

原文地址:<http://drops.wooyun.org/tips/9471>

0x00 前言

Android应用的加固和对抗不断升级，单纯的静态加固效果已无法满足需求，所以出现了隐藏方法加固，运行时动态恢复和反调试等方法来对抗，本文通过实例来分析有哪些对抗和反调试手段。

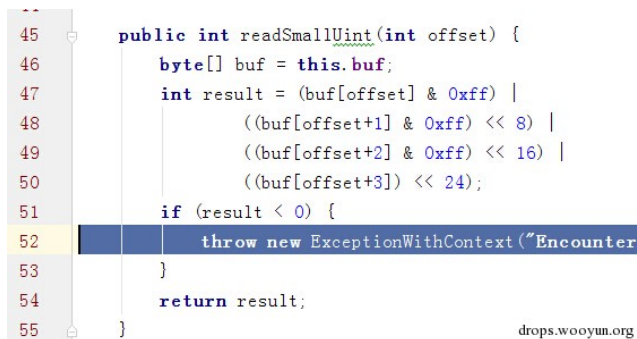
0x01 对抗反编译

首先使用apktool进行反编译，发现该应用使用的加固方式会让apktool卡死，通过调试apktool源码（如何调试apktool可参见前文《Android应用资源文件格式解析与保护对抗研究》），发现解析时抛出异常，如下图：



```
Run RunApkTool
Error occurred while disassembling class Lcom.tencent.exmobiwin.spot.b; - skipping class
org.jf.util.ExceptionWithContext: Encountered small uint that is out of range at offset 0x597a8
    at org.jf.dexlib2.dexbacked.BaseDexBuffer.readSmallUInt (BaseDexBuffer.java:52)
    at org.jf.dexlib2.dexbacked.DexBackedMethodImplementation.getDebugInfo (DexBackedMethodImplementation.java:126)
    at org.jf.dexlib2.dexbacked.DexBackedMethodImplementation.getDebugItems (DexBackedMethodImplementation.java:131)
    at org.jf.baksmali.Adaptors.MethodDefinition.addDebugInfo (MethodDefinition.java:575)
    at org.jf.baksmali.Adaptors.MethodDefinition.getMethodItems (MethodDefinition.java:377)
    at org.jf.baksmali.Adaptors.MethodDefinition.writeTo (MethodDefinition.java:238)
    at org.jf.baksmali.Adaptors.ClassDefinition.writeDirectMethods (ClassDefinition.java:283)
    at org.jf.baksmali.Adaptors.ClassDefinition.writeTo (ClassDefinition.java:112)
    at org.jf.baksmali.baksmali.disassembleClass (baksmali.java:226)
    at org.jf.baksmali.baksmali.access$000 (baksmali.java:56)
    at org.jf.baksmali.baksmali$1.call (baksmali.java:150)
    at org.jf.baksmali.baksmali$1.call (baksmali.java:148)
    at java.util.concurrent.FutureTask.run (FutureTask.java:286)
    drops.wooyun.org
```

根据异常信息可知是readSmallUInt出错，调用者是getDebugInfo，查看源码如下：



```
45 public int readSmallUInt(int offset) {
46     byte[] buf = this.buf;
47     int result = (buf[offset] & 0xff) |
48         ((buf[offset+1] & 0xff) << 8) |
49         ((buf[offset+2] & 0xff) << 16) |
50         ((buf[offset+3] << 24);
51     if (result < 0) {
52         throw new ExceptionWithContext("Encounter
53     }
54     return result;
55 }
```

可见其在计算该偏移处的uleb值时得到的结果小于0，从而抛出异常。在前文《Android程序的反编译对抗研究》中介绍了DEX的文件格式，其中提到与DebugInfo相关的字段为DexCode结构的debugInfoOff字段。猜测应该是在此处做了手脚，在010editor中打开dex文件，运行模板DEXTemplate.bt，找到debugInfoOff字段。果然，该值被设置为了0xFEEEEEEE。

2:9010h:	07 00 00 00 04 00 01 00 EE EE EE FE 7B 00 00 00 i i i p { . . .
2:9020h:	1A 00 98 03 71 10 13 07 00 00 13 01 00 01 23 10 q # .
2:9030h:	4B 02 69 00 01 00 23 10 38 02 69 00 02 00 22 00	K . i . . . # . 8 . i . . . " .
2:9040h:	06 00 70 10 01 00 00 00 6E 10 CC 0A 00 00 0C 00	. . p n . i
2:9050h:	22 01 D5 01 70 10 FD 0A 01 00 13 02 49 00 6E 20	* . Ö . p . ý l . n
2:9060h:	00 0B 21 00 0C 01 13 02 2E 00 6E 20 00 0B 21 00	. . ! n . . ! .
2:9070h:	0C 01 13 02 67 00 6E 20 00 0B 21 00 0C 01 13 02 g . n . . !
2:9080h:	69 00 6E 20 00 0B 21 00 0C 01 13 02 66 00 6E 20	i . n . . ! f . n

Template Results - DEXTemplate.bt	
Name	Value
uint source_file_idx	NO_INDEX
uint annotations_off	0h
uint class_data_off	5A145h
struct class_data_item class_data	3 static fields, 0 instance fields, 3 direct methods, 0 virtual methods
▷ struct uleb128 static_fields_size	0x3
▷ struct uleb128 instance_fields_size	0x0
▷ struct uleb128 direct_methods_size	0x3
▷ struct uleb128 virtual_methods_size	0x0
▷ struct encoded_field_list static_fields	3 fields
▷ struct encoded_method_list direct_methods	3 methods
▷ struct encoded_method method[0]	static constructor void I.I.<clinit>()
▷ struct uleb128 method_idx_diff	0x0
▷ struct uleb128 access_flags	(0x10008) ACC_STATIC ACC_CONSTRUCTOR
▷ struct uleb128 code_off	0x29010
▷ struct code_item code	7 registers, 0 in arguments, 4 out arguments, 1 tries, 123 instructions
ushort registers_size	7h
ushort ins_size	0h
ushort outs_size	4h
ushort tries_size	1h
uint debug_info_off	FFFFFFFFh
uint insns_size	7Bh
▷ ushort insns[123]	
ushort padding	0h
▷ struct try_item tries	
▷ struct encoded_catch_handler_list handlers	1 handler lists

接下来修复就比较简单了，由于debugInfoOff一般情况下是无关紧要的字段，所以只要关闭异常就行了。

为了保险起见，在readSmallUint方法后面添加一个新方法readSmallUint_DebugInfo，复制readSmallUint的代码，if语句内result赋值为0并注释掉抛异常代码。

```

56 public int readSmallUint_DebugInfo(int offset) {
57     byte[] buf = this.buf;
58     int result = (buf[offset] & 0xff) |
59         ((buf[offset+1] & 0xff) << 8) |
60         ((buf[offset+2] & 0xff) << 16) |
61         ((buf[offset+3] << 24);
62     if (result < 0) {
63         result = 0;
64         //throw new ExceptionWithContext("Encountered small
65     }
66     return result;
67 }
```

然后在getDebugInfo中调用readSmallUint_DebugInfo即可。

```

125 private DebugInfo getDebugInfo() {
126     return DebugInfo.newOrEmpty(dexFile, dexFile.readSmallUint_DebugInfo(codeOffset + CodeItem.DEBUG_
127     //return DebugInfo.newOrEmpty(dexFile, dexFile.readSmallUint(codeOffset + CodeItem.DEBUG_
128 }
```

重新编译apktool，对apk进行反编译，一切正常。

然而以上只是开胃菜，虽然apktool可以正常反编译了，但查看反编译后的smali代码，发现所有的虚方法都是native方法，而且类的初始化方法中开头多了2行代码，如下图：

```

6      # direct methods
7      .method static constructor <clinit>()V
8      .locals 1
9
10     const-string v0, "aHcuaGVsbG93b3JsZC5NYWluQWN0aXZpdHk="
11
12     invoke-static {v0}, Lcom.wooyun.ProxyApplication;->init(Ljava/lang/String;)V
13
14     return-void
15 .end method
16
17 .method public constructor <init>()V
18 .locals 0
19
20     invoke-direct {p0}, Landroid/app/Activity;-><init>()V
21
22     return-void
23 .end method
24
25 # virtual methods
26 >>> .method public final native finaltest()Z
27 .end method
28
29 >>> .method protected native onCreate(Landroid/os/Bundle;)V
30 .end method
31
32 >>> .method public native onCreateOptionsMenu(Landroid/view/Menu;)Z
33 .end method
34
35 >>> .method public native onOptionsItemSelected(Landroid/view/MenuItem;)Z
36 .end method
37
38

```

drops.wooyun.org

其基本原理是在dex文件中隐藏虚方法，运行后在第一次加载类时通过在方法（如果没有方法，则会自动添加该方法）中调用ProxyApplication的init方法来恢复被隐藏的虚方法，其中字符串"aHcuaGVsbG93b3JsZC5NYWluQWN0aXZpdHk="是当前类名的base64编码。

ProxyApplication类只有2个方法，clinit和init，clinit主要是判断系统版本和架构，加载指定版本的so保护模块（X86或ARM）；而init方法也是native方法，调用时直接进入了so模块。

```

ProxyApplication.class x
package com.

import android.os.Build;
import android.os.Build.VERSION;
import android.util.Log;

public class ProxyApplication
{
    static
    {
        if ((Build.VERSION.SDK_INT == 21) && (Build.MANUFACTURER.equals("asus")))
        {
            System.loadLibrary("-x86");
            return;
        }
        if (Build.CPU_ABI.contains("x86"))
        {
            System.loadLibrary("-x86");
            return;
        }
        if (System.getProperty("os.arch").contains("arm"))
        {
            System.loadLibrary("");
            return;
        }
        Log.e(
            System.loadLibrary("");
        );
    }

    public static native void init(String paramString);
}

```

drops.wooyun.org

那么它是如何恢复被隐藏的方法的呢？这就要深入SO模块内部一探究竟了。

0x02 动态调试so模块

如何使用IDA调试android的SO模块，网上有很多教程，这里简单说明一下。

1. 准备工作

1.1准备好模拟器并安装目标APP。

1.2 将IDA\dbgsrv\目录下的android_server复制到模拟器里，并赋予可执行权限。

```
adb push d:\IDA\dbgsrv\android_server /data/data/sv
adb shell chmod 755 /data/data/sv
```

1.3 运行android_server，默认监听23946端口。

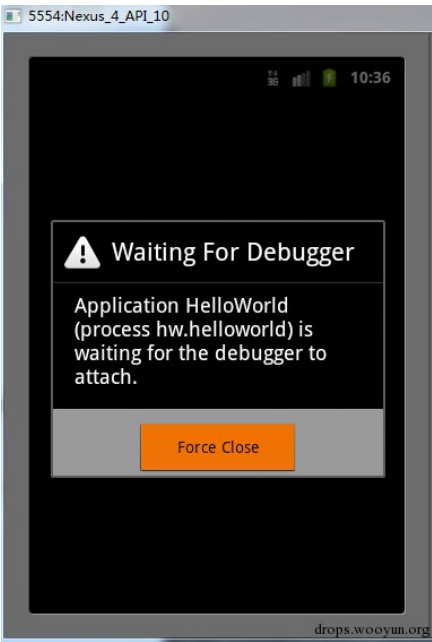
```
adb shell /data/data/sv
```

1.4 端口转发。

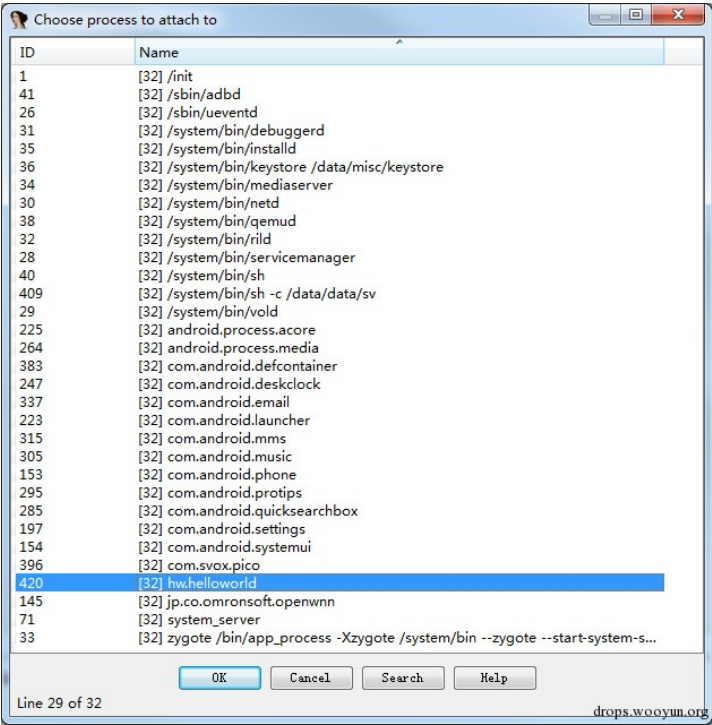
```
adb forward tcp:23946 tcp:23946
```

2 以调试模式启动APP，模拟器将出现等待调试器的对话框。

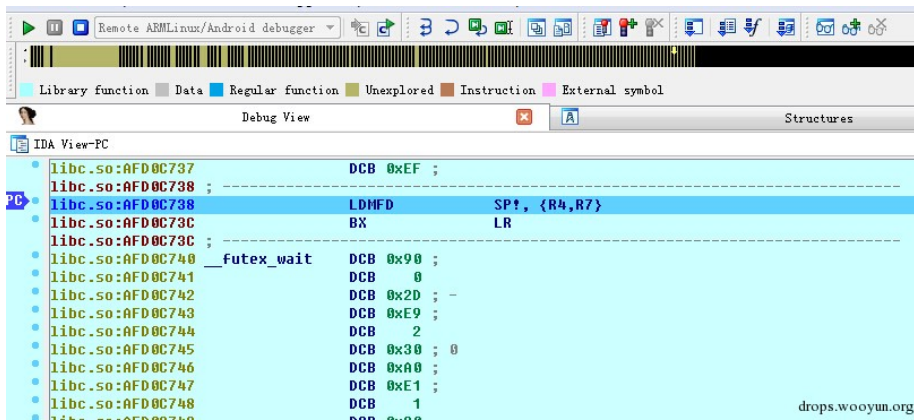
```
adb shell am start -D -n hw.helloworld/hw.helloworld.MainActivity
```



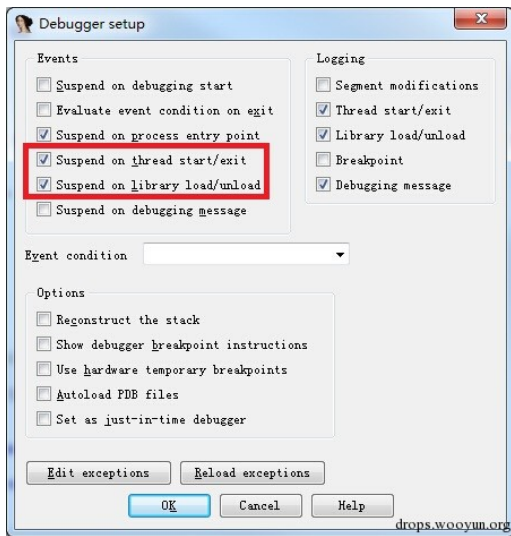
3 启动IDA，打开debugger->attach->remote Armlinux/andoid debugger，设置hostname为localhost，port为23946，点击OK；然后选择要调试的APP并点击OK。



这时，正常状态下会断下来：



然后设置在模块加载时中断:

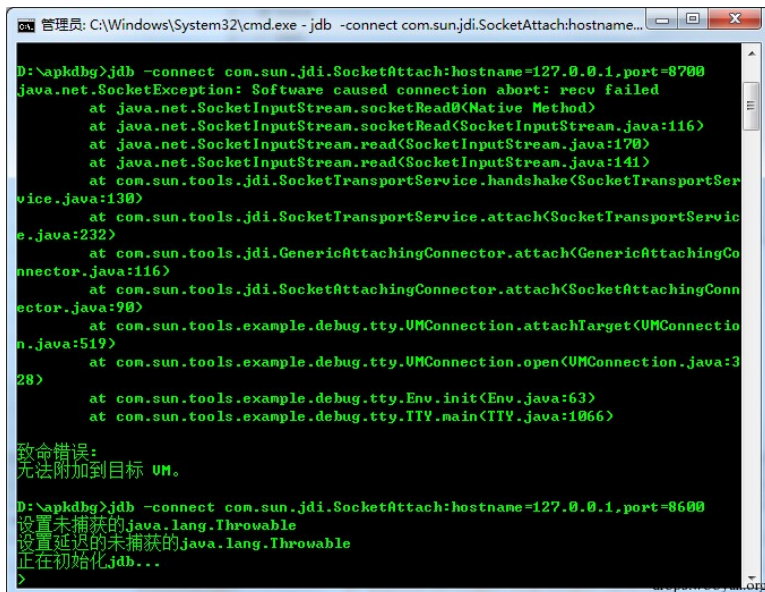


点击OK, 按F9运行。

然后打开DDMS并执行以下命令, 模拟器就会自动断下来:

```
jdb -connect com.sun.jdi.SocketAttach:hostname=127.0.0.1,port=8700
```

(如果出现如下无法附加到目标VM的错误, 可尝试端口8600)



此时, 可在IDA中正常下断点调试, 这里我们断JNI_OnLoad和init函数。


```

.so:80407A18
.so:80407A18 ; ===== SUBROUTINE =====
.so:80407A18
.so:80407A18 JNI_OnLoad
.so:80407A18
.so:80407A18 var_14 = -0x14
.so:80407A18
.so:80407A18 PUSH {R0-R2,R4,R5,LR}
.so:80407A1A MOVS R3, #0
.so:80407A1C STR R3, [SP,#0x18+var_14]
.so:80407A1E LDR R3, [R0]
.so:80407A20 ADD R1, SP, #0x18+var_14
.so:80407A22 LDR R2, =0x10006
.so:80407A24 LDR R3, [R3,#0x18]
.so:80407A26 BLX R3
.so:80407A28 CMP R0, #0
.so:80407A2A BNE loc_80407A70

```

drops.wooyun.org

```

so:8040773C init
so:8040773C
so:8040773C var_428 = -0x428
so:8040773C var_424 = -0x424
so:8040773C var_41C = -0x41C
so:8040773C var_2C = -0x2C
so:8040773C
so:8040773C PUSH {R4-R7,LR}
so:8040773E 23 4D LDR R5, =(dword_80419D48 - 0x80407748)
so:80407740 23 4C LDR R4, =0xFFFFFBE0
so:80407742 06 1C MOVS R6, R0
so:80407744 7D 44 ADD R5, PC ; dword_80419D48
so:80407746 2D 68 LDR R5, [R5]
so:80407748 A5 44 ADD SP, R4
so:8040774A FF A9 ADD R1, SP, #0x428+var_2C
so:8040774C 2B 68 LDR R3, [R5]
so:8040774E 10 31 ADDS R1, #0x10
so:80407750 20 4C LDR R4, =(dword_80419D4C - 0x80407772)
so:80407752 0B 60 STR R3, [R1]

```

drops.wooyun.org

由于IDA调试器还不够完善，单步调试的时候经常报错，最好先做一个内存快照，然后分析关键点的函数调用，在关键点下断而不是单步调试。

0x03 反调试初探

一般反调试在JNI_OnLoad中执行，也有的是在INIT_ARRAY段和INIT段中早于JNI_OnLoad执行。可通过readelf工具查看INIT_ARRAY段和INIT段的信息，定位到对应代码进行分析。

```

root@kali: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
0x0000000e (SONAME) Library soname: [REDACTED].so
0x0000001a (FINI_ARRAY) 0x194b8
0x0000001c (FINI_ARRAYSZ) 8 (bytes)
0x00000019 (INIT_ARRAY) 0x194c0
0x0000001b (INIT_ARRAYSZ) 16 (bytes)
0x00000010 (SYMBOLIC) 0x0
0x0000001e (FLAGS) SYMBOLIC BIND_NOW
0x6fffffff (FLAGS_1) Flags: NOW
0x00000000 (NULL) 0x0

```

drops.wooyun.org

INIT_ARRAY如下：

```

LOAD:804194C0 89 7A 40 80 DCD sub_80407A88+1
LOAD:804194C4 09 14 41 80 DCD sub_80411408+1
LOAD:804194C8 85 1B 41 80 DCD loc_80411B84+1
LOAD:804194CC 00 00 00 00 ALIGN 0x4

```

drops.wooyun.org

其中函数sub_80407A88的代码如下，通过检测时间差来检测是否中间有被单步调试执行：

```

1 void __fastcall sub_80407A88(__int64 a1, int a3)
2 {
3     int v2; // r001
4     int v3; // r003
5     __int64 v4; // [sp+0h] [bp-10h]@1
6     int v5; // [sp+8h] [bp-8h]@1
7
8     v4 = a1;
9     v5 = a3;
10    j_android_log_print(4, "[REDACTED]");
11    v2 = j_time(&v4);
12    sub_8040903C(v2);
13    j_time((char *)&v4 + 4);
14    if (HIDWORD(v4) - (signed int)v4 > 1)
15        j_exit(0);
16    v3 = sub_804087A0((int)&dword_8041A0F0);
17    check_com_android_reverse(v3);
18 }

```

drops.wooyun.org

sub_8040903C函数里就是脱壳了，首先读取/proc/self/maps找到自身模块基址，然后解析ELF文件格式，从程序头部表中找到类型为PT_LOAD，p_offset=0的程序头部表项，并从该程序段末尾读取自定义的数组，该数组保存了被加密的代码的偏移和大小，然后逐项解密。

```

1 int64 __fastcall sub_8040903C(int a1)
2 {
3     str_table *offset; // r4@0
4     unsigned int table_size; // r6@0
5     int v3; // r0@1
6     Elf32_Ehdr *p_elfHeader; // r5@1
7     program_table_entry32_t *p_program_header; // r3@1
8     signed int idx; // r2@1
9     int p_program_section_end; // r4@4
10    int v8; // r6@7
11    __int64 v10; // [sp+0h] [bp-20h]@1
12
13    LODWORD(v10) = a1;
14    v3 = getpid_0(a1);
15    p_elfHeader = (Elf32_Ehdr *)getMapAddr(v3, (int)"libegis.so");// 读取/proc/self/maps找到自身模块基址
16    p_program_header = (program_table_entry32_t *)((char *)p_elfHeader + p_elfHeader->e_phoff_PROGRAM_HEADER_OFFSET_IN_FILE);
17    idx = 0;
18    while ( idx < p_elfHeader->e_phnum_NUMBER_OF_PROGRAM_HEADER_ENTRIES )
19    {
20        if ( p_program_header->p_type == PT_LOAD && p_program_header->p_offset_FROM_FILE_BEGIN )
21        {
22            p_program_section_end = (int)((char *)p_elfHeader
23                + p_program_header->p_memsz_SEGMENT_RAM_LENGTH// 段在内存映像中占用的字节数
24                + p_program_header->p_vaddr_VIRTUAL_ADDRESS);// 段在内存中的虚拟地址
25            table_size = *(DWORD *)(p_program_section_end - 4);// 自定义的数组大小
26            offset = (str_table *)((p_program_section_end - table_size - 4));// 指向末尾的自定义数组, 保存了加密数据的偏移和大小
27            break;
28        }
29        ++idx;
30        ++p_program_header;
31    }
32    HIDWORD(v10) = table_size >> 3;
33    v8 = 0;
34    while ( v8 != HIDWORD(v10) ) // 循环异或解密
35    {
36        ++v8;
37        XORDecode((unsigned int)((char *)p_elfHeader + offset->offset) & 0xFFFFFFFF, offset->size);
38        ++offset;
39    }
40    return v10;
41}

```

drops.wooyun.org

函数check_com_android_reverse里检测是否加载了com.android.reverse, 检测到则直接退出。

```

1 int __fastcall check_com_android_reverse(int a1)
2 {
3     int v1; // r0@1
4     int result; // r0@1
5
6     v1 = getpid_0(a1);
7     result = getMapAddr(v1, (int)"com.android.reverse");
8     if ( !result )
9     {
10        j_exit(0);
11    }
12    return result;
13}

```

drops.wooyun.org

JNI_OnLoad函数中有几个关键的函数调用:

```

50:80407A18          JNI_OnLoad
50:80407A18          var_14          = -0x14
50:80407A18
50:80407A18 37 B5          PUSH          {R0-R2,R4,R5,LR}
50:80407A1A 00 23          MOVS          R3, #0
50:80407A1C 01 93          STR          R3, [SP,#0x18+var_14]
50:80407A1E 03 68          LDR          R3, [R0]
50:80407A20 01 A9          ADD          R1, SP, #0x18+var_14
50:80407A22 15 4a          LDR          R2, =0x10006
50:80407A24 98 69          LDR          R3, [R3,#0x18]
50:80407A26 98 47          BLX          R3
50:80407A28 00 28          CHP          R0, #0
50:80407A2A 21 D1          BNE          loc_80407A70
50:80407A2C 13 4b          LDR          R3, =(off_80419D50 - 0x80407A34)
50:80407A2E 01 9D          LDR          R5, [SP,#0x18+var_14]
50:80407A30 7B 44          ADD          R3, PC ; off_80419D50
50:80407A32 1B 68          LDR          R3, [R3]
50:80407A34 2A 68          LDR          R2, [R5]
50:80407A36 28 1C          MOVS          R0, R5
50:80407A38 19 68          LDR          R1, [R3]
50:80407A3A 93 69          LDR          R3, [R2,#0x18]
50:80407A3C 98 47          BLX          R3
50:80407A3E 01 1E          SUBS          R1, R0, #0
50:80407A40 08 D0          BEQ          loc_80407A54
50:80407A42 28 68          LDR          R0, [R5]
50:80407A44 0E 4a          LDR          R2, =(off_8041A0C4 - 0x80407A50)
50:80407A46 D7 23 9B 00  MOVS          R3, #0x35C
50:80407A4A C4 58          LDR          R4, [R0,R3]
50:80407A4C 7A 44          ADD          R2, PC
50:80407A4E 28 1C          MOVS          R0, R5
50:80407A50 01 23          MOVS          R3, #1
50:80407A52 A0 47          BLX          R4
50:80407A54
50:80407A54          loc_80407A54          ; CODE XREF: JNI_OnLoad+28↑j
50:80407A54 FF FF 0E FF  BL          call_system_property_get
50:80407A58 02 F0 52 F8  BL          sub_80409B00
50:80407A5C 09 4b          LDR          R3, =(dword_8041A0F0 - 0x80407A62)
50:80407A5E 7B 44          ADD          R3, PC ; dword_8041A0F0
50:80407A60 1B 68          LDR          R0, [R3]
50:80407A62 01 7B 17 F8  BL          checkProcStatus
50:80407A66 01 98          LDR          R0, [SP,#0x18+var_14]
50:80407A68 FF F7 B8 FE  BL          sub_804077DC
50:80407A6C 02 48          LDR          R0, =0x10006
50:80407A6E 01 E0          B            locret_80407A74

```

drops.wooyun.org

call_system_property_get检测手机上的一些硬件信息，判断是否在调试器中。

```

17 v11 = 0xD5405C9E;
18 j__system_property_get((int)"ro.product.manufacturer", (int)&v9);
19 ro_product_manufacturer = j_strdup(&v9);
20 j__system_property_get((int)"ro.build.version.release", (int)&v10);
21 ro_build_version_release = j_strdup(&v10);
22 j__system_property_get((int)"ro.build.version.sdk", (int)&v5);
23 ro_build_version_sdk = j_atoi(&v5);
24 if ( !j__system_property_get((int)"persist.sys.dalvik.vm.lib", (int)&v4) )
25     j_strncpy((int)&v4, (int)"libdvm.so", 11);
26 persist_sys_dalvik_vm_lib = j_strdup(&v4);
27 if ( !j__system_property_get((int)"ro.product.cpu.abi", (int)v6) )
28 {
29     strcpy(v6, "armeabi");
30     v7 = 0;
31 }
32 ro_product_cpu_abi = j_strdup(v6);
33 v0 = j_strncpy((int)"libart.so", 0);
34 v1 = 1;
35 if ( v0 )
36     v1 = 0;
37 unk_80431900 = v1;
38 v2 = j__system_property_get((int)"ro.yunos.version", (int)&v8);
39 result = v2 - (((unsigned int)v2 < 1) + v2 - 1);
40 unk_80431901 = result;
41 if ( v11 != 0xD5405C9E )
42     j__stack_chk_fail(result);
43 return result;

```

drops.wooyun.org

checkProcStatus函数检测进程的状态，打开/proc/SPID/status，读取第6行得到TracerPid，发现被跟踪调试则直接退出。


```

1 int __fastcall checkProcStatus(int a1)
2 {
3     int v1; // r7@1
4     signed int v2; // r5@1
5     int v3; // r0@1
6     int v4; // r6@1
7     int result; // r0@3
8     char v6; // zf@3
9     int v7; // [sp+0h] [bp-160h]@3
10    char v8; // [sp+4h] [bp-15Ch]@3
11    char v9; // [sp+18h] [bp-148h]@1
12    char v10; // [sp+7Ch] [bp-E4h]@2
13    unsigned int v11; // [sp+144h] [bp-1Ch]@1
14
15    v1 = a1;
16    v2 = 5;
17    v11 = 0xD5405C9E;
18    v3 = getpid_0(a1);
19    j_sprintf(&v9, "/proc/%d/status", v3);
20    v4 = j_fopen(&v9, "r");
21    do
22    {
23        --v2;
24        j_fgets(&v10, 1024, v4);
25    }
26    while ( v2 );
27    j_fscanf(v4, "%s %d", &v8, &v7);
28    result = j_fclose_0(v4);
29    v6 = v7 == 0;
30    if ( v7 )
31        v6 = v7 == v1;
32    if ( !v6 )
33        j_exit(0);
34    if ( v11 != 0xD5405C9E )
35        j__stack_chk_fail(result);
36    return result;
37 }

```

drops.wooyun.org

通过命令行查询进程信息，一共有3个同名进程，创建顺序为33->415->430->431。其中415和431处于调试状态：

```

D:\apkdbg>adb shell ps 33
USER      PID    PPID  USIZE  RSS      WCHAN    PC      NAME
root      33      1      115708  26176   c009b74c  afd0b844  $ zygote

D:\apkdbg>adb shell ps hw
USER      PID    PPID  USIZE  RSS      WCHAN    PC      NAME
app_34    415     33    124812  18536   ffffffff  80407a18  T hw.helloworld
app_34    430     415    123796  15436   c00520f8  afd0bdac  S hw.helloworld
app_34    431     430    124820  15440   ffffffff  afd15e50  T hw.helloworld

D:\apkdbg>_

```

进程415被进程405（即IDA的android_server）调试：

```

D:\apkdbg>adb shell ps 405
USER      PID    PPID  USIZE  RSS      WCHAN    PC      NAME
root      405     404     5956   4528   ffffffff  afd0c164  S /data/data/sv

D:\apkdbg>adb shell cat /proc/415/status
Name:      hw.helloworld
State:     T (tracing stop)
Tgid:      415
Pid:       415
PPid:      33
TracerPid: 405
Uid:       10034 10034 10034 10034
Gid:       10034 10034 10034 10034
FDSize:    256

```

进程431被其父进程430调试：

```

D:\apkdbg>adb shell cat /proc/431/status
Name:      hw.helloworld
State:     T (tracing stop)
Tgid:      431
Pid:       431
PPid:      430
TracerPid: 430

```

要过这种反调试可在调用点直接修改跳转指令，让代码在检测到被调试后继续正常的执行路径，或者干脆nop掉整个函数即可。检测调试之后，就是调用ptrace附加自身，防止其他进程再一次附加，起到反调试作用。

```

1 int call_ptrace()
2 {
3     if ( j_ptrace(0, 0) == -1 )
4         j_exit(0);
5     return 0;
6 }

```

drops.wooyun.org

修改跳转指令BNE (0xD1) 为B(0xE0)，直接返回即可。

```

:80408A74      call_ptrace          ; DATA XREF: sub_804087A0:loc_8041
:80408A74 D8 16      ASRS      R0, R3, #0x1B
:80408A76 01 00      MOVS      R1, R0
:80408A78 08 05      PUSH     {R3,LR}
:80408A7A 00 20      MOVS      R0, #0
:80408A7C 01 1C      MOVS      R1, R0
:80408A7E 02 1C      MOVS      R2, R0
:80408A80 03 1C      MOVS      R3, R0
:80408A82 FE F7 C4 EB BLX      j_ptrace
:80408A86 01 30      ADDS     R0, #1
:80408A88 02 E0      B        loc_80408A90
:80408A8A      ; -----
:80408A8A 00 20      MOVS      R0, #0
:80408A8C FE F7 82 EB BLX      j_exit
:80408A90      ; -----
:80408A90      loc_80408A90          ; CODE XREF: call_ptrace+14fj
:80408A90 00 20      MOVS      R0, #0
:80408A92 08 0D      POP      {R3,PC}
:80408A92      ; End of function call_ptrace
drops.wooyun.org

```

当然，更加彻底的方法是修改android源码中bionic中的libc中的ptrace系统调用。检测到一个进程试图附加自身时直接返回0即可。

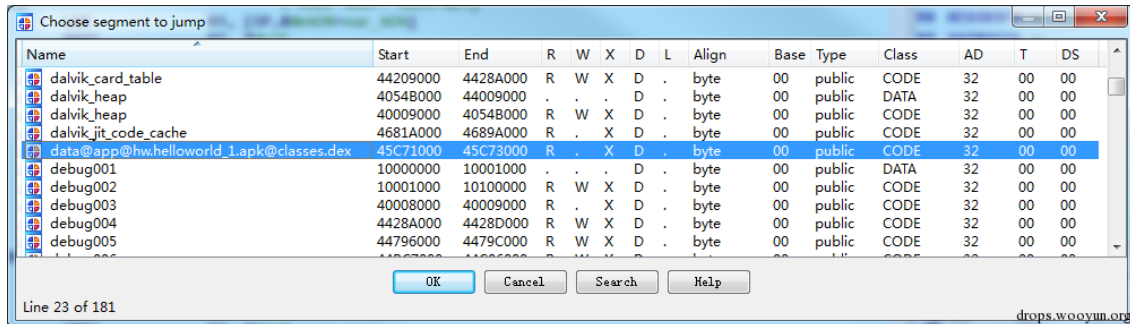
上面几处反调试点在检测到调试器后都直接调用exit()退出进程了，所以直接nop掉后按F9执行。然后就断在了init函数入口，顺利过掉反调试：

```

R12
PC
:so:8040773C      init
:so:8040773C
:so:8040773C      var_428= -0x428
:so:8040773C      var_424= -0x424
:so:8040773C      var_41C= -0x41C
:so:8040773C      var_2C= -0x2C
:so:8040773C
:so:8040773C F0 05      PUSH     {R4-R7,LR}
:so:8040773E 23 4D      LDR      R5, =(dword_80419D48 - 0x80407748)
:so:80407740 23 4C      LDR      R4, =0xFFFFFBE0
:so:80407742 06 1C      MOVS     R6, R0
:so:80407744 7D 44      ADD      R5, PC ; dword_80419D48
:so:80407746 2D 68      LDR      R5, [R5]
:so:80407748 A5 44      ADD      SP, R4
:so:8040774A FF A9      ADD      R1, SP, #0x428+var_2C
:so:8040774C 2B 68      LDR      R3, [R5]
:so:8040774E 10 31      ADDS     R1, #0x10
drops.wooyun.org

```

init函数在每个类加载的时候被调用，用于恢复当前类的被隐藏方法。首次调用时解密dex文件末尾的附加数据，得到事先保存的所有类的方法属性，然后根据传入的类名查找该类的被隐藏方法，并恢复对应属性字段。执行完init函数，当前类的方法已经恢复了。然后转到dex文件的内存地址



dump出dex文件，保存为dump.dex。

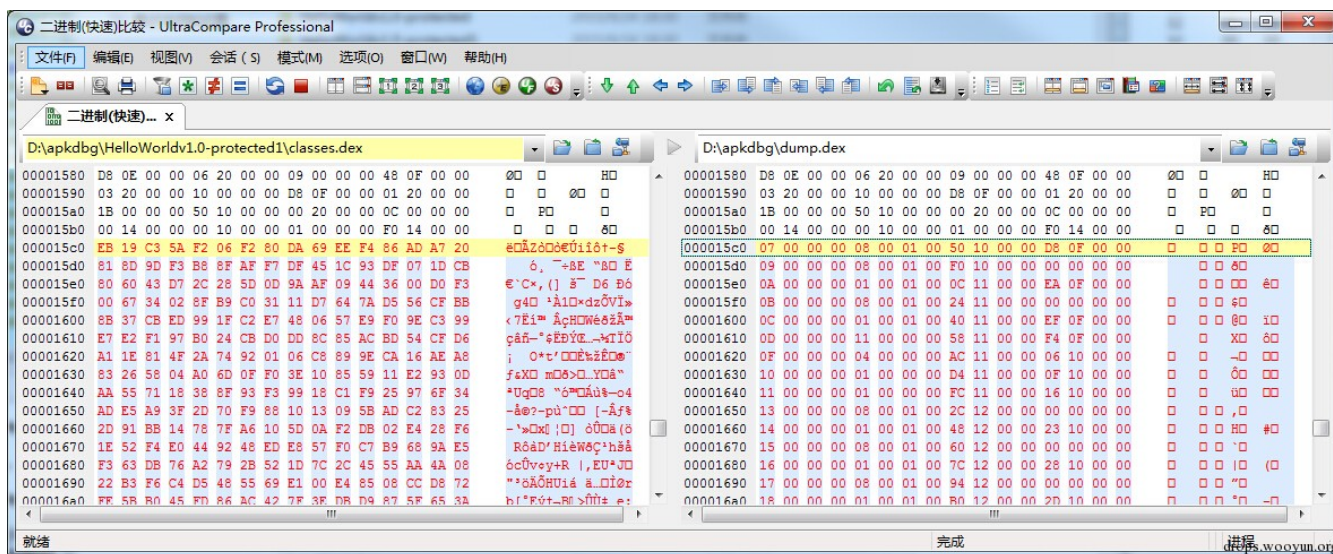
```

Hex View-1
45C71000 64 65 79 0A 30 33 36 00 28 00 00 00 00 1B 00 00 dey.036.(.....
45C71010 28 1B 00 00 67 02 00 00 90 1D 00 00 98 01 00 00 (...g.....dex.035.
45C71020 00 00 00 00 97 16 5F 09 64 65 78 0A 3C 33 35 00 .....dex.035.
45C71030 C7 7E 11 0F D4 69 EF 08 C3 34 2C 26 C0 1A 4A 3B ~...i...4,R...J;
45C71040 98 D5 57 E9 EA 6D 5E 21 00 1B 00 00 70 00 00 00 xW...m^t...p...
45C71050 78 56 34 12 00 00 00 00 00 00 00 00 F0 14 00 00 xU4.....
45C71060 71 00 00 00 70 00 00 00 1E 00 00 00 34 02 00 00 q...p.....4...
45C71070 0E 00 00 00 AC 02 00 00 13 00 00 00 54 03 00 00 .....T...
45C71080 2A 00 00 00 EC 03 00 00 0C 00 00 00 3C 05 00 00 *.....<...
45C71090 04 0F 00 00 BC 06 00 00 BC 06 00 00 BE 06 00 00 .....
45C710A0 C3 06 00 00 CD 06 00 00 D5 06 00 00 E5 06 00 00 .....
45C710B0 EF 06 00 00 FB 06 00 00 00 07 00 00 16 07 00 00 .....
45C710C0 42 07 00 00 49 07 00 00 51 07 00 00 54 07 00 00 B...I...Q...T...
45C710D0 59 07 00 00 5C 07 00 00 60 07 00 00 78 07 00 00 V...\. ...x...
45C710E0 94 07 00 00 A8 07 00 00 8D 07 00 00 D1 07 00 00 .....
45C710F0 E6 07 00 00 03 08 00 00 1C 08 00 00 3C 08 00 00 .....<...
45C71100 60 08 00 00 80 08 00 00 A3 08 00 00 C0 08 00 00 .....
45C71110 DE 08 00 00 F6 08 00 00 0F 09 00 00 25 09 00 00 .....%...
45C71120 3F 09 00 00 57 09 00 00 71 09 00 00 8B 09 00 00 ?...W...q...
45C71130 A4 09 00 00 B7 09 00 00 D1 09 00 00 E5 09 00 00 .....
45C71140 F9 09 00 00 0A 00 00 00 1B 0A 00 00 2E 0A 00 00 .....drops.wooyun.org

```

0x04 恢复隐藏方法

对比一下原始dex文件，发现dex文件末尾的附加数据被解密出来了：



仔细分析一下附加数据的数据结构可以发现，它是一个数组，保存了所有类的所有方法的method_idx、access_flags、code_off、debug_info_off属性，解密后的这些属性都是uint类型的,如下图：

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
15C0h:	07	00	00	00	08	00	01	00	50	10	00	00	D8	0F	00	00	
15D0h:	09	00	00	00	08	00	01	00	F0	10	00	00	00	00	00	00	
15E0h:	0A	00	00	00	01	00	01	00	0C	11	00	00	EA	0F	00	00	
15F0h:	0B	00	00	00	08	00	01	00	24	11	00	00	00	00	00	00	
1600h:	0C	00	00	00	01	00	01	00	40	11	00	00	EF	0F	00	00	
1610h:	0D	00	00	00	11	00	00	00	58	11	00	00	F4	0F	00	00	
1620h:	0E	00	00	00	04	00	00	00	AC	11	00	00	06	10	00	00	
1630h:	10	00	00	00	01	00	00	00	D4	11	00	00	0F	10	00	00	
1640h:	11	00	00	00	01	00	00	00	FC	11	00	00	16	10	00	00	
1650h:	13	00	00	00	08	00	01	00	2C	12	00	00	00	00	00	00	
1660h:	14	00	00	00	01	00	01	00	48	12	00	00	23	10	00	00	
1670h:	15	00	00	00	08	00	01	00	60	12	00	00	00	00	00	00	
1680h:	16	00	00	00	01	00	01	00	7C	12	00	00	28	10	00	00	
1690h:	17	00	00	00	08	00	01	00	94	12	00	00	00	00	00	00	
16A0h:	18	00	00	00	01	00	01	00	B0	12	00	00	2D	10	00	00	
16B0h:	19	00	00	00	08	00	01	00	C8	12	00	00	00	00	00	00	
16C0h:	1A	00	00	00	01	00	01	00	E4	12	00	00	32	10	00	00	
16D0h:	1B	00	00	00	08	00	01	00	FC	12	00	00	00	00	00	00	
16E0h:	1C	00	00	00	01	00	01	00	18	13	00	00	37	10	00	00	
16F0h:	1D	00	00	00	08	00	01	00	30	13	00	00	00	00	00	00	
1700h:	1E	00	00	00	01	00	01	00	4C	13	00	00	3C	10	00	00	
1710h:	1F	00	00	00	08	00	01	00	64	13	00	00	00	00	00	00	
1720h:	20	00	00	00	01	00	01	00	80	13	00	00	41	10	00	00	
1730h:	21	00	00	00	08	00	01	00	98	13	00	00	00	00	00	00	
1740h:	22	00	00	00	01	00	01	00	B4	13	00	00	46	10	00	00	
1750h:	23	00	00	00	08	00	01	00	CC	13	00	00	00	00	00	00	
1760h:	24	00	00	00	01	00	01	00	E8	13	00	00	4B	10	00	00	
17																																	
17	method_idx	access_				code_off				debug_info_off																							
1790h:	00	00	00	00	flags	00	00	00	00	00	00	00	00	00	00	00	
17A0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
17B0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
17C0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
17D0h:	03	9A	94	B0	A5	09	00	00	9C	06	00	00	02	00	00	00	
17E0h:	19	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

drops.wooyun.org

其中黄色框里的就是MainActivity的各方法的属性，知道这些就可以修复dex文件，恢复出被隐藏的方法了。下图就是恢复后的MainActivity类：

```
MainActivity.class x
package hw.helloworld;

import android.app.Activity;

public class MainActivity
    extends Activity
{
    static
    {
        ProxyApplication.init("aHcuaGVzbG93b3JsZC5NYWluQWw0aXZpdHk=");
    }

    public final boolean finaltest()
    {
        int i = 3 + 1;
        Log.i("i", "finaltest");
        if (i == 4)
        {
            Log.i("i", "fianltest return true");
            return true;
        }
        (i + 1);
        Log.i("i", "fianltest return false");
        return false;
    }

    protected void onCreate(Bundle paramBundle)
    {
        super.onCreate(paramBundle);
        setContentView(2130903040);
        finaltest();
    }

    public boolean onCreateOptionsMenu(Menu paramMenu)
    {
        getMenuInflater().inflate(2131165184, paramMenu);
        return true;
    }

    public boolean onOptionsItemSelected(MenuItem paramMenuItem)
    {
        if (paramMenuItem.getItemId() == 2131230720) {
            return true;
        }
        return super.onOptionsItemSelected(paramMenuItem);
    }
}
```

drops.wooyun.org

0x05 总结

以上就是通过实例分析展示出来的对抗和反调试手段。so模块中的反调试手段比较初级，可以非常简单的手工patch内存指令过掉，而隐藏方法的这种手段对art模式不兼容，不推荐使用这种方法加固应用。总的来说还是过于简单。预计未来通过虚拟机来加固应用将是一大发展方向。