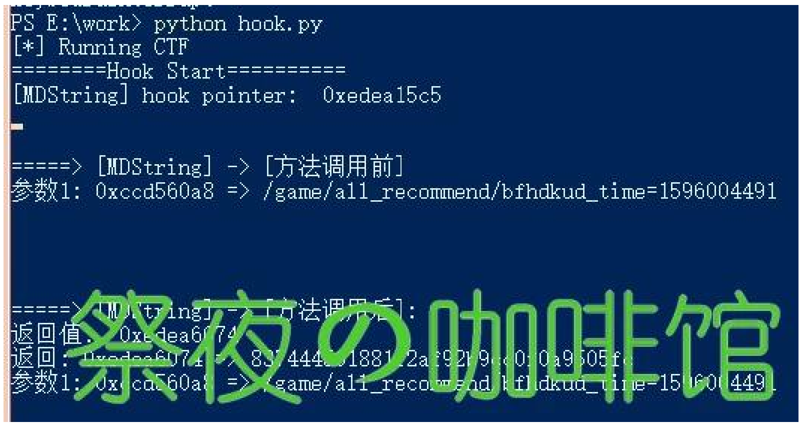
既然程序处理了一些不能看的东西，那就尝试去找出能看的东西吧
xpressionless

我觉得选择MDString比较好，因为可以看到什么东西被拿去算md5了。

hook MDString：

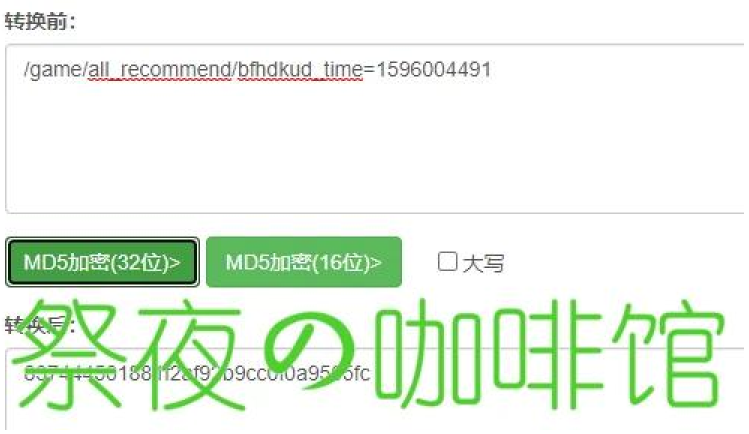
```
console.log("========Hook Start==========")

String.prototype.format = function () {
    var values = arguments;
    return this.replace(/\{(\d+)\}/g, function (match, index) {
        if (values.length > index) {
            return values[index];
        } else {
            return "";
        }
    });
}

var JNI_LOAD_POINTER = Module.getExportByName('libnative-lib.so', 'JNI_OnLoad'); // 首先
到 JNI_OnLoad方法的地址
var BASE_ADDR = parseInt(JNI_LOAD_POINTER) - parseInt('0x1C6C'); // 用程序运行中JNI_OnLo
d的绝对地址减去它的相对地址得到基址

// MDString
Java.perform(function() {
    var hookpointer = '0x' + parseInt(BASE_ADDR + parseInt('0x15C4')).toString(16) // 获取要h
ok方法的地址
    var pointer = new NativePointer(hookpointer) // 根据方法地址构建NativePointer
    console.log('[MDString] hook pointer: ', pointer)

    var arg0, arg1, arg2, arg3
    Interceptor.attach(pointer, {
            onEnter: function(args) {
                arg0 = args[0]
                arg1 = args[1]
                arg2 = args[2]
                console.log('\n')
                console.log('=====> [MDString] -> [方法调用前]')
                console.log('参数1: {0} => {1}'.format(arg0, Memory.readCString(arg0)))
                console.log('\n')
            },
            onLeave: function(retval) {
                console.log('\n')
                console.log('=====> [MDString] -> [方法调用后]:')
                console.log('返回值: ', retval)
                console.log('返回: {0} => {1}'.format(retval, Memory.readCString(retval)))
                console.log('参数1: {0} => {1}'.format(arg0, Memory.readCString(arg0)))
                console.log('\n')
            }
        }
    )
})
```

输出：



```
PS E:\work> python hook.py
[*] Running CTF
========Hook Start=========
[MDString] hook pointer:  0xedea15c5

=====> [MDString] -> [方法调用前]
参数1: 0xccd560a8 => /game/all_recommend/bfhdkud_time=1596004491


=====> [MDString] -> [方法调用后]:
返回值: 0xedea6074
返回: 0xedea6074 => 837444501881f2a92b9cc0f0a9505fc
参数1: 0xccd560a8 => /game/all_recommend/bfhdkud_time=1596004491
```

可以发现，返回值正是将/game/all_recommend/bfhdkud_time=1596004491进行MD5加密：



转换前：
/game/all_recommend/bfhdkud_time=1596004491
MD5加密(32位)>
MD5加密(16位)>
☐大写
转换后：
837444501881f2a92b9cc0f0a9505fc

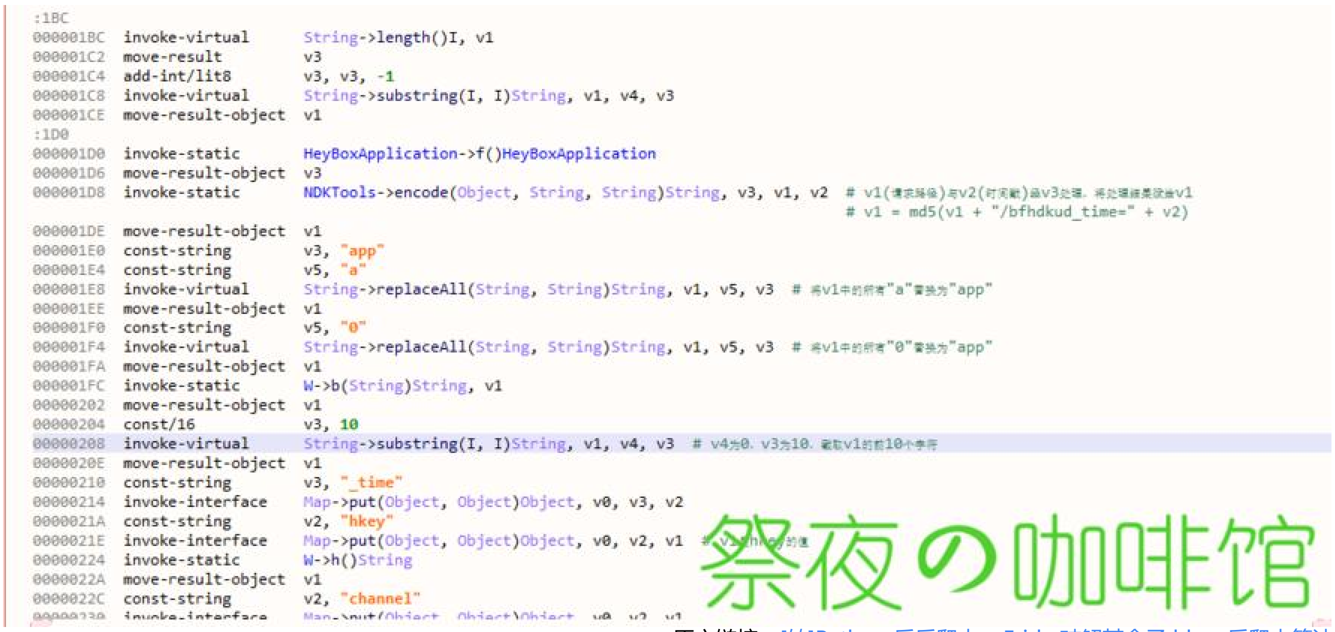
结合抓包到的请求，可以得到NDKTOOL->encode的原理：

由路径/game/all_recommend与时间戳1596004491以及/bfhdkud_time=

拼接成/game/all_recommend/bfhdkud_time=1596004491算出32位 小写md5 837444501881f2a92b9cc0f0a9505fc

结论

hkey的处理流程:



```
:1BC
000001BC  invoke-virtual        String->length()I, v1
000001C2  move-result           v3
000001C4  add-int/lit8          v3, v3, -1
000001C8  invoke-virtual        String->substring(I, I)String, v1, v4, v3
000001CE  move-result-object    v1
:1D0
000001D0  invoke-static         HeyBoxApplication->f()HeyBoxApplication
000001D6  move-result-object    v3
000001D8  invoke-static         NDKTools->encode(Object, String, String)String, v3, v1, v2  # v1(请求路径)与v2(时间戳)和v3处理，将处理结果保存v1
                                                                                            # v1 = md5(v1 + "/bfhdkud_time=" + v2)
000001DE  move-result-object    v1
000001E0  const-string          v3, "app"
000001E4  const-string          v5, "a"
000001E8  invoke-virtual        String->replaceAll(String, String)String, v1, v5, v3  # 将v1中的所有"a"替换为"app"
000001EE  move-result-object    v1
000001F0  const-string          v5, "0"
000001F4  invoke-virtual        String->replaceAll(String, String)String, v1, v5, v3  # 将v1中的所有"0"替换为"app"
000001FA  move-result-object    v1
000001FC  invoke-static         W->b(String)String, v1
00000202  move-result-object    v1
00000204  const/16              v3, 10
00000208  invoke-virtual        String->substring(I, I)String, v1, v4, v3  # v4为0, v3为10, 截取v1的前10个字符
0000020E  move-result-object    v1
00000210  const-string          v3, "_time"
00000214  invoke-interface      Map->put(Object, Object)Object, v0, v3, v2
0000021A  const-string          v2, "hkey"
0000021E  invoke-interface      Map->put(Object, Object)Object, v0, v2, v1  # v1为hkey的值
00000224  invoke-static         W->h()String
0000022A  move-result-object    v1
0000022C  const-string          v2, "channel"
00000230  invoke-interface      Map->put(Object, Object)Object, v0, v2, v1
```

原文链接：[转]Python 反反爬虫 – Frida 破解某盒子 hkey 反爬虫算法

注：Hook代码编写，参考：Python反反爬虫 – Frida破解某安卓社区token反爬虫

原文链接：[转]Python 反反爬虫 – Frida 破解某盒子 hkey 反爬虫算法