

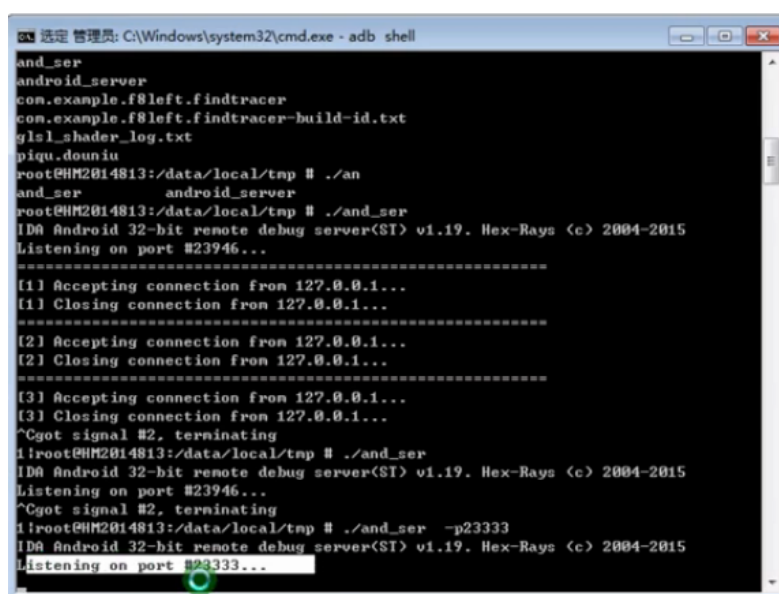
1. 调试检测

在调试加壳软件的时候，总是会突然出现调试中断，或者是程序的异常退出，但是一旦不处于调试的时候又能够正常启动，这些都是因为调试被 app 检测到的原因。

2. 基本的调试监控

常用的监控手段：

- 1) 检测用户组 cmdline 中是否存在调试进程(gdb、gdbserver、android_server、xposed 等)
修改调试器的名称
- 2) 检测线程状态，查看是否存在被调试的线程
修改读取文件中特征值
- 3) 检测进程状态，查看是否存在调试的进程
修改读取文件中特征值
- 4) 检测传输端口，如 23946（ida 调试的默认端口）等端口是否被调试进程启用了



```
and_ser
android_server
com.example.f8left.findtracer
com.example.f8left.findtracer-build-id.txt
y1sl_shader_log.txt
piqu.douniu
root@HM2014813:/data/local/tmp # ./an
and_ser
root@HM2014813:/data/local/tmp # ./and_ser
IDA Android 32-bit remote debug server(ST) v1.19. Hex-Rays (c) 2004-2015
Listening on port #23946...

[1] Accepting connection from 127.0.0.1...
[1] Closing connection from 127.0.0.1...

[2] Accepting connection from 127.0.0.1...
[2] Closing connection from 127.0.0.1...

[3] Accepting connection from 127.0.0.1...
[3] Closing connection from 127.0.0.1...

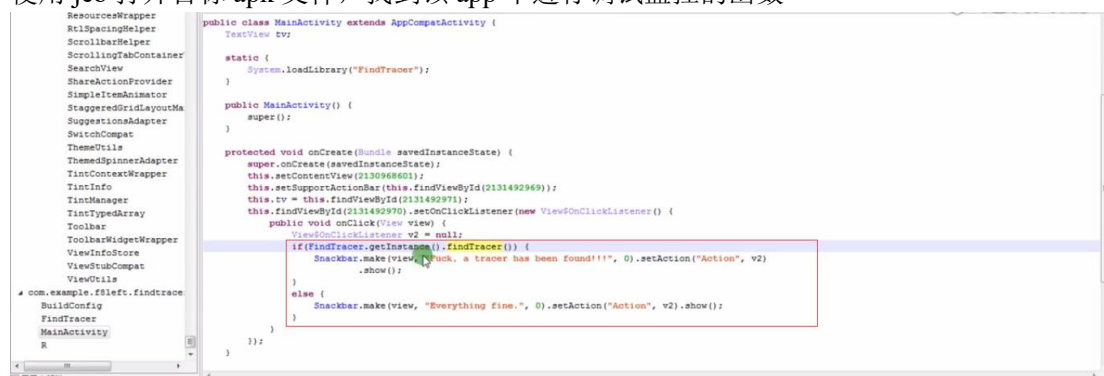
^Cgot signal #2, terminating
1:root@HM2014813:/data/local/tmp # ./and_ser
IDA Android 32-bit remote debug server(ST) v1.19. Hex-Rays (c) 2004-2015
Listening on port #23946...

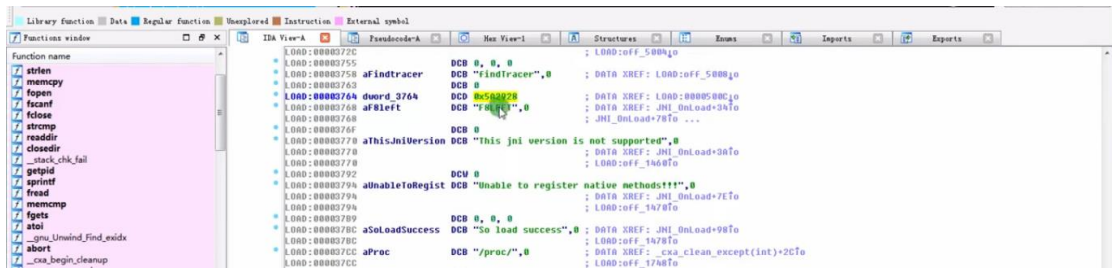
^Cgot signal #2, terminating
1:root@HM2014813:/data/local/tmp # ./and_ser -p23333
IDA Android 32-bit remote debug server(ST) v1.19. Hex-Rays (c) 2004-2015
Listening on port #23333...
```

可以通过修改调试端口绕过

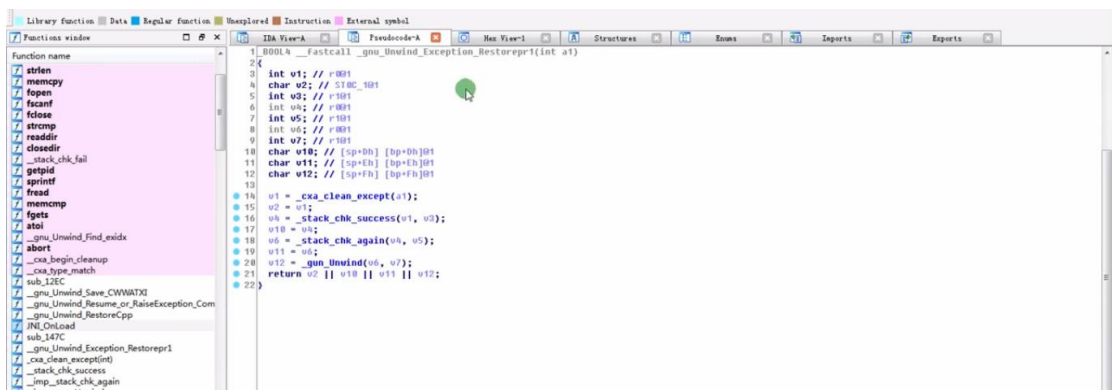
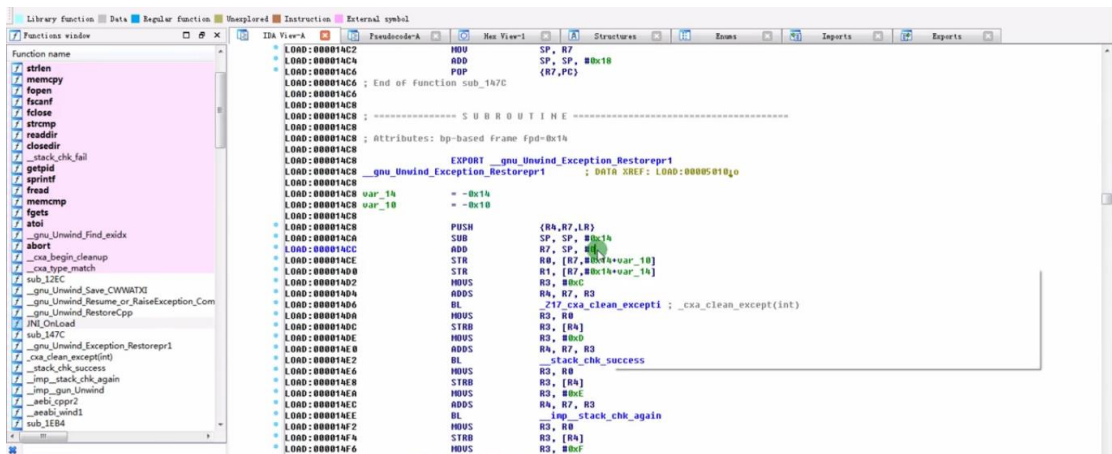
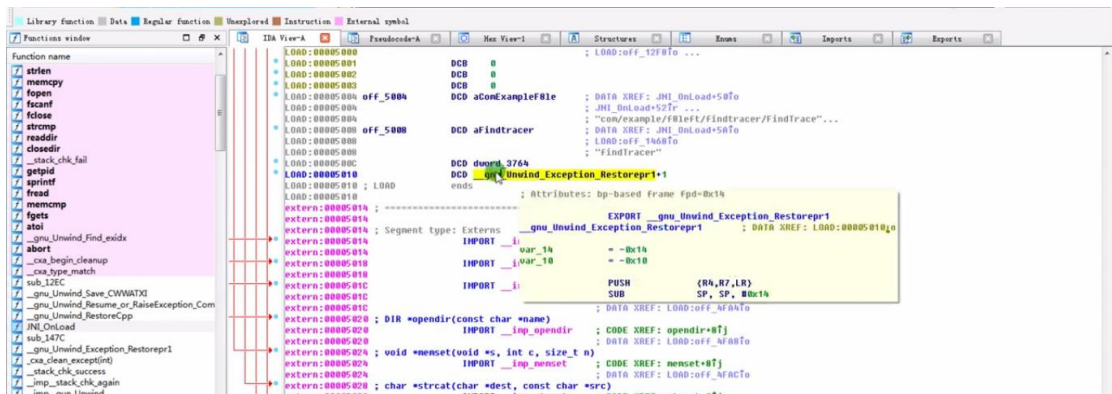
3. Demo 演示

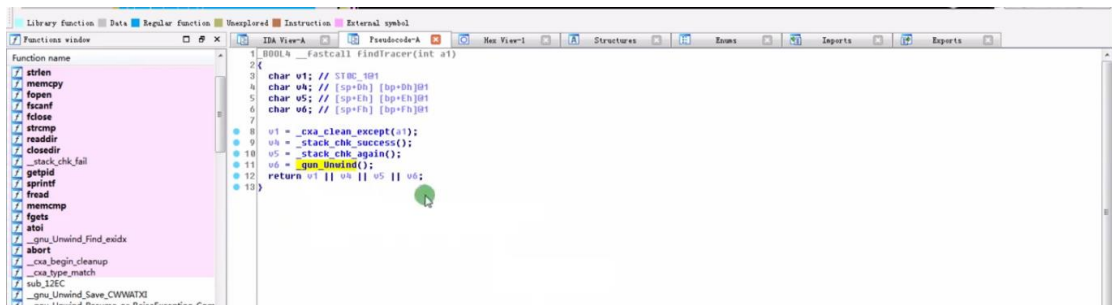
使用 jeb 打开目标 apk 文件，找到该 app 中进行调试监控的函数



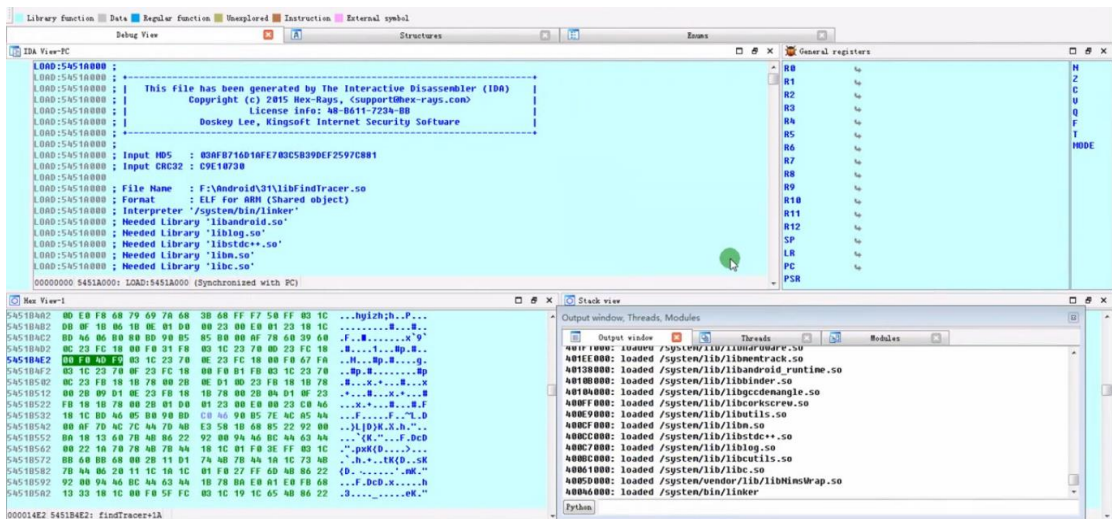


找到 findTrace()函数的地址，跳转过去之后找到 findTrace()函数

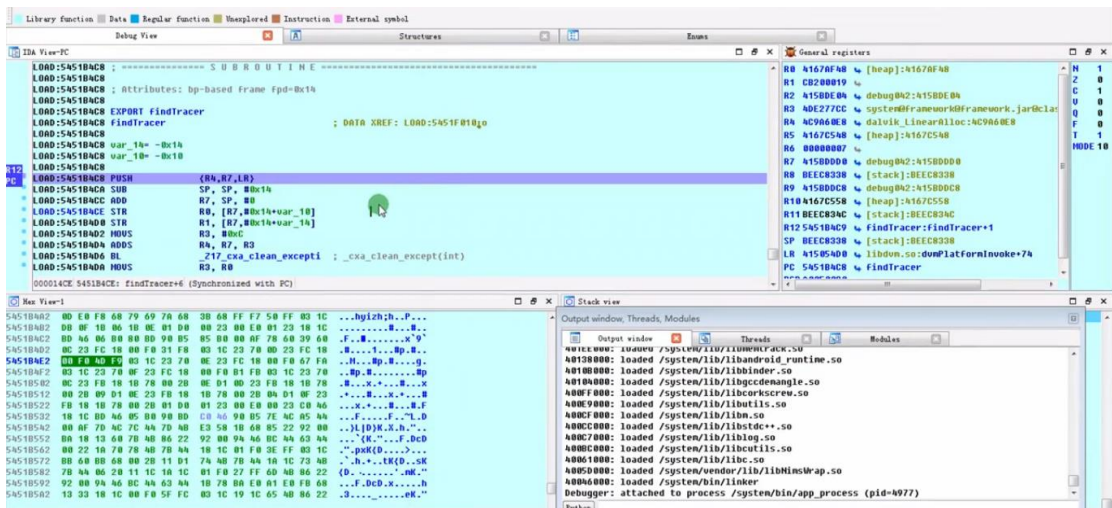




然后使用 ida 进行动态挂接



开始动态调试，程序在 findTrace()处断下



调试器已经被检测到

Name

2014813-12758134

com.miui.home

com.android.mipop

com.android.updater

com.baidu.map.location

com.android.musicfx

com.xiaom.market

com.lbe.security.miui

com.android.dc

com.miui.core

de.robov.android.xposed.installer

com.miui.securitycenter

com.android.packageinstaller

com.miui.powerkeeper.service

com.android.mms

Offline

1281

1218

2626

1158

2824

2188

1485

1872

1042

4178

1492

2965

1366

2006

4.4.4 debug

8600

8601

8602

8603

8604 / 8700

8605

8606

8607

8608

8609

8610

8611

8612

8613

LogCat

Console

Verbose

Filter

Clear

Search for messages. Accepts Java regexes. Prefix with pid, app, tag; or text to limit scope.

verbose

Filter

Clear

All messages (no filters) (65)

cocos

FBLEFT

L	Time	PID	TID	Application	Tag	Text
E	05-09 12:44:30.397	4977	4977	com.example.flileft.fildnt...	FILEFT	A debugger ./android_arm64 has been found!!!
E	05-09 12:44:30.397	4977	4977	com.example.flileft.fildnt...	FILEFT	A TracerPid has been found: 5029
E	05-09 12:44:30.397	4977	4977	com.example.flileft.fildnt...	FILEFT	A TracerPid has been found: 5029
E	05-09 12:44:30.397	4977	4977	com.example.flileft.fildnt...	FILEFT	A TracerPid has been found: 5029
E	05-09 12:44:30.397	4977	4977	com.example.flileft.fildnt...	FILEFT	A TracerPid has been found: 5029
E	05-09 12:44:30.397	4977	4977	com.example.flileft.fildnt...	FILEFT	A TracerPid has been found: 5029
E	05-09 12:44:30.397	4977	4977	com.example.flileft.fildnt...	FILEFT	A TracerPid has been found: 5029
E	05-09 12:44:30.397	4977	4977	com.example.flileft.fildnt...	FILEFT	A TracerPid has been found: 5029
E	05-09 12:44:30.407	4977	4977	com.example.flileft.fildnt...	FILEFT	A TracerPid has been found: 5029
E	05-09 12:44:30.407	4977	4977	com.example.flileft.fildnt...	FILEFT	A TracerPid has been found: 5029
E	05-09 12:44:30.407	4977	4977	com.example.flileft.fildnt...	FILEFT	A TracerPid has been found: 5029

调试器判断的代码（函数1）

The screenshot displays the Android Studio IDE with the decompiled Java code for the 'start' method of the 'Service' class. The code is split into two panels: the top panel shows the original Java code, and the bottom panel shows the decompiled code. The decompiled code is annotated with comments in Chinese, explaining the logic of the 'start' method, including the creation of a 'Thread' object, the 'run' method call, and the 'stop' method call. The decompiled code is also annotated with comments in Chinese, explaining the logic of the 'start' method, including the creation of a 'Thread' object, the 'run' method call, and the 'stop' method call. The decompiled code is also annotated with comments in Chinese, explaining the logic of the 'start' method, including the creation of a 'Thread' object, the 'run' method call, and the 'stop' method call.

```

Library function Data Regular function Unexplored Instruction External symbol
Debug View Structures Enums General registers
IDA View-PC Fcodecode-k General registers
signed int v0; // [sp+10h] [bp+10h]05
int v1; // [sp+10h] [bp+10h]05
int v10; // [sp+210h] [bp+210h]01
v0 = 0;
dirp = j_0ff5_5451EF70;
if ( ! dirp )
    JUMPOUT(0x05_54510D0C);
while ( 1 )
{
    v2 = j_1_readdir(dirp);
    v3 = v2;
    if ( ! ( ( ( _PAIR_((unsigned int)v2, (unsigned int)v3) >= 0x25) && 0x1) >> 32) & 0xFF ) )
        break;
    if ( !_aabi_winsz(0x02->d_name[0]) )
    {
        v7 = 18697709;
        v8 = 12131;
        j_1_monotonic(0, 0, &v7);
        j_1_strerror(char *0x07, 0x05->d_name[0]);
        v1 = j_1_strlen(const char *0x07);
        j_1_memcpy(char *0x07 + v1, "condition", 9);
        stream = j_1_fopen(const char *0x07, "r");
        if ( ! stream )
        {
            j_1_fscanf(stream, "%s", 0x07);
            j_1_fclose(stream);
        }
        break;
    }
    if ( !_aabi_winsz(0x02->d_name[0]) )
    {
        v7 = 18697709;
        v8 = 12131;
        j_1_monotonic(0, 0, &v7);
        j_1_strerror(char *0x07, 0x05->d_name[0]);
        v1 = j_1_strlen(const char *0x07);
        j_1_memcpy(char *0x07 + v1, "condition", 9);
        stream = j_1_fopen(const char *0x07, "r");
        if ( ! stream )
        {
            j_1_fscanf(stream, "%s", 0x07);
            j_1_fclose(stream);
        }
        break;
    }
    if ( !_aabi_winsz(0x02->d_name[0]) )
    {
        v7 = 18697709;
        v8 = 12131;
        j_1_monotonic(0, 0, &v7);
        j_1_strerror(char *0x07, 0x05->d_name[0]);
        v1 = j_1_strlen(const char *0x07);
        j_1_memcpy(char *0x07 + v1, "condition", 9);
        stream = j_1_fopen(const char *0x07, "r");
        if ( ! stream )
        {
            j_1_fscanf(stream, "%s", 0x07);
            j_1_fclose(stream);
        }
        break;
    }
    if ( !_aabi_winsz(0x02->d_name[0]) )
    {
        v7 = 18697709;
        v8 = 12131;
        j_1_monotonic(0, 0, &v7);
        j_1_strerror(char *0x07, 0x05->d_name[0]);
        v1 = j_1_strlen(const char *0x07);
        j_1_memcpy(char *0x07 + v1, "condition", 9);
        stream = j_1_fopen(const char *0x07, "r");
        if ( ! stream )
        {
            j_1_fscanf(stream, "%s", 0x07);
            j_1_fclose(stream);
        }
        break;
    }
    if ( !_aabi_winsz(0x02->d_name[0]) )
    {
        v7 = 18697709;
        v8 = 12131;
        j_1_monotonic(0, 0, &v7);
        j_1_strerror(char *0x07, 0x05->d_name[0]);
        v1 = j_1_strlen(const char *0x07);
        j_1_memcpy(char *0x07 + v1, "condition", 9);
        stream = j_1_fopen(const char *0x07, "r");
        if ( ! stream )
        {
            j_1_fscanf(stream, "%s", 0x07);
            j_1_fclose(stream);
        }
        break;
    }
    if ( !_aabi_winsz(0x02->d_name[0]) )
    {
        v7 = 18697709;
        v8 = 12131;
        j_1_monotonic(0, 0, &v7);
        j_1_strerror(char *0x07, 0x05->d_name[0]);
        v1 = j_1_strlen(const char *0x07);
        j_1_memcpy(char *0x07 + v1, "condition", 9);
        stream = j_1_fopen(const char *0x07, "r");
        if ( ! stream )
        {
            j_1_fscanf(stream, "%s", 0x07);
            j_1_fclose(stream);
        }
        break;
    }
    if ( !_aabi_winsz(0x02->d_name[0]) )
    {
        v7 = 18697709;
        v8 = 12131;
        j_1_monotonic(0, 0, &v7);
        j_1_strerror(char *0x07, 0x05->d_name[0]);
        v1 = j_1_strlen(const char *0x07);
        j_1_memcpy(char *0x07 + v1, "condition", 9);
        stream = j_1_fopen(const char *0x07, "r");
        if ( ! stream )
        {
            j_1_fscanf(stream, "%s", 0x07);
            j_1_fclose(stream);
        }
        break;
    }
    if ( !_aabi_winsz(0x02->d_name[0]) )
    {
        v7 = 18697709;
        v8 = 12131;
        j_1_monotonic(0, 0, &v7);
        j_1_strerror(char *0x07, 0x05->d_name[0]);
        v1 = j_1_strlen(const char *0x07);
        j_1_memcpy(char *0x07 + v1, "condition", 9);
        stream = j_1_fopen(const char *0x07, "r");
        if ( ! stream )
        {
            j_1_fscanf(stream, "%s", 0x07);
            j_1_fclose(stream);
        }
        break;
    }
    if ( !_aabi_winsz(0x02->d_name[0]) )
    {
        v7 = 18697709;
        v8 = 12131;
        j_1_monotonic(0, 0, &v7);
        j_1_strerror(char *0x07, 0x05->d_name[0]);
        v1 = j_1_strlen(const char *0x07);
        j_1_memcpy(char *0x07 + v1, "condition", 9);
        stream = j_1_fopen(const char *0x07, "r");
        if ( ! stream )
        {
            j_1_fscanf(stream, "%s", 0x07);
            j_1_fclose(stream);
        }
        break;
    }
    if ( !_aabi_winsz(0x02->d_name[0]) )
    {
        v7 = 18697709;
        v8 = 12131;
        j_1_monotonic(0, 0, &v7);
        j_1_strerror(char *0x07, 0x05->d_name[0]);
        v1 = j_1_strlen(const char *0x07);
        j_1_memcpy(char *0x07 + v1, "condition", 9);
        stream = j_1_fopen(const char *0x07, "r");
        if ( ! stream )
        {
            j_1_fscanf(stream, "%s", 0x07);
            j_1_fclose(stream);
        }
        break;
    }
    if ( !_aabi_winsz(0x02->d_name[0]) )
    {
        v7 = 18697709;
        v8 = 12131;
        j_1_monotonic(0, 0, &v7);
        j_1_strerror(char *0x07, 0x05->d_name[0]);
        v1 = j_1_strlen(const char *0x07);
        j_1_memcpy(char *0x07 + v1, "condition", 9);
        stream = j_1_fopen(const char *0x07, "r");
        if ( ! stream )
        {
            j_1_fscanf(stream, "%s", 0x07);
            j_1_fclose(stream);
        }
        break;
    }
    if ( !_aabi_winsz(0x02->d_name[0]) )
    {
        v7 = 18697709;
        v8 = 12131;
        j_1_monotonic(0, 0, &v7);
        j_1_strerror(char *0x07, 0x05->d_name[0]);
        v1 = j_1_strlen(const char *0x07);
        j_1_memcpy(char *0x07 + v1, "condition", 9);
        stream = j_1_fopen(const char *0x07, "r");
        if ( ! stream )
        {
            j_1_fscanf(stream, "%s", 0x07);
            j_1_fclose(stream);
        }
        break;
    }
    if ( !_aabi_winsz(0x02->d_name[0]) )
    {
        v7 = 18697709;
        v8 = 12131;
        j_1_monotonic(0, 0, &v7);
        j_1_strerror(char *0x07, 0x05->d_name[0]);
        v1 = j_1_strlen(const char *0x07);
        j_1_memcpy(char *0x07 + v1, "condition", 9);
        stream = j_1_fopen(const char *0x07, "r");
        if ( ! stream )
        {
            j_1_fscanf(stream, "%s", 0x07);
            j_1_fclose(stream);
        }
        break;
    }
    if ( !_aabi_winsz(0x02->d_name[0]) )
    {
        v7 = 18697709;
        v8 = 12131;
        j_1_monotonic(0, 0, &v7);
        j_1_strerror(char *0x07, 0x05->d_name[0]);
        v1 = j_1_strlen(const char *0x07);
        j_1_memcpy(char *0x07 + v1, "condition", 9);
        stream = j_1_fopen(const char *0x07, "r");
        if ( ! stream )
        {
            j_1_fscanf(stream, "%s", 0x07);
            j_1_fclose(stream);
        }
        break;
    }
    if ( !_aabi_winsz(0x02->d_name[0]) )
    {
        v7 = 18697709;
        v8 = 12131;
        j_1_monotonic(0, 0, &v7);
        j_1_strerror(char *0x07, 0x05->d_name[0]);
        v1 = j_1_strlen(const char *0x07);
        j_1_memcpy(char *0x07 + v1, "condition", 9);
        stream = j_1_fopen(const char *0x07, "r");
        if ( ! stream )
        {
            j_1_fscanf(stream, "%s", 0x07);
            j_1_fclose(stream);
        }
        break;
    }
    if ( !_aabi_winsz(0x02->d_name[0]) )
    {
        v7 = 18697709;
        v8 = 12131;
        j_1_monotonic(0, 0, &v7);
        j_1_strerror(char *0x07, 0x05->d_name[0]);
        v1 = j_1_strlen(const char *0x07);
        j_1_memcpy(char *0x07 + v1, "condition", 9);
        stream = j_1_fopen(const char *0x07, "r");
        if ( ! stream )
        {
            j_1_fscanf(stream, "%s", 0x07);
            j_1_fclose(stream);
        }
        break;
    }
    if ( !_aabi_winsz(0x02->d_name[0]) )
    {
        v7 = 18697709;
        v8 = 12131;
        j_1_monotonic(0, 0, &v7);
        j_1_strerror(char *0x07, 0x05->d_name[0]);
        v1 = j_1_strlen(const char *0x07);
        j_1_memcpy(char *0x07 + v1, "condition", 9);
        stream = j_1_fopen(const char *0x07, "r");
        if ( ! stream
```

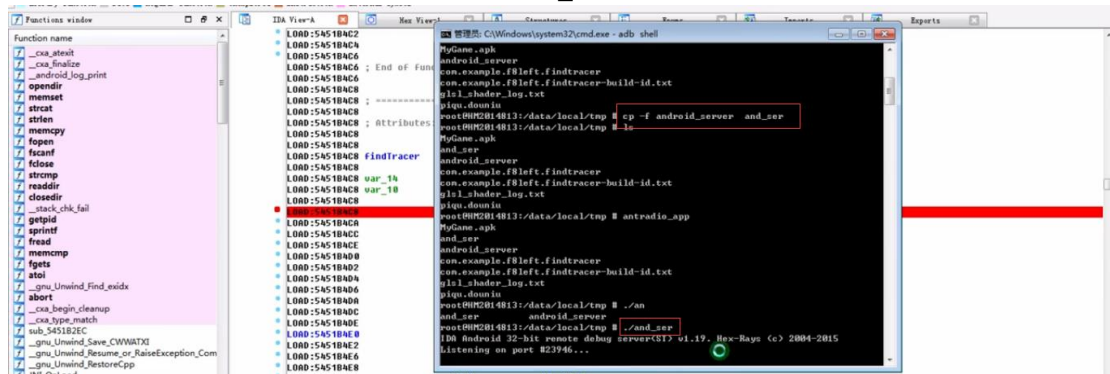
```
管理员: C:\Windows\system32\cmd.exe - adb shell

uptime
version
vmallocinfo
vmstat
zoneinfo
root@HM2014813:/proc # ps | grep "f8"
system 204 1 2492 704 ffffffff b6f828d4 $ /system/bin/rfs_access
system 207 1 2484 800 c011f8dc b6eefec $ /system/bin/qsecond
root 239 1 2368 588 ffffffff b6f8147c $ /bin/otad
shell 258 1 1416 796 c03da2b0 b6f804c4 $ /system/bin/sh
media_rv 284 1 5180 1864 ffffffff b6f804c4 $ /system/bin/sdcard
media_rv 287 1 3812 584 ffffffff b6f874c4 $ /system/bin/sdcard
u0_a63 4830 4786 1512 592 c011f8dc 400c9fec $ su
root 4835 1 1512 308 c011f8dc b6eefec $ /system/xbin/su
root 4836 4835 1512 160 c011f8dc b6eefec $ /system/xbin/su
shell 4914 425 1860 1060 c01f977c b6eaf8d4 $ logcat
u0_a0 4977 212 371564 44400 ffffffff 5451b4d6 t con.example.f8left.findtr
acer
root 5004 4990 1144 172 c011f8dc b6f5cfec $ su
root 5008 1 1512 308 c011f8dc b6eefec $ /system/xbin/su
root 5010 5008 1512 160 c011f8dc b6eefec $ /system/xbin/su
root 5112 5106 1144 172 c011f8dc b6eefec $ su
root 5116 1 1512 308 c011f8dc b6eefec $ /system/xbin/su
root 5118 5116 1512 160 c011f8dc b6eefec $ /system/xbin/su
root@HM2014813:/proc #
```

```
管理员: C:\Windows\system32\cmd.exe - adb shell

anon
mountinfo
mounts
mountstats
net
ns
oon_adj
oon_score
oon_score_adj
pagenap
personality
root
schedstat
sessionid
snaps
stack
stat
statm
status
syscall
task
uchan
root@HM2014813:/proc/4977 # cat cmdline
con.example.f8left.findtracer
root@HM2014813:/proc/4977 #
```

根据代码可以得出绕过的方法：修改 android_server 的名字



此外，调试检测还会检测 stat 文件中的内容（函数 2）

