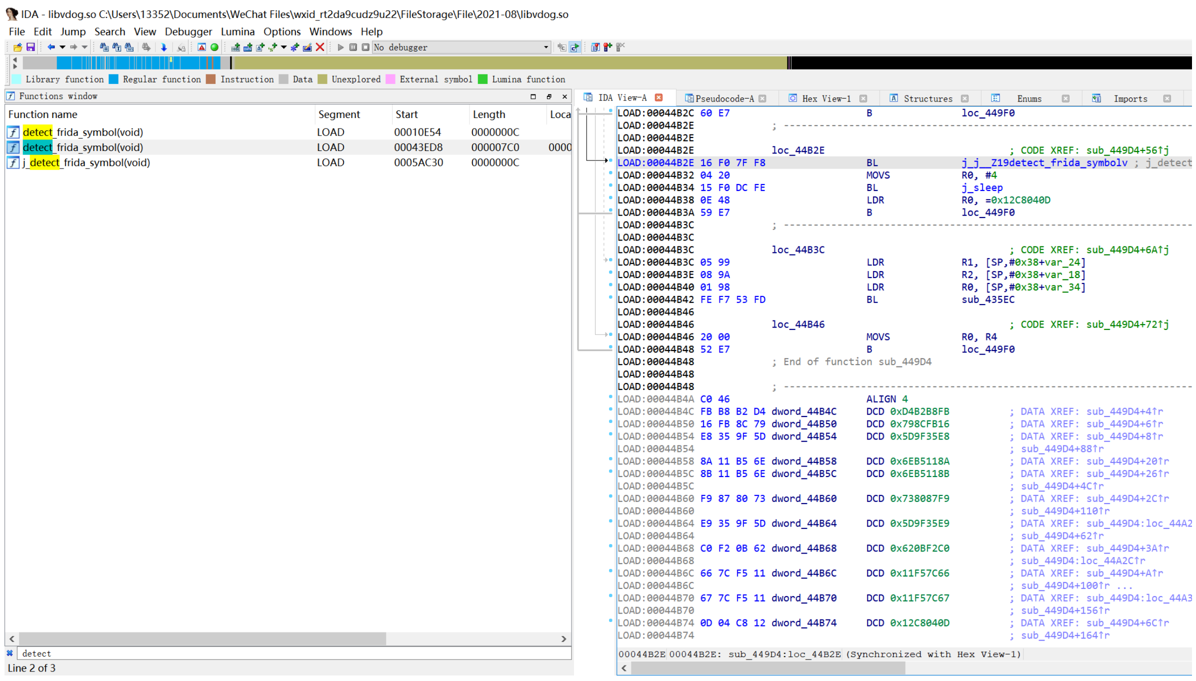
一、前言

朋友遇到了一个样本，说有Frida检测，用反检测版本的Frida Server也无济于事，真让人期待呀，我们来看看吧。首先朋友告知我，检测 发生在libvdog.so中，且函数名中有明显的线索。


IDA - libvdog.so C:\Users\13352\Documents\WeChat Files\wxid_rt2da9cudz9u22\FileStorage\File\2021-08\libvdog.so

Library function | Regular function | Instruction | Data | Unexplored | External symbol | Lumina function

Functions window

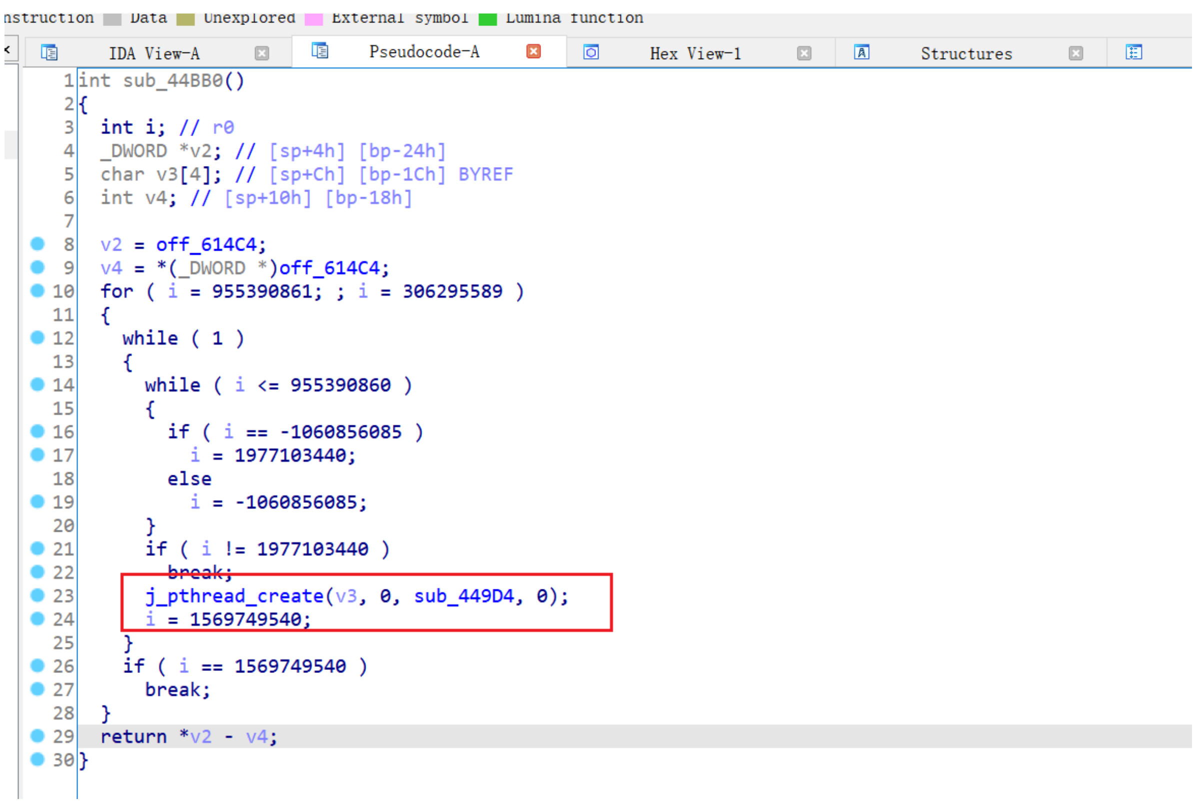
Function name	Segment	Start	Length	Loca
detect_frida_symbol(void)	LOAD	00010E54	0000000C	
detect_frida_symbol(void)	LOAD	00043ED8	000007C0	0000
j_detect_frida_symbol(void)	LOAD	0005AC30	0000000C	

detect

Line 2 of 3

```
LOAD:00044B2C 60 E7                    B               loc_449F0
LOAD:00044B2E                 ; ---------------------------------------------------------------
LOAD:00044B2E
LOAD:00044B2E                 loc_44B2E                               ; CODE XREF: sub_449D4+56↑j
LOAD:00044B2E 16 F0 7F F8              BL              j_j__Z19detect_frida_symbolv ; j_detect
LOAD:00044B32 04 20                    MOVS            R0, #4
LOAD:00044B34 15 F0 DC FE              BL              j_sleep
LOAD:00044B38 0E 48                    LDR             R0, =0x12C8040D
LOAD:00044B3A 59 E7                    B               loc_449F0
LOAD:00044B3C                 ; ---------------------------------------------------------------
LOAD:00044B3C
LOAD:00044B3C                 loc_44B3C                               ; CODE XREF: sub_449D4+6A↑j
LOAD:00044B3C 05 99                    LDR             R1, [SP,#0x38+var_24]
LOAD:00044B3E 08 9A                    LDR             R2, [SP,#0x38+var_18]
LOAD:00044B40 01 98                    LDR             R0, [SP,#0x38+var_34]
LOAD:00044B42 FE F7 53 FD              BL              sub_435EC
LOAD:00044B46
LOAD:00044B46                 loc_44B46                               ; CODE XREF: sub_449D4+72↑j
LOAD:00044B46 20 00                    MOVS            R0, R4
LOAD:00044B48 52 E7                    B               loc_449F0
LOAD:00044B48                 ; End of function sub_449D4
LOAD:00044B48
LOAD:00044B48                 ; ---------------------------------------------------------------
LOAD:00044B4A C0 46                    ALIGN 4
LOAD:00044B4C FB B8 B2 D4 dword_44B4C  DCD 0xD4B2B8FB            ; DATA XREF: sub_449D4+4↑r
LOAD:00044B50 16 FB 8C 79 dword_44B50  DCD 0x798CFB16            ; DATA XREF: sub_449D4+6↑r
LOAD:00044B54 E8 35 9F 5D dword_44B54  DCD 0x5D9F35E8            ; DATA XREF: sub_449D4+8↑r
LOAD:00044B54                                                    ; sub_449D4+88↑r
LOAD:00044B58 8A 11 B5 6E dword_44B58  DCD 0x6EB5118A            ; DATA XREF: sub_449D4+20↑r
LOAD:00044B5C 8B 11 B5 6E dword_44B5C  DCD 0x6EB5118B            ; DATA XREF: sub_449D4+26↑r
LOAD:00044B5C                                                    ; sub_449D4+4C↑r
LOAD:00044B60 F9 87 80 73 dword_44B60  DCD 0x738087F9            ; DATA XREF: sub_449D4+2C↑r
LOAD:00044B60                                                    ; sub_449D4+110↑r
LOAD:00044B64 E9 35 9F 5D dword_44B64  DCD 0x5D9F35E9            ; DATA XREF: sub_449D4:loc_44A2
LOAD:00044B64                                                    ; sub_449D4+62↑r
LOAD:00044B68 C0 F2 0B 62 dword_44B68  DCD 0x620BF2C0            ; DATA XREF: sub_449D4+3A↑r
LOAD:00044B68                                                    ; sub_449D4:loc_44A2C↑r
LOAD:00044B6C 66 7C F5 11 dword_44B6C  DCD 0x11F57C66            ; DATA XREF: sub_449D4+A↑r
LOAD:00044B6C                                                    ; sub_449D4+100↑r ...
LOAD:00044B70 67 7C F5 11 dword_44B70  DCD 0x11F57C67            ; DATA XREF: sub_449D4:loc_44A3
LOAD:00044B70                                                    ; sub_449D4+156↑r
LOAD:00044B74 0D 04 C8 12 dword_44B74  DCD 0x12C8040D            ; DATA XREF: sub_449D4+6C↑r
LOAD:00044B74                                                    ; sub_449D4+164↑r
00044B2E 00044B2E: sub_449D4:loc_44B2E (Synchronized with Hex View-1)
```

既然是反Frida，那肯定得开个线程轮询检测。通过交叉引用静态分析一下这个detect_frida_symbol。

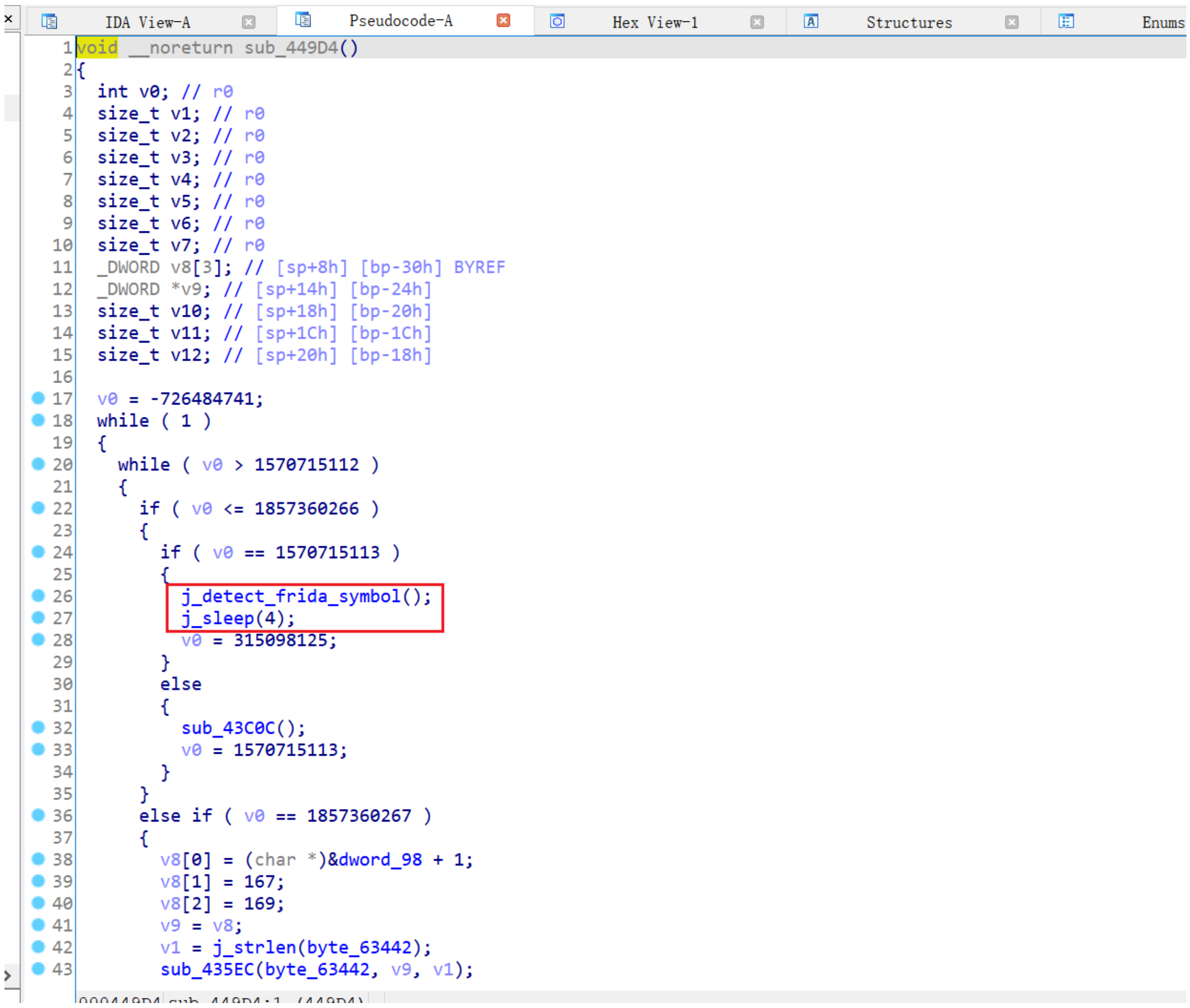

```
int sub_44BB0()
{
  int i; // r0
  _DWORD *v2; // [sp+4h] [bp-24h]
  char v3[4]; // [sp+Ch] [bp-1Ch] BYREF
  int v4; // [sp+10h] [bp-18h]

  v2 = off_614C4;
  v4 = *(_DWORD *)off_614C4;
  for ( i = 955390861; ; i = 306295589 )
  {
    while ( 1 )
    {
      while ( i <= 955390860 )
      {
        if ( i == -1060856085 )
          i = 1977103440;
        else
          i = -1060856085;
      }
      if ( i != 1977103440 )
        break;
      j_pthread_create(v3, 0, sub_449D4, 0);
      i = 1569749540;
    }
    if ( i == 1569749540 )
      break;
  }
  return *v2 - v4;
}
```

第一个参数为指向线程 标识符的 指针。

第二个参数用来设置线程属性。

第三个参数是线程运行函数的起始地址。

最后一个参数是运行函数的参数。



```
void __noreturn sub_449D4()
{
  int v0; // r0
  size_t v1; // r0
  size_t v2; // r0
  size_t v3; // r0
  size_t v4; // r0
  size_t v5; // r0
  size_t v6; // r0
  size_t v7; // r0
  _DWORD v8[3]; // [sp+8h] [bp-30h] BYREF
  _DWORD *v9; // [sp+14h] [bp-24h]
  size_t v10; // [sp+18h] [bp-20h]
  size_t v11; // [sp+1Ch] [bp-1Ch]
  size_t v12; // [sp+20h] [bp-18h]

  v0 = -726484741;
  while ( 1 )
  {
    while ( v0 > 1570715112 )
    {
      if ( v0 <= 1857360266 )
      {
        if ( v0 == 1570715113 )
        {
          j_detect_frida_symbol();
          j_sleep(4);
          v0 = 315098125;
        }
        else
        {
          sub_43C0C();
          v0 = 1570715113;
        }
      }
      else if ( v0 == 1857360267 )
      {
        v8[0] = (char *)&dword_98 + 1;
        v8[1] = 167;
        v8[2] = 169;
        v9 = v8;
        v1 = j_strlen(byte_63442);
        sub_435EC(byte_63442, v9, v1);
```

我们来用Unidbg分析它吧!

二、准备

样本中使用到了多线程，但Unidbg对多线程的实现和支持较差，unidbgMutilThread是一位大神实现的，支持ARM32下多线程的Unidbg版本，我们在这里使用它。 = =

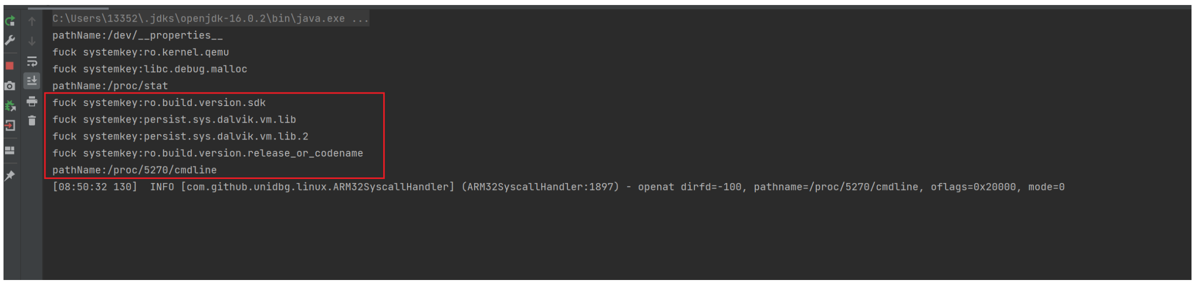
这是我们的基础代码，检测或者反调试的样本，读取系统信息是很常见的，所以进行了文件重定向以及systemPropertyHook。

```java
package com.detectFrida;

import com.github.unidbg.AndroidEmulator;
import com.github.unidbg.Emulator;
import com.github.unidbg.Module;
import com.github.unidbg.file.FileResult;
import com.github.unidbg.file.IOResolver;
import com.github.unidbg.linux.android.AndroidARMEmulator;
import com.github.unidbg.linux.android.AndroidResolver;
import com.github.unidbg.linux.android.SystemPropertyHook;
import com.github.unidbg.linux.android.SystemPropertyProvider;
import com.github.unidbg.linux.android.dvm.AbstractJni;
import com.github.unidbg.linux.android.dvm.DalvikModule;
import com.github.unidbg.linux.android.dvm.VM;
import com.github.unidbg.memory.Memory;

import java.io.File;
```

```java
public class vdog extends AbstractJni implements IOResolver {
    private final AndroidEmulator emulator;
    private final VM vm;
    private final Module module;

    vdog() {
        // 创建模拟器实例，要模拟32位或者64位，在这里区分
        emulator = new AndroidARMEmulator(new File("target/rootfs"));
        // 模拟器的内存操作接口
        final Memory memory = emulator.getMemory();
        // 设置系统类库解析
        memory.setLibraryResolver(new AndroidResolver(23));
        // 创建Android虚拟机
        vm = emulator.createDalvikVM(new File("unidbg-
android/src/test/resources/vdog/459032971C6EFC9DEECA8734C99C34BB.apk"));

        // 设置是否打印Jni调用细节
        vm.setVerbose(true);

        emulator.getSyscallHandler().addIOResolver(this);

        SystemPropertyHook systemPropertyHook = new
SystemPropertyHook(emulator);
        systemPropertyHook.setPropertyProvider(new SystemPropertyProvider() {
            @Override
            public String getProperty(String key) {
                System.out.println("fuck systemkey:"+key);
                switch (key){

                }
                return "";
            };
        });
        memory.addHookListener(systemPropertyHook);

        // 加载so到虚拟内存，加载成功以后会默认调用init_array等函数
        DalvikModule dm = vm.loadLibrary(new File("unidbg-
android/src/test/resources/vdog/libvdog.so"), true);
        // 加载好的 libscmain.so对应为一个模块
        module = dm.getModule();
        // 设置JNI
        vm.setJni(this);
    }
    public static void main(String[] args) {
        vdog demo = new vdog();
    }

    @Override
    public FileResult resolve(Emulator emulator, String pathname, int oflags) {
        System.out.println("pathName:"+pathname);
        return null;
    }
}
```

三、Unidbg实战

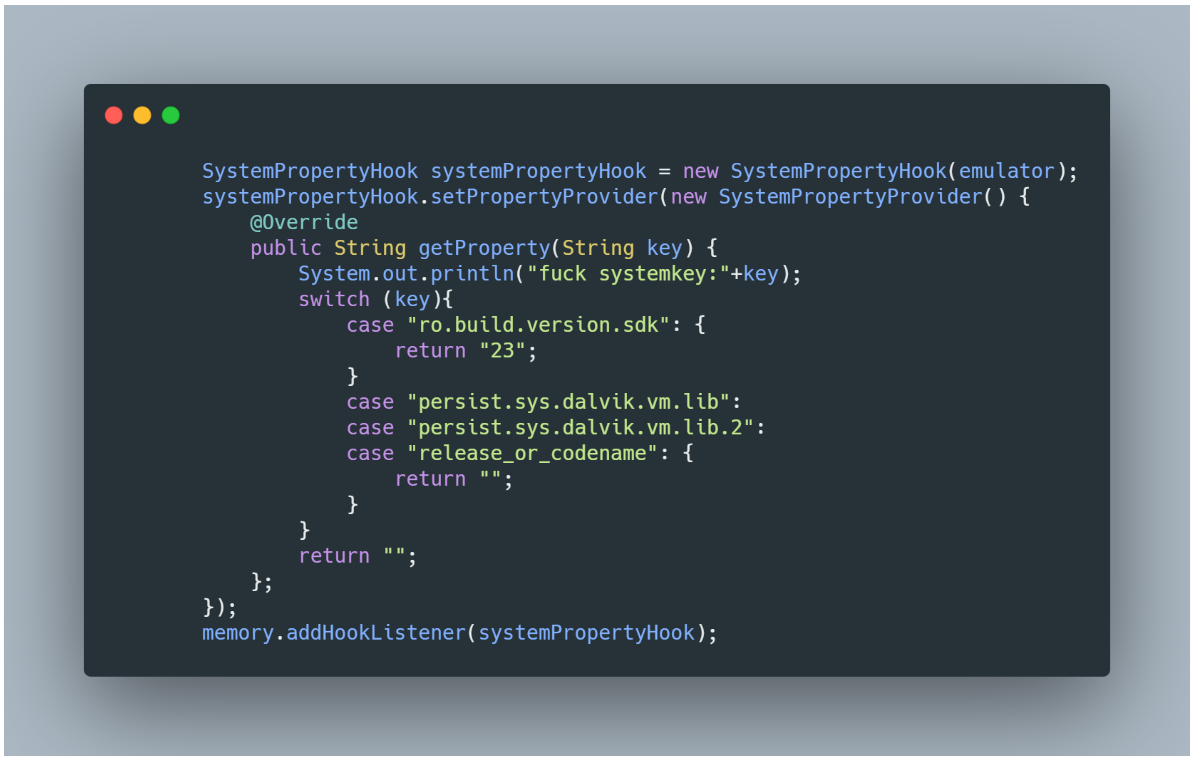
运行代码



```
C:\Users\13352\.jdks\openjdk-16.0.2\bin\java.exe ...
pathName:/dev/__properties__
fuck systemkey:ro.kernel.qemu
fuck systemkey:libc.debug.malloc
pathName:/proc/stat
fuck systemkey:ro.build.version.sdk
fuck systemkey:persist.sys.dalvik.vm.lib
fuck systemkey:persist.sys.dalvik.vm.lib.2
fuck systemkey:ro.build.version.release_or_codename
pathName:/proc/5270/cmdline
[08:50:32 130]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1897) - openat dirfd=-100, pathname=/proc/5270/cmdline, oflags=0x20000, mode=0
```

前文我们说过，前四个是不用管的，所以只用处理红框中的这些。

系统属性，可以通过*adb shell* getprop 获取后填进去，如果你的测试机并没有对应的属性，那就返回空字符串即可。



```java
SystemPropertyHook systemPropertyHook = new SystemPropertyHook(emulator);
systemPropertyHook.setPropertyProvider(new SystemPropertyProvider() {
    @Override
    public String getProperty(String key) {
        System.out.println("fuck systemkey:"+key);
        switch (key){
            case "ro.build.version.sdk": {
                return "23";
            }
            case "persist.sys.dalvik.vm.lib":
            case "persist.sys.dalvik.vm.lib.2":
            case "release_or_codename": {
                return "";
            }
        }
        return "";
    };
});
memory.addHookListener(systemPropertyHook);
```

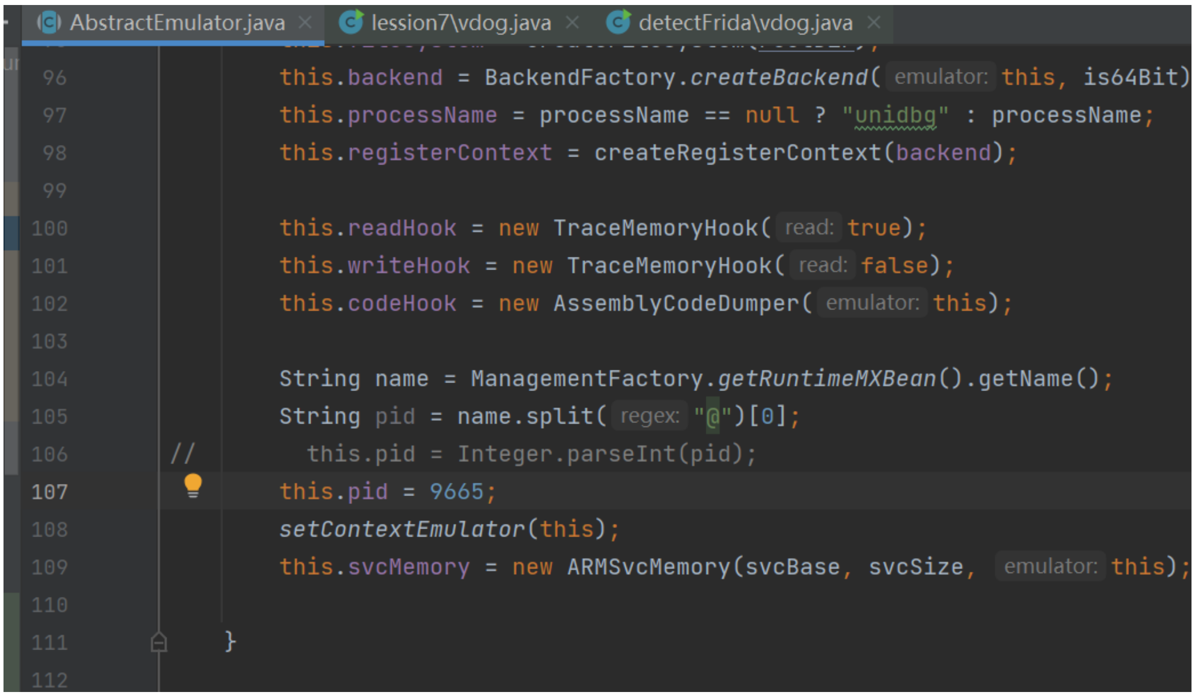
对 */proc/5270/cmdline* 的文件访问呢？样本一我们提到过，这是为了获取进程名，从而确定自己是否被重打包运行在不安全的环境中。

我们最好做一下PID的固定，PID一直变动对我们补环境可能有影响，最好将其固定成测试机中真实运行的样本PID。

下面演示一下这个过程，首先根据包名找到样本进程

```
polaris:/ $ ps -A | grep sfex
u0_a378       9665   690 1707336 274444 0                   0 S
com.sfexpress.merchant
u0_a378       9699   690 1413648 107840 0                   0 S
com.sfexpress.merchant:v5chat
polaris:/ $ su
polaris:/ # cat /proc/9665/cmdline
com.sfexpress.merchantpolaris:/ #
```

src/main/java/com/github/unidbg/AbstractEmulator.java 中修改进程名



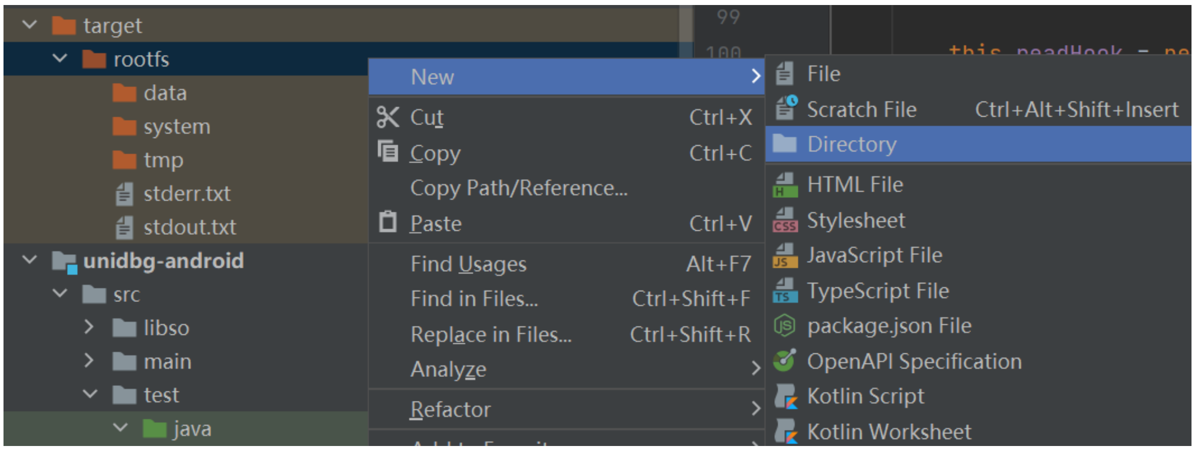
```
        this.backend = BackendFactory.createBackend( emulator: this, is64Bit)
        this.processName = processName == null ? "unidbg" : processName;
        this.registerContext = createRegisterContext(backend);

        this.readHook = new TraceMemoryHook( read: true);
        this.writeHook = new TraceMemoryHook( read: false);
        this.codeHook = new AssemblyCodeDumper( emulator: this);

        String name = ManagementFactory.getRuntimeMXBean().getName();
        String pid = name.split( regex: "@")[0];
//        this.pid = Integer.parseInt(pid);
        this.pid = 9665;
        setContextEmulator(this);
        this.svcMemory = new ARMSvcMemory(svcBase, svcSize,  emulator: this);

    }
```

接下来，我们不用代码方式去补文件访问，而是rootfs虚拟文件系统。

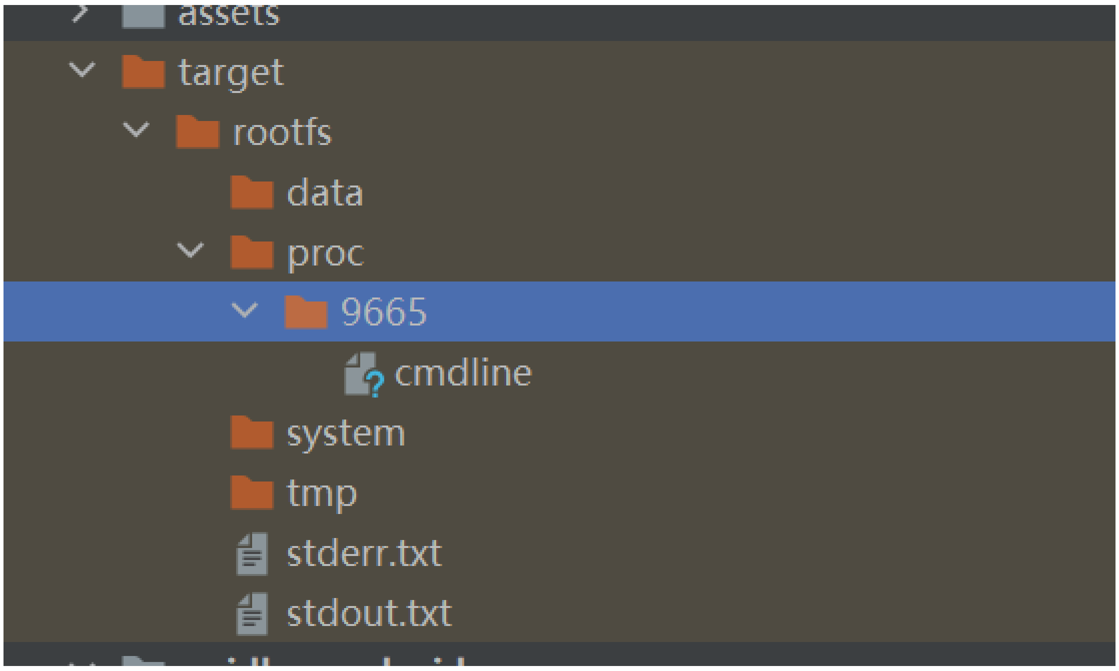
首先将android测试机里的cmdline 复制到sdcard，然后adb pull 出来

```
com.sfexpress.merchantpolaris:/ # cp /proc/9665/cmdline /sdcard
```

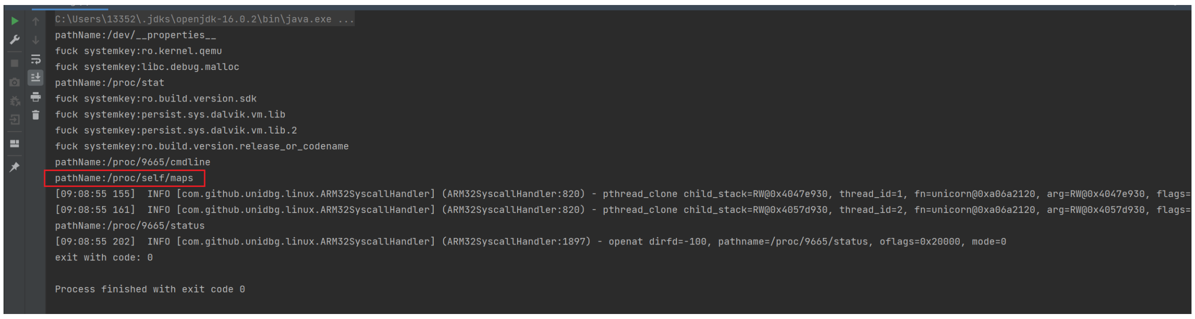
```
adb pull /sdcard/cmdline yourpath
```



在rootfs中新建proc目录以及9665子目录，再把从测试机中拖出来的cmdline放进去



大功告成，继续运行

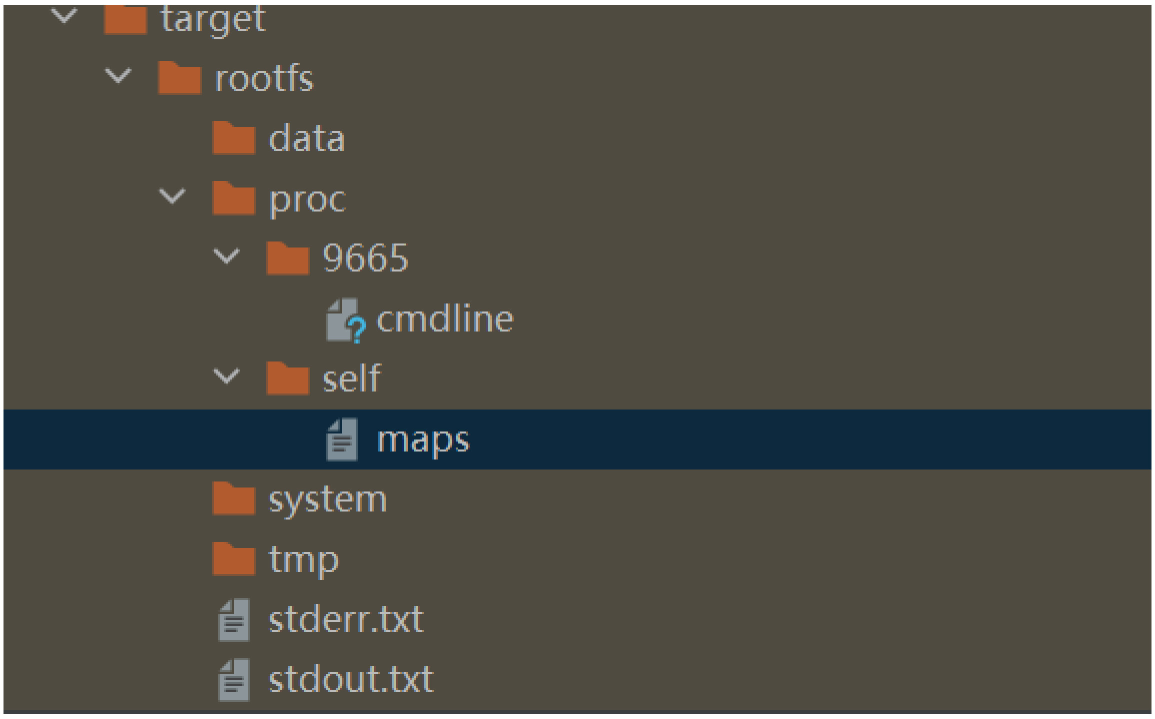


```
C:\Users\13352\.jdks\openjdk-16.0.2\bin\java.exe ...
pathName:/dev/__properties__
fuck systemkey:ro.kernel.qemu
fuck systemkey:libc.debug.malloc
pathName:/proc/stat
fuck systemkey:ro.build.version.sdk
fuck systemkey:persist.sys.dalvik.vm.lib
fuck systemkey:persist.sys.dalvik.vm.lib.2
fuck systemkey:ro.build.version.release_or_codename
pathName:/proc/9665/cmdline
pathName:/proc/self/maps
[09:08:55 155]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:820) - pthread_clone child_stack=RW@0x4047e930, thread_id=1, fn=unicorn@0xa06a2120, arg=RW@0x4047e930, flags=
[09:08:55 161]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:820) - pthread_clone child_stack=RW@0x4057d930, thread_id=2, fn=unicorn@0xa06a2120, arg=RW@0x4057d930, flags=
pathName:/proc/9665/status
[09:08:55 202]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1897) - openat dirfd=-100, pathname=/proc/9665/status, oflags=0x20000, mode=0
exit with code: 0

Process finished with exit code 0
```

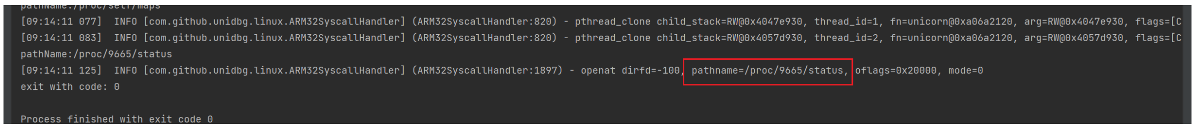
读取maps，如法炮制，记得此时需要使样本APP一直处于测试机界面中

```
polaris:/ # cp /proc/self/maps /sdcard
```

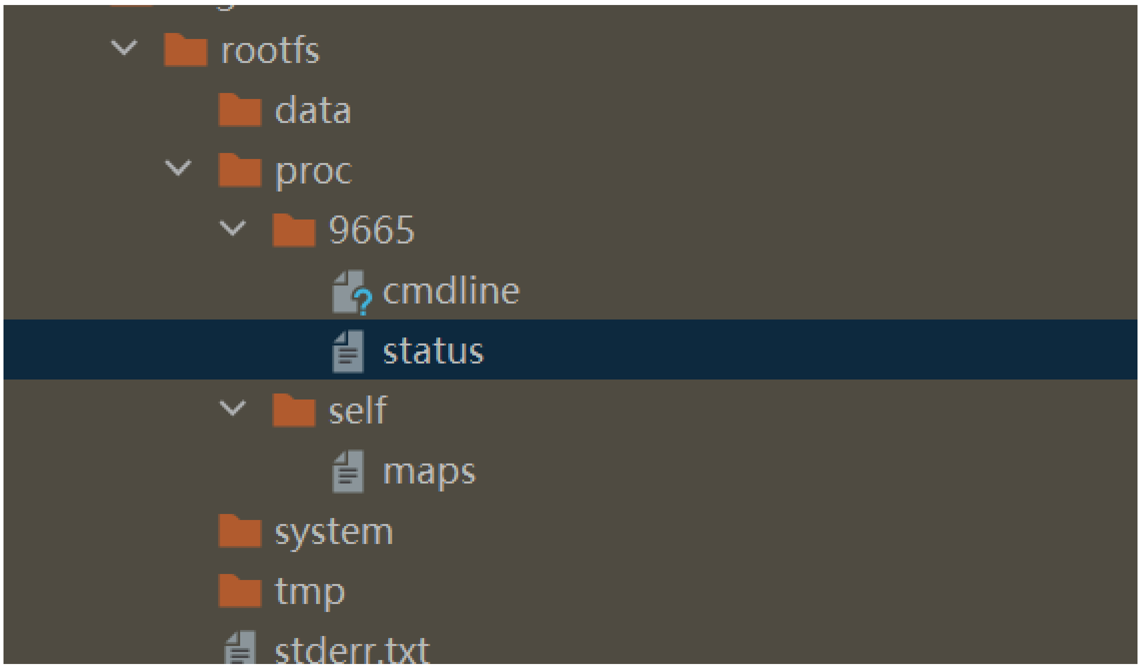
```
adb pull /sdcard/maps yourpath
```



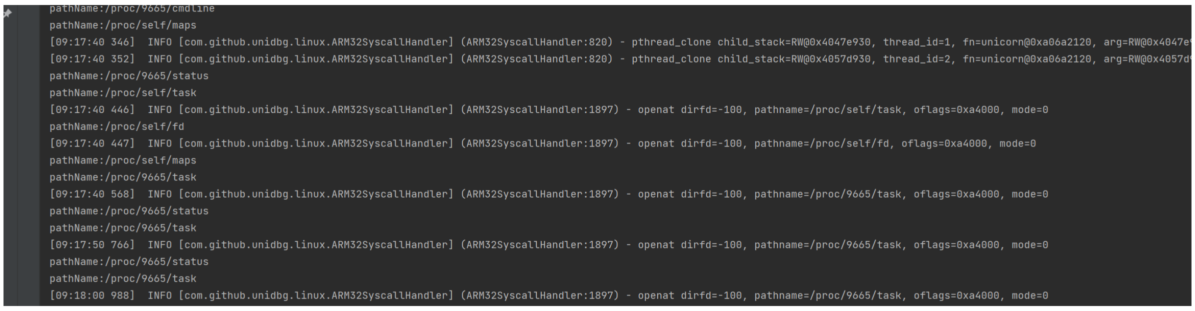
接下来读取status文件，前文我们也说过，status中的ptrace端口状态是反调试的基本方式。



```
pathName:/proc/self/maps
[09:14:11 077]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:820) - pthread_clone child_stack=RW@0x4047e930, thread_id=1, fn=unicorn@0xa06a2120, arg=RW@0x4047e930, flags=[C
[09:14:11 083]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:820) - pthread_clone child_stack=RW@0x4057d930, thread_id=2, fn=unicorn@0xa06a2120, arg=RW@0x4057d930, flags=[C
pathName:/proc/9665/status
[09:14:11 125]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1897) - openat dirfd=-100, pathname=/proc/9665/status, oflags=0x20000, mode=0
exit with code: 0

Process finished with exit code 0
```



继续运行



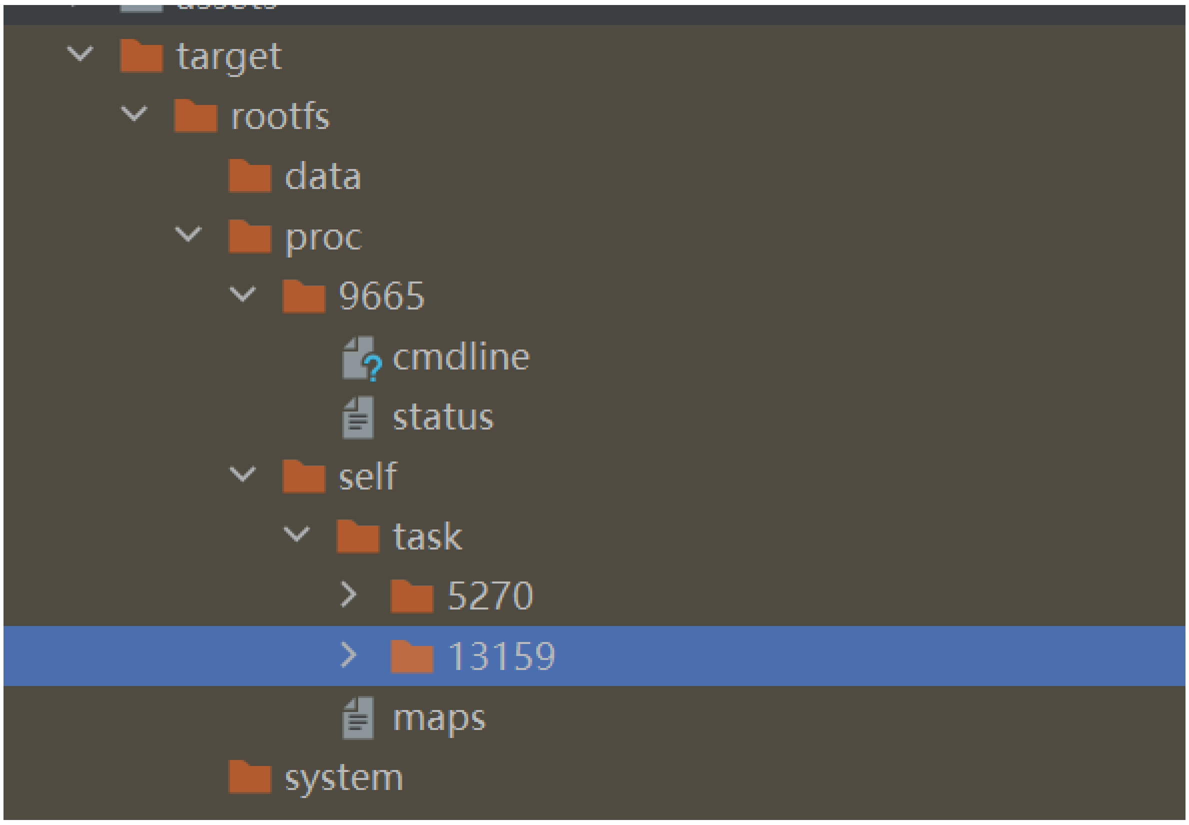
```
pathName:/proc/9665/cmdline
pathName:/proc/self/maps
[09:17:40 346]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:820) - pthread_clone child_stack=RW@0x4047e930, thread_id=1, fn=unicorn@0xa06a2120, arg=RW@0x4047e
[09:17:40 352]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:820) - pthread_clone child_stack=RW@0x4057d930, thread_id=2, fn=unicorn@0xa06a2120, arg=RW@0x4057d
pathName:/proc/9665/status
pathName:/proc/self/task
[09:17:40 446]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1897) - openat dirfd=-100, pathname=/proc/self/task, oflags=0xa4000, mode=0
pathName:/proc/self/fd
[09:17:40 447]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1897) - openat dirfd=-100, pathname=/proc/self/fd, oflags=0xa4000, mode=0
pathName:/proc/self/maps
pathName:/proc/9665/task
[09:17:40 568]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1897) - openat dirfd=-100, pathname=/proc/9665/task, oflags=0xa4000, mode=0
pathName:/proc/9665/status
pathName:/proc/9665/task
[09:17:50 766]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1897) - openat dirfd=-100, pathname=/proc/9665/task, oflags=0xa4000, mode=0
pathName:/proc/9665/status
pathName:/proc/9665/task
[09:18:00 988]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1897) - openat dirfd=-100, pathname=/proc/9665/task, oflags=0xa4000, mode=0
```

此时程序运行已经不带停的了，看来已经进入子线程中对Frida的轮询检查。

每次都有三个文件访问，先从第一个补

```
polaris:/ # cp -r /proc/self/task /sdcard
cp: /sdcard/task/13159/fd/0: Operation not permitted
cp: /sdcard/task/13159/fd/1: Operation not permitted
cp: /sdcard/task/13159/fd/2: Operation not permitted
cp: /sdcard/task/13159/fd/3: Operation not permitted
cp: /sdcard/task/13159/fd/4: Operation not permitted
cp: /sdcard/task/13159/fd/5: Operation not permitted
cp: /sdcard/task/13159/fd/6: Operation not permitted
cp: /sdcard/task/13159/fd/7: Operation not permitted
cp: /sdcard/task/13159/fd/8: Operation not permitted
cp: /sdcard/task/13159/ns/net: Operation not permitted
cp: /sdcard/task/13159/ns/mnt: Operation not permitted
cp: /sdcard/task/13159/ns/cgroup: Operation not permitted
```

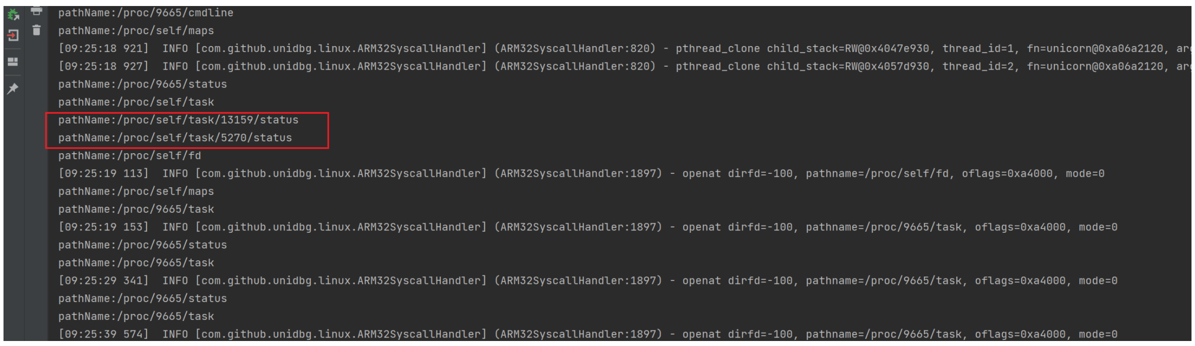
有些没拷贝成功，那也没办法



解释一下task文件夹的内容

> 该目录包含的是进程中的每一个线程.每一个目录的名字是以线程ID命名的(tid).在每一个tid下面的目录结构与`/proc/pid`下面的目录结构相同.

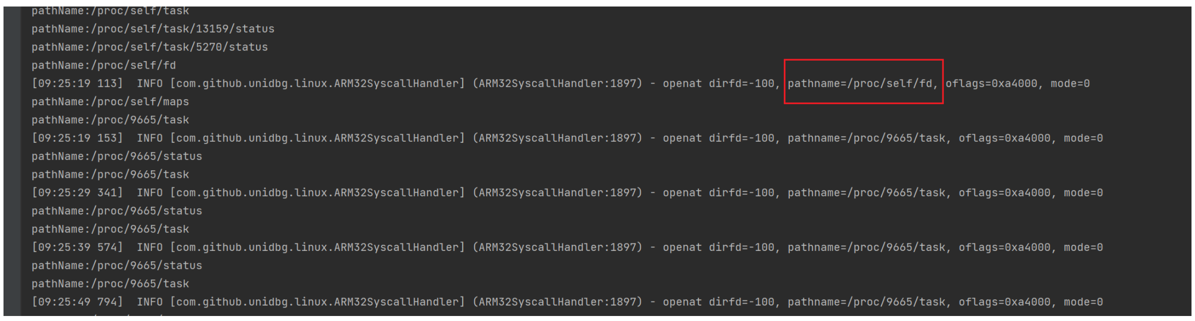
再运行试试



```
pathName:/proc/9665/cmdline
pathName:/proc/self/maps
[09:25:18 921]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:820) - pthread_clone child_stack=RW@0x4047e930, thread_id=1, fn=unicorn@0xa06a2120, ar
[09:25:18 927]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:820) - pthread_clone child_stack=RW@0x4057d930, thread_id=2, fn=unicorn@0xa06a2120, ar
pathName:/proc/9665/status
pathName:/proc/self/task
pathName:/proc/self/task/13159/status
pathName:/proc/self/task/5270/status
pathName:/proc/self/fd
[09:25:19 113]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1897) - openat dirfd=-100, pathname=/proc/self/fd, oflags=0xa4000, mode=0
pathName:/proc/self/maps
pathName:/proc/9665/task
[09:25:19 153]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1897) - openat dirfd=-100, pathname=/proc/9665/task, oflags=0xa4000, mode=0
pathName:/proc/9665/status
pathName:/proc/9665/task
[09:25:29 341]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1897) - openat dirfd=-100, pathname=/proc/9665/task, oflags=0xa4000, mode=0
pathName:/proc/9665/status
pathName:/proc/9665/task
[09:25:39 574]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1897) - openat dirfd=-100, pathname=/proc/9665/task, oflags=0xa4000, mode=0
```

刚给它传入的task，样本访问了其中每个线程的status文件，这是什么原因呢，推测还是检测TracerPid。

> /proc/pid/status 和 /proc/pid/task/pid/status：普通状态下，TracerPid这项应该为0；调试状态下为调试进程的PID。



```
pathName:/proc/self/task
pathName:/proc/self/task/13159/status
pathName:/proc/self/task/5270/status
pathName:/proc/self/fd
[09:25:19 113]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1897) - openat dirfd=-100, pathname=/proc/self/fd, oflags=0xa4000, mode=0
pathName:/proc/self/maps
pathName:/proc/9665/task
[09:25:19 153]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1897) - openat dirfd=-100, pathname=/proc/9665/task, oflags=0xa4000, mode=0
pathName:/proc/9665/status
pathName:/proc/9665/task
[09:25:29 341]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1897) - openat dirfd=-100, pathname=/proc/9665/task, oflags=0xa4000, mode=0
pathName:/proc/9665/status
pathName:/proc/9665/task
[09:25:39 574]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1897) - openat dirfd=-100, pathname=/proc/9665/task, oflags=0xa4000, mode=0
pathName:/proc/9665/status
pathName:/proc/9665/task
[09:25:49 794]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1897) - openat dirfd=-100, pathname=/proc/9665/task, oflags=0xa4000, mode=0
```

接下来是它

> /proc/pid/fd 是一个子目录,包含了当前进程打开的每一个文件.每一个条目都是一个文件描述符,是一个符号链接,指向的是实际打开的地址

即fd目录下应该是一堆文件描述符，并且这些文件描述符导向真正的文件。这可有点难办了。

首先文件描述符没法复制，复制过来也不能把其对应的“真正文件信息”一并复制过来，我们来看一下这些文件描述符指向哪儿

```
polaris:/ $ su
polaris:/ # cd /proc/9665/fd
polaris:/proc/9665/fd # ls
0   104 110 125 132 139 145 151 158 165 171 186 22 29 35  38 44 50 57  60 67 73
8  86 92 99
1   105 116 126 133 14  146 152 159 166 173 188 23 3  353 39 45 51 576 61 68 74
80 87 93
10  106 117 128 134 140 147 153 16  167 174 189 24 30 354 4  46 52 577 62 69 75
81 88 94
100 107 118 129 135 141 148 154 160 168 176 19  25 31 355 40 47 53 578 63 7  76
82 89 95
101 108 119 13  136 142 149 155 162 169 177 2   26 32 356 41 48 54 58  64 70 77
83 9  96
102 109 12  130 137 143 15  156 163 17  18  20  27 33 36  42 49 55 59  65 71 78
84 90 97
103 11  124 131 138 144 150 157 164 170 183 21  28 34 37  43 5  56 6   66 72 79
85 91 98
polaris:/proc/9665/fd # ls -l
total 0
lrwx------ 1 root root 64 2021-08-18 08:59 0 -> /dev/null
lrwx------ 1 root root 64 2021-08-18 08:59 1 -> /dev/null
lrwx------ 1 root root 64 2021-08-18 08:59 10 -> /dev/binder
lrwx------ 1 root root 64 2021-08-18 08:59 100 -> anon_inode:[eventpoll]
lrwx------ 1 root root 64 2021-08-18 08:59 101 -> socket:[2679332]
l-wx------ 1 root root 64 2021-08-18 08:59 103 ->
/sys/kernel/debug/tracing/trace_marker
lrwx------ 1 root root 64 2021-08-18 08:59 105 -> anon_inode:[eventfd]
lrwx------ 1 root root 64 2021-08-18 08:59 106 -> socket:[2678106]
lrwx------ 1 root root 64 2021-08-18 08:59 107 -> socket:[2678108]
lrwx------ 1 root root 64 2021-08-18 08:59 109 -> anon_inode:[eventpoll]
lr-x------ 1 root root 64 2021-08-18 08:59 11 ->
/apex/com.android.runtime/javalib/core-oj.jar
lrwx------ 1 root root 64 2021-08-18 08:59 110 -> anon_inode:[eventfd]
lrwx------ 1 root root 64 2021-08-18 11:29 112 -> socket:[2668324]
lr-x------ 1 root root 64 2021-08-18 08:59 116 -> /dev/ion
lrwx------ 1 root root 64 2021-08-18 08:59 117 -> /dev/kgsl-3d0
lr-x------ 1 root root 64 2021-08-18 08:59 118 -> /dev/ion
lrwx------ 1 root root 64 2021-08-18 08:59 119 -> /dev/hwbinder
lr-x------ 1 root root 64 2021-08-18 08:59 12 ->
/apex/com.android.runtime/javalib/core-libart.jar
lr-x------ 1 root root 64 2021-08-18 08:59 124 ->
/system/product/app/WebViewGoogle/WebViewGoogle.apk
lr-x------ 1 root root 64 2021-08-18 08:59 125 ->
/system/product/app/TrichromeLibrary/TrichromeLibrary.apk
lrwx------ 1 root root 64 2021-08-18 08:59 126 -> /dev/ashmem
lr-x------ 1 root root 64 2021-08-18 08:59 128 ->
/system/product/app/TrichromeLibrary/TrichromeLibrary.apk
lr-x------ 1 root root 64 2021-08-18 08:59 129 ->
/system/product/app/WebViewGoogle/WebViewGoogle.apk
lr-x------ 1 root root 64 2021-08-18 08:59 13 ->
/apex/com.android.runtime/javalib/okhttp.jar
lrwx------ 1 root root 64 2021-08-18 08:59 130 -> socket:[2678562]
```

```
lr-x------ 1 root root 64 2021-08-18 08:59 131 ->
/system/product/app/WebViewGoogle/WebViewGoogle.apk
lrwx------ 1 root root 64 2021-08-18 08:59 132 ->
/data/data/com.sfexpress.merchant/app_webview/webview_data.lock
lrwx------ 1 root root 64 2021-08-18 08:59 133 -> anon_inode:[eventfd]
lrwx------ 1 root root 64 2021-08-18 08:59 134 -> anon_inode:[eventpoll]
lr-x------ 1 root root 64 2021-08-18 08:59 135 ->
/system/product/app/TrichromeLibrary/TrichromeLibrary.apk
lr-x------ 1 root root 64 2021-08-18 08:59 136 ->
/system/product/app/WebViewGoogle/WebViewGoogle.apk
lr-x------ 1 root root 64 2021-08-18 08:59 137 ->
/system/product/app/TrichromeLibrary/TrichromeLibrary.apk
lrwx------ 1 root root 64 2021-08-18 08:59 138 -> anon_inode:[eventfd]
lrwx------ 1 root root 64 2021-08-18 08:59 139 -> anon_inode:[timerfd]
lr-x------ 1 root root 64 2021-08-18 08:59 14 ->
/apex/com.android.runtime/javalib/bouncycastle.jar
lr-x------ 1 root root 64 2021-08-18 08:59 140 ->
/system/product/app/WebViewGoogle/WebViewGoogle.apk
lr-x------ 1 root root 64 2021-08-18 08:59 141 ->
/system/product/app/WebViewGoogle/WebViewGoogle.apk
lr-x------ 1 root root 64 2021-08-18 08:59 142 ->
/system/product/app/WebViewGoogle/WebViewGoogle.apk
lrwx------ 1 root root 64 2021-08-18 08:59 143 -> anon_inode:[eventpoll]
lr-x------ 1 root root 64 2021-08-18 08:59 144 -> /dev/urandom
lrwx------ 1 root root 64 2021-08-18 08:59 145 -> socket:[2675202]
lrwx------ 1 root root 64 2021-08-18 08:59 146 -> socket:[2675203]
lr-x------ 1 root root 64 2021-08-18 08:59 147 -> pipe:[2675204]
l-wx------ 1 root root 64 2021-08-18 08:59 148 -> pipe:[2675204]
lrwx------ 1 root root 64 2021-08-18 08:59 149 -> anon_inode:[eventpoll]
lr-x------ 1 root root 64 2021-08-18 08:59 15 ->
/apex/com.android.runtime/javalib/apache-xml.jar
lrwx------ 1 root root 64 2021-08-18 08:59 150 -> socket:[2679967]
lrwx------ 1 root root 64 2021-08-18 08:59 151 -> socket:[2679968]
lr-x------ 1 root root 64 2021-08-18 08:59 152 -> pipe:[2679969]
l-wx------ 1 root root 64 2021-08-18 08:59 153 -> pipe:[2679969]
lrwx------ 1 root root 64 2021-08-18 08:59 154 -> socket:[2680018]
lrwx------ 1 root root 64 2021-08-18 08:59 155 -> socket:[2680019]
lrwx------ 1 root root 64 2021-08-18 08:59 156 ->
/data/data/com.sfexpress.merchant/app_webview/Default/Web Data
lrwx------ 1 root root 64 2021-08-18 08:59 157 -> socket:[2678347]
lrwx------ 1 root root 64 2021-08-18 08:59 158 -> socket:[2678348]
lr-x------ 1 root root 64 2021-08-18 08:59 159 -> pipe:[2678349]
lr-x------ 1 root root 64 2021-08-18 08:59 16 ->
/system/framework/com.nxp.nfc.nq.jar
l-wx------ 1 root root 64 2021-08-18 08:59 160 -> pipe:[2678349]
lrwx------ 1 root root 64 2021-08-18 08:59 162 -> /dev/ashmem
lrwx------ 1 root root 64 2021-08-18 08:59 163 -> anon_inode:[eventfd]
lrwx------ 1 root root 64 2021-08-18 08:59 164 -> anon_inode:[timerfd]
lrwx------ 1 root root 64 2021-08-18 08:59 165 -> /dev/ashmem
lr-x------ 1 root root 64 2021-08-18 08:59 166 ->
/system/product/app/TrichromeLibrary/TrichromeLibrary.apk
lrwx------ 1 root root 64 2021-08-18 08:59 167 -> anon_inode:[eventpoll]
lr-x------ 1 root root 64 2021-08-18 08:59 168 ->
/system/product/app/WebViewGoogle/WebViewGoogle.apk
lrwx------ 1 root root 64 2021-08-18 08:59 169 -> anon_inode:[eventpoll]
lr-x------ 1 root root 64 2021-08-18 08:59 17 -> /system/framework/framework.jar
lrwx------ 1 root root 64 2021-08-18 08:59 170 -> socket:[2674480]
lrwx------ 1 root root 64 2021-08-18 08:59 171 -> socket:[2674481]
```

```
lr-x------ 1 root root 64 2021-08-18 08:59 173 -> pipe:[2674482]
l-wx------ 1 root root 64 2021-08-18 08:59 174 -> pipe:[2674482]
lrwx------ 1 root root 64 2021-08-18 08:59 176 -> anon_inode:[eventfd]
lrwx------ 1 root root 64 2021-08-18 08:59 177 -> anon_inode:[timerfd]
lr-x------ 1 root root 64 2021-08-18 08:59 18 -> /system/framework/ext.jar
lrwx------ 1 root root 64 2021-08-18 08:59 183 -> /dev/ashmem
lrwx------ 1 root root 64 2021-08-18 08:59 186 ->
/data/data/com.sfexpress.merchant/app_webview/Default/Cookies
lrwx------ 1 root root 64 2021-08-18 08:59 188 -> anon_inode:[eventpoll]
lrwx------ 1 root root 64 2021-08-18 08:59 189 -> /dev/ashmem
lrwx------ 1 root root 64 2021-08-18 08:59 19 -> /dev/null
l-wx------ 1 root root 64 2021-08-18 08:59 2 ->
/data/data/com.sfexpress.merchant/app_bugly/bugly_trace_1629253537605.txt
lrwx------ 1 root root 64 2021-08-18 08:59 20 -> /dev/null
lr-x------ 1 root root 64 2021-08-18 08:59 21 -> /system/framework/telephony-
common.jar
lr-x------ 1 root root 64 2021-08-18 08:59 22 -> /system/framework/voip-
common.jar
lr-x------ 1 root root 64 2021-08-18 08:59 23 -> /system/framework/ims-
common.jar
lr-x------ 1 root root 64 2021-08-18 08:59 24 ->
/system/framework/miuisdk@boot.jar
lr-x------ 1 root root 64 2021-08-18 08:59 25 ->
/system/framework/miuisystemsdk@boot.jar
lr-x------ 1 root root 64 2021-08-18 08:59 26 ->
/system/framework/android.test.base.jar
lr-x------ 1 root root 64 2021-08-18 08:59 27 -> /system/framework/telephony-
ext.jar
lr-x------ 1 root root 64 2021-08-18 08:59 28 -> /system/framework/tcmiface.jar
lr-x------ 1 root root 64 2021-08-18 08:59 29 ->
/system/framework/QPerformance.jar
lrwx------ 1 root root 64 2021-08-18 08:59 3 -> socket:[2668142]
lr-x------ 1 root root 64 2021-08-18 08:59 30 ->
/system/framework/UxPerformance.jar
lr-x------ 1 root root 64 2021-08-18 08:59 31 -> /system/framework/WfdCommon.jar
lr-x------ 1 root root 64 2021-08-18 08:59 32 ->
/apex/com.android.conscrypt/javalib/conscrypt.jar
lr-x------ 1 root root 64 2021-08-18 08:59 33 ->
/apex/com.android.media/javalib/updatable-media.jar
lr-x------ 1 root root 64 2021-08-18 08:59 34 -> /system/framework/framework-
res.apk
lr-x------ 1 root root 64 2021-08-18 08:59 35 ->
/system/product/overlay/GmsCnConfigOverlay.apk
l-wx------ 1 root root 64 2021-08-18 08:59 353 ->
/data/data/com.sfexpress.merchant/app_webview/Default/Session Storage/LOG
lrwx------ 1 root root 64 2021-08-18 08:59 354 ->
/data/data/com.sfexpress.merchant/app_webview/Default/Session Storage/LOCK
l-wx------ 1 root root 64 2021-08-18 08:59 355 ->
/data/data/com.sfexpress.merchant/app_webview/Default/Session Storage/MANIFEST-
000001
l-wx------ 1 root root 64 2021-08-18 08:59 356 ->
/data/data/com.sfexpress.merchant/app_webview/Default/Session Storage/000003.log
lr-x------ 1 root root 64 2021-08-18 08:59 36 ->
/system/product/overlay/GoogleExtServicesConfigOverlay.apk
lr-x------ 1 root root 64 2021-08-18 08:59 37 ->
/vendor/overlay/FrameworksResCommon.apk
lr-x------ 1 root root 64 2021-08-18 08:59 38 ->
/vendor/overlay/DevicesAndroidOverlay.apk
```

```
lr-x------ 1 root root 64 2021-08-18 08:59 39 -> /system/framework/framework-
ext-res/framework-ext-res.apk
l-wx------ 1 root root 64 2021-08-18 08:59 4 -> /dev/pmsg0
lr-x------ 1 root root 64 2021-08-18 08:59 40 ->
/system/app/miuisystem/miuisystem.apk
lr-x------ 1 root root 64 2021-08-18 08:59 41 -> /system/app/miui/miui.apk
lrwx------ 1 root root 64 2021-08-18 08:59 42 -> anon_inode:[eventfd]
lrwx------ 1 root root 64 2021-08-18 08:59 43 -> anon_inode:[eventpoll]
lr-x------ 1 root root 64 2021-08-18 08:59 44 -> /proc/9665/task/9665/delay
lr-x------ 1 root root 64 2021-08-18 08:59 45 -> /data/system/theme/framework-
res
lr-x------ 1 root root 64 2021-08-18 08:59 46 ->
/system/media/theme/default/icons
lr-x------ 1 root root 64 2021-08-18 08:59 47 -> /data/system/theme/lockscreen
lr-x------ 1 root root 64 2021-08-18 08:59 48 ->
/data/app/com.sfexpress.merchant-SqSjnfLvF-MmKY8y4tFh0g==/base.apk
lr-x------ 1 root root 64 2021-08-18 08:59 49 ->
/data/app/com.sfexpress.merchant-SqSjnfLvF-MmKY8y4tFh0g==/base.apk
l-wx------ 1 root root 64 2021-08-18 08:59 5 ->
/sys/kernel/debug/tracing/trace_marker
lr-x------ 1 root root 64 2021-08-18 08:59 50 ->
/apex/com.android.runtime/lib/libart.so
lr-x------ 1 root root 64 2021-08-18 08:59 51 ->
/data/app/com.sfexpress.merchant-SqSjnfLvF-MmKY8y4tFh0g==/lib/arm
lr-x------ 1 root root 64 2021-08-18 08:59 52 ->
/apex/com.android.runtime/lib/libart.so
lr-x------ 1 root root 64 2021-08-18 08:59 53 -> /system/lib/liblog.so
lr-x------ 1 root root 64 2021-08-18 08:59 54 -> /system/lib/vndk-sp-
29/libcutils.so
lrwx------ 1 root root 64 2021-08-18 08:59 55 ->
/data/data/com.sfexpress.merchant/.cache/res.data
lrwx------ 1 root root 64 2021-08-18 08:59 56 ->
/data/data/com.sfexpress.merchant/.cache/datarc
lrwx------ 1 root root 64 2021-08-18 08:59 57 -> /dev/ashmem
lr-x------ 1 root root 64 2021-08-18 08:59 576 ->
/data/app/com.xiaomi.aiasst.service-ofy3y3n3ceGgDZlPBk-l9w==/base.apk
lr-x------ 1 root root 64 2021-08-18 08:59 577 -> /data/app/com.achievo.vipshop-
E2RAmZXnPOhGcCfhLsOgKA==/base.apk
lr-x------ 1 root root 64 2021-08-18 08:59 578 -> /system/priv-
app/MusicFX/MusicFX.apk
lrwx------ 1 root root 64 2021-08-18 08:59 58 ->
/data/data/com.sfexpress.merchant/databases/LSStatisticDataBase
lrwx------ 1 root root 64 2021-08-18 08:59 59 -> anon_inode:[eventfd]
lrwx------ 1 root root 64 2021-08-18 08:59 6 -> /dev/null
lrwx------ 1 root root 64 2021-08-18 08:59 60 -> anon_inode:[eventpoll]
lrwx------ 1 root root 64 2021-08-18 08:59 61 -> anon_inode:[eventfd]
lrwx------ 1 root root 64 2021-08-18 08:59 62 -> anon_inode:[eventpoll]
lrwx------ 1 root root 64 2021-08-18 08:59 63 -> anon_inode:[eventfd]
lrwx------ 1 root root 64 2021-08-18 08:59 64 -> anon_inode:[eventpoll]
lrwx------ 1 root root 64 2021-08-18 08:59 65 -> /dev/ashmem
lrwx------ 1 root root 64 2021-08-18 08:59 66 -> anon_inode:[eventfd]
lrwx------ 1 root root 64 2021-08-18 08:59 67 -> anon_inode:[eventpoll]
lr-x------ 1 root root 64 2021-08-18 08:59 68 -> anon_inode:inotify
lrwx------ 1 root root 64 2021-08-18 08:59 69 -> anon_inode:[eventfd]
lrwx------ 1 root root 64 2021-08-18 08:59 7 -> /dev/null
lrwx------ 1 root root 64 2021-08-18 08:59 70 -> anon_inode:[eventpoll]
l-wx------ 1 root root 64 2021-08-18 08:59 71 ->
/data/data/com.sfexpress.merchant/app_bugly/sys_log_1629248368068.txt
```

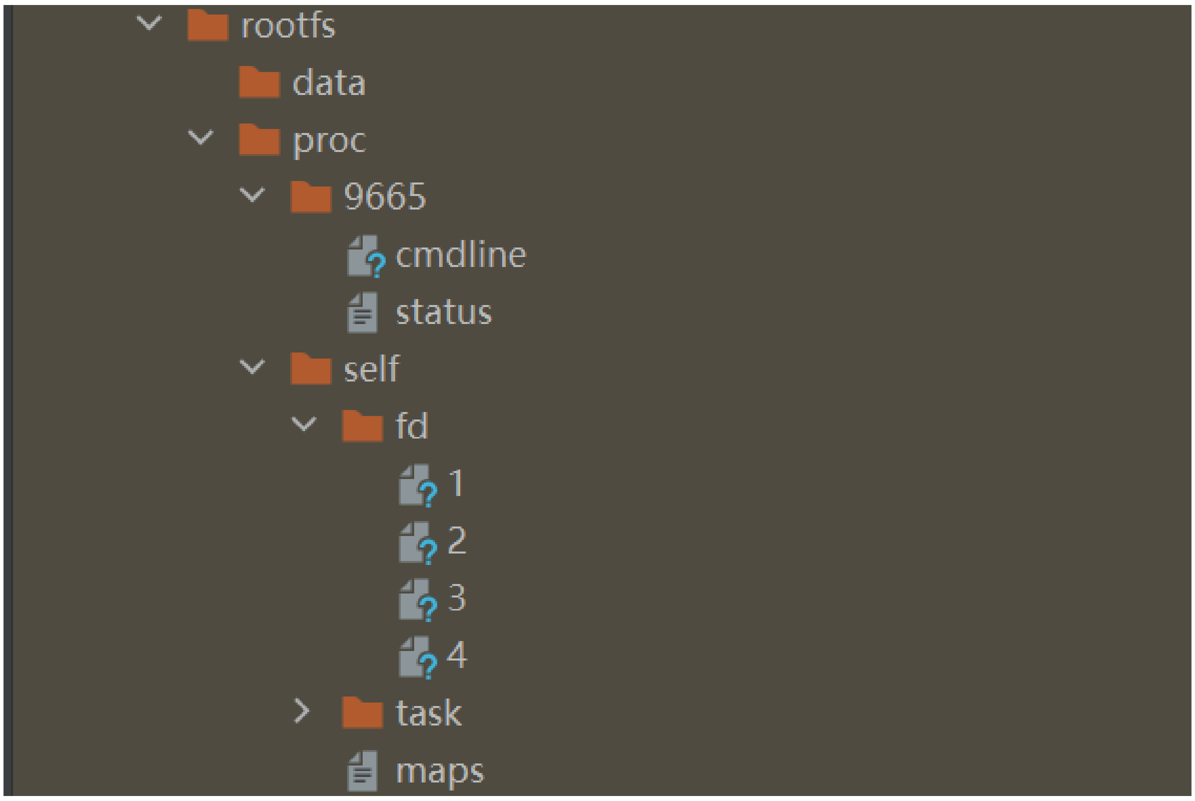
```
lrwx------ 1 root root 64 2021-08-18 08:59 72 ->
/data/data/com.sfexpress.merchant/databases/bugly_db_
lr-x------ 1 root root 64 2021-08-18 08:59 73 -> pipe:[4141436]
l-wx------ 1 root root 64 2021-08-18 08:59 74 ->
/data/data/com.sfexpress.merchant/app_bugly/jni_log_1629248368068.txt
lrwx------ 1 root root 64 2021-08-18 08:59 75 -> anon_inode:[eventfd]
lrwx------ 1 root root 64 2021-08-18 08:59 76 -> anon_inode:[eventpoll]
l-wx------ 1 root root 64 2021-08-18 10:22 77 -> pipe:[4141436]
lrwx------ 1 root root 64 2021-08-18 08:59 78 -> anon_inode:[eventfd]
lrwx------ 1 root root 64 2021-08-18 10:22 79 -> socket:[4139799]
lrwx------ 1 root root 64 2021-08-18 08:59 8 -> /dev/null
lrwx------ 1 root root 64 2021-08-18 10:22 80 -> socket:[4160619]
l-wx------ 1 root root 64 2021-08-18 08:59 81 ->
/storage/emulated/0/Android/data/com.sfexpress.merchant/files/tbslog/tbslog.txt
lrwx------ 1 root root 64 2021-08-18 11:27 82 -> socket:[4160159]
lrwx------ 1 root root 64 2021-08-18 08:59 83 -> anon_inode:[eventpoll]
lr-x------ 1 root root 64 2021-08-18 11:27 84 -> anon_inode:sync_file
lr-x------ 1 root root 64 2021-08-18 11:27 85 -> pipe:[4160621]
lrwx------ 1 root root 64 2021-08-18 08:59 86 -> anon_inode:[eventfd]
lrwx------ 1 root root 64 2021-08-18 08:59 87 -> anon_inode:[eventpoll]
lrwx------ 1 root root 64 2021-08-18 08:59 88 -> /dev/ashmem
lrwx------ 1 root root 64 2021-08-18 08:59 89 -> anon_inode:[eventfd]
lrwx------ 1 root root 64 2021-08-18 08:59 9 -> /dev/null
lrwx------ 1 root root 64 2021-08-18 08:59 90 -> anon_inode:[eventpoll]
l-wx------ 1 root root 64 2021-08-18 11:27 91 -> pipe:[4160621]
lrwx------ 1 root root 64 2021-08-18 08:59 92 -> /dev/ashmem
lrwx------ 1 root root 64 2021-08-18 08:59 93 -> socket:[2675165]
lr-x------ 1 root root 64 2021-08-18 08:59 94 -> pipe:[4160164]
lrwx------ 1 root root 64 2021-08-18 08:59 95 -> /dev/ashmem
lrwx------ 1 root root 64 2021-08-18 08:59 96 -> anon_inode:[eventfd]
lrwx------ 1 root root 64 2021-08-18 08:59 97 -> anon_inode:[eventpoll]
l-wx------ 1 root root 64 2021-08-18 11:27 98 -> pipe:[4160164]
lrwx------ 1 root root 64 2021-08-18 08:59 99 -> anon_inode:[eventfd]
```

Unidbg中当然是有文件描述符和符号链接的概念，但是，怎么把样本真实的fd目录，100% copy到Unidbg环境中呢，我没有找到解决办法。那我们就得更多的了解样本在这里想干啥了。

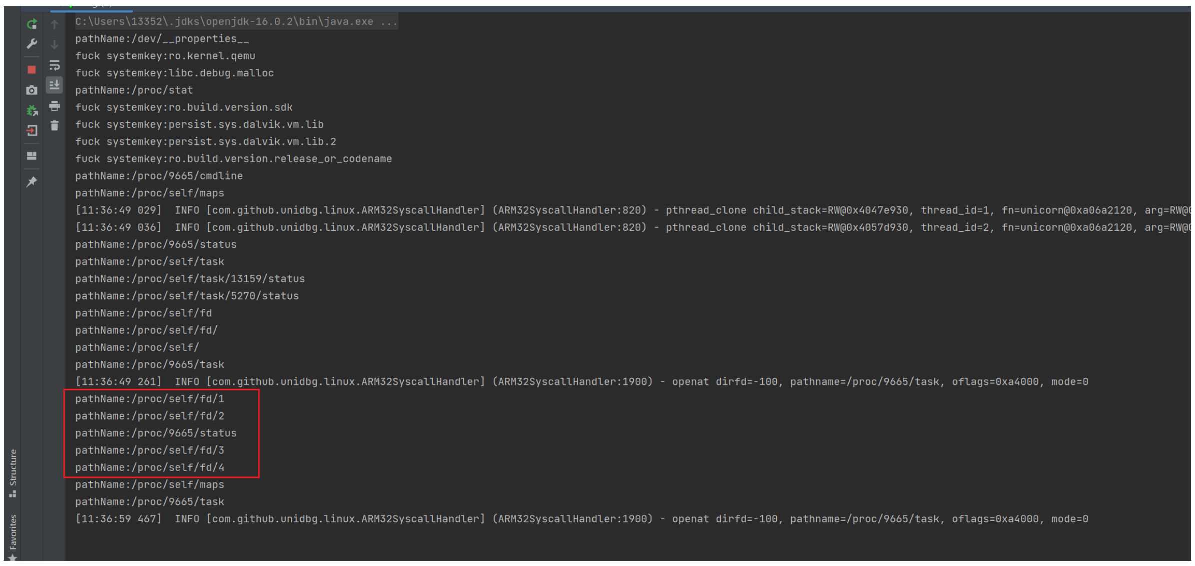
网上冲浪一下

> frida-server与frida-agent通信时，通过管道进行通信。 由于这些管道具有明确的名称，因此可以遍历 */proc//fd* 下的不同文件以找到与 frida 对应的命名管道。

意思就是说查看fd目录下所有文件描述符指向的真正文件呗，看里面有没有frida相关的。那它是不是做了遍历文件描述符这个操作呢？我们验证一下，在fd目录下随便创建几个1234空文件，充当文件描述符。



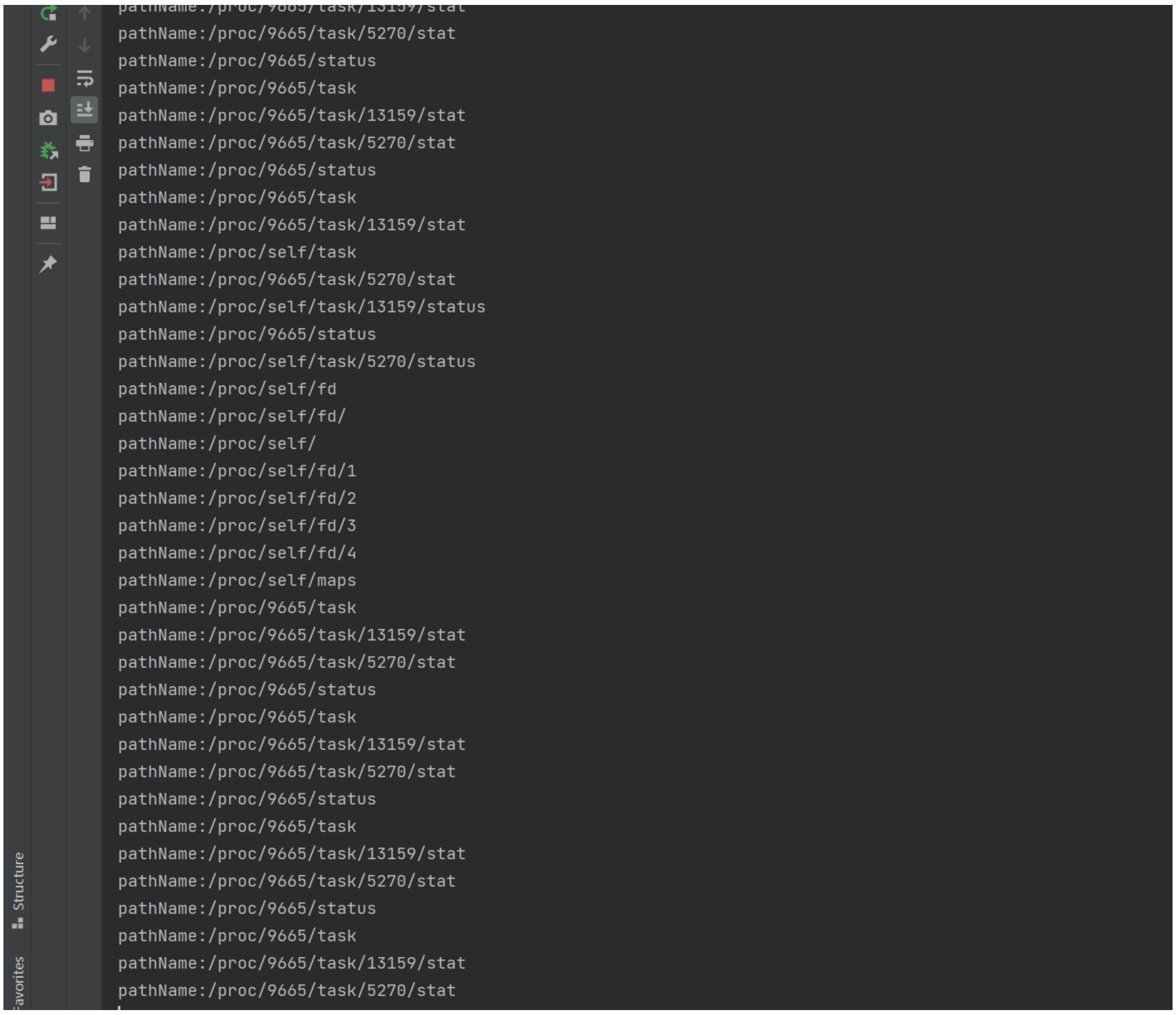
再运行测试一下



```
C:\Users\13352\.jdks\openjdk-16.0.2\bin\java.exe ...
pathName:/dev/__properties__
fuck systemkey:ro.kernel.qemu
fuck systemkey:libc.debug.malloc
pathName:/proc/stat
fuck systemkey:ro.build.version.sdk
fuck systemkey:persist.sys.dalvik.vm.lib
fuck systemkey:persist.sys.dalvik.vm.lib.2
fuck systemkey:ro.build.version.release_or_codename
pathName:/proc/9665/cmdline
pathName:/proc/self/maps
[11:36:49 029]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:820) - pthread_clone child_stack=RW@0x4047e930, thread_id=1, fn=unicorn@0xa06a2120, arg=RW@
[11:36:49 036]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:820) - pthread_clone child_stack=RW@0x4057d930, thread_id=2, fn=unicorn@0xa06a2120, arg=RW@
pathName:/proc/9665/status
pathName:/proc/self/task
pathName:/proc/self/task/13159/status
pathName:/proc/self/task/5270/status
pathName:/proc/self/fd
pathName:/proc/self/fd/
pathName:/proc/self/
pathName:/proc/9665/task
[11:36:49 261]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1900) - openat dirfd=-100, pathname=/proc/9665/task, oflags=0xa4000, mode=0
pathName:/proc/self/fd/1
pathName:/proc/self/fd/2
pathName:/proc/9665/status
pathName:/proc/self/fd/3
pathName:/proc/self/fd/4
pathName:/proc/self/maps
pathName:/proc/9665/task
[11:36:59 467]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1900) - openat dirfd=-100, pathname=/proc/9665/task, oflags=0xa4000, mode=0
```

那我们怎么办呢？除此之外，后续访问了/proc/9665/task，adb pull 对应文件夹到rootfs对应位置。

继续运行，发现已经没有文件缺失了



```
pathName:/proc/9665/task/13159/stat
pathName:/proc/9665/task/5270/stat
pathName:/proc/9665/status
pathName:/proc/9665/task
pathName:/proc/9665/task/13159/stat
pathName:/proc/9665/task/5270/stat
pathName:/proc/9665/status
pathName:/proc/9665/task
pathName:/proc/9665/task/13159/stat
pathName:/proc/self/task
pathName:/proc/9665/task/5270/stat
pathName:/proc/self/task/13159/status
pathName:/proc/9665/status
pathName:/proc/self/task/5270/status
pathName:/proc/self/fd
pathName:/proc/self/fd/
pathName:/proc/self/
pathName:/proc/self/fd/1
pathName:/proc/self/fd/2
pathName:/proc/self/fd/3
pathName:/proc/self/fd/4
pathName:/proc/self/maps
pathName:/proc/9665/task
pathName:/proc/9665/task/13159/stat
pathName:/proc/9665/task/5270/stat
pathName:/proc/9665/status
pathName:/proc/9665/task
pathName:/proc/9665/task/13159/stat
pathName:/proc/9665/task/5270/stat
pathName:/proc/9665/status
pathName:/proc/9665/task
pathName:/proc/9665/task/13159/stat
pathName:/proc/9665/task/5270/stat
pathName:/proc/9665/status
pathName:/proc/9665/task
pathName:/proc/9665/task/13159/stat
pathName:/proc/9665/task/5270/stat
```

我们捋一下，每个文件访问是干啥的

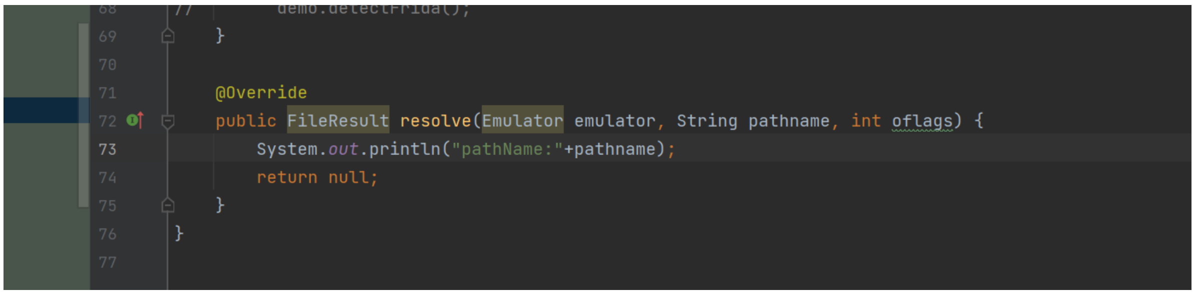
task目录下是线程，当处于调试状态时，在stat文件的第三个字段处会显示小写t，表示Tracing stop.所以对stat文件的访问是反调试。

对status文件的访问同理，TracerPid字段外，第二行的state字段也能够用来判断进程是否处于被调试状态（Tracing stop）。

对maps文件的访问呢?

这里有个小问题要说明一下

我们可以把真机的maps放到rootfs的对应位置，但你会发现程序读取到的并不是你传入的maps，因为Unidbg设计了一个简单的maps，优先级比rootfs补的maps高，但这个maps比较单调，如果样本对maps做复杂一些的解析或者处理，很可能过不了关。



```
//         demo.detectFrida();
    }

    @Override
    public FileResult resolve(Emulator emulator, String pathname, int oflags) {
        System.out.println("pathName:"+pathname);
        return null;
    }
}
```

需要在我们的resolve中用代码方式补maps，才能用上我们从测试机里pull的maps。对maps的读取常常是重要的反Frida战场。

> /proc/pid/maps 包含了当前进程映射的内存区域以及他们的访问权限

比如我将frida脚本注入到应用中，可以在其/proc/pid/maps中找到如下内容

```
7695e08000-769651f000 r-xp 00000000 103:05 1712140
/data/local/tmp/re.frida.server/frida-agent-64.so
7696520000-7696566000 r--p 00717000 103:05 1712140
/data/local/tmp/re.frida.server/frida-agent-64.so
7696566000-769656f000 rw-p 0075d000 103:05 1712140
/data/local/tmp/re.frida.server/frida-agent-64.so
```

当我们使用过检测的frida server时，可以找到如下内容

```
768be42000-768d0c1000 r-xp 00000000 103:05 3031717
/data/local/tmp/dd743070-cd39-441f-b435-0c868b0b9263/74a101a5-edb5-4687-a9f9-
cd086dc46ec0-64.so
768d0c2000-768d15c000 r--p 0127f000 103:05 3031717
/data/local/tmp/dd743070-cd39-441f-b435-0c868b0b9263/74a101a5-edb5-4687-a9f9-
cd086dc46ec0-64.so
768d15c000-768d171000 rw-p 01319000 103:05 3031717
/data/local/tmp/dd743070-cd39-441f-b435-0c868b0b9263/74a101a5-edb5-4687-a9f9-
cd086dc46ec0-64.so
```

可以发现，strongR-frida-android的frida server隐去了明显的frida字符串特征，但是其命名和全路径仍然很不对劲，有较为明显的格式特征，因此建议将server不要放在tmp下，或许可以避免一部分检测。

笔者前些天和朋友分析某样本的frida检测时，发现就是简单的端口检查+maps中匹配如上形式的可疑SO。

但是，通过maps可以获取到的信息，应该远不止此，还有其他角度验证存在frida注入，我就不继续分析了。

我们再看最后一次，访问fd目录。

上面说了一半，但没说完，首先，我们看一下用正常frida server时，fd目录下存在哪些文件？

```
l-wx------ 1 root root 64 2021-08-18 14:50 83 -> /data/local/tmp/re.frida.server/linjector-6
```

而使用strongR-frida-android后，是这样

```
l-wx------ 1 root root 64 2021-08-18 14:47 86 -> /data/local/tmp/dd743070-cd39-441f-b435-
0c868b0b9263/0x7b5b3679204
```

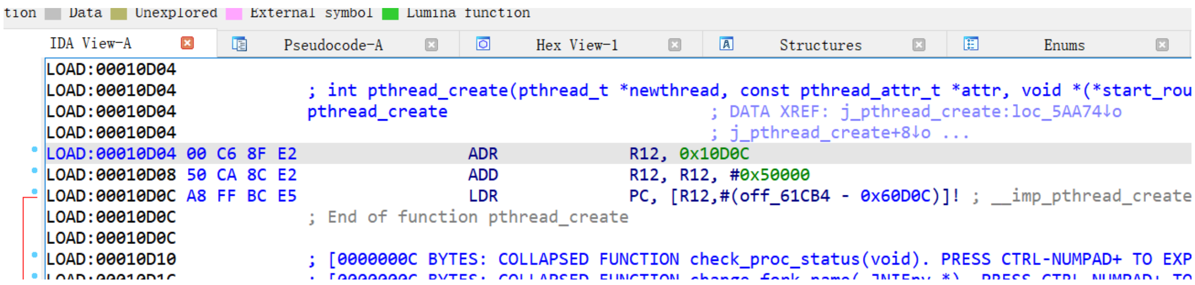
接下来，感兴趣的朋友试着更深入的研究一下，代码逻辑中到底是如何检测Frida 的，欢迎和我讨论交流= =

四、分析与解决方案

咱们这只是一篇小文章，就不花时间去仔细分析了，但app无法使用frida这个问题总得去解决吧？

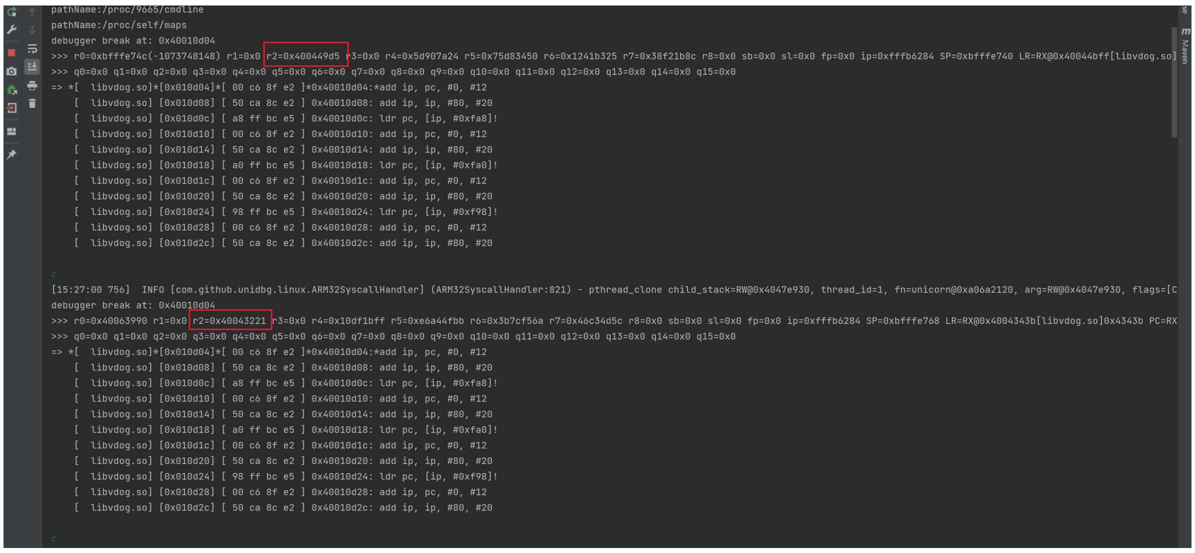
首先，我们要纠正一个错误，上文我们讨论了样本通过哪些文件访问对Frida进行检查，但细心的朋友可能会发现，有些文件访问更像是反调试而非反Frida，更细心一些会发现样本创建了两个子线程而非一个。

哈哈，确实如此



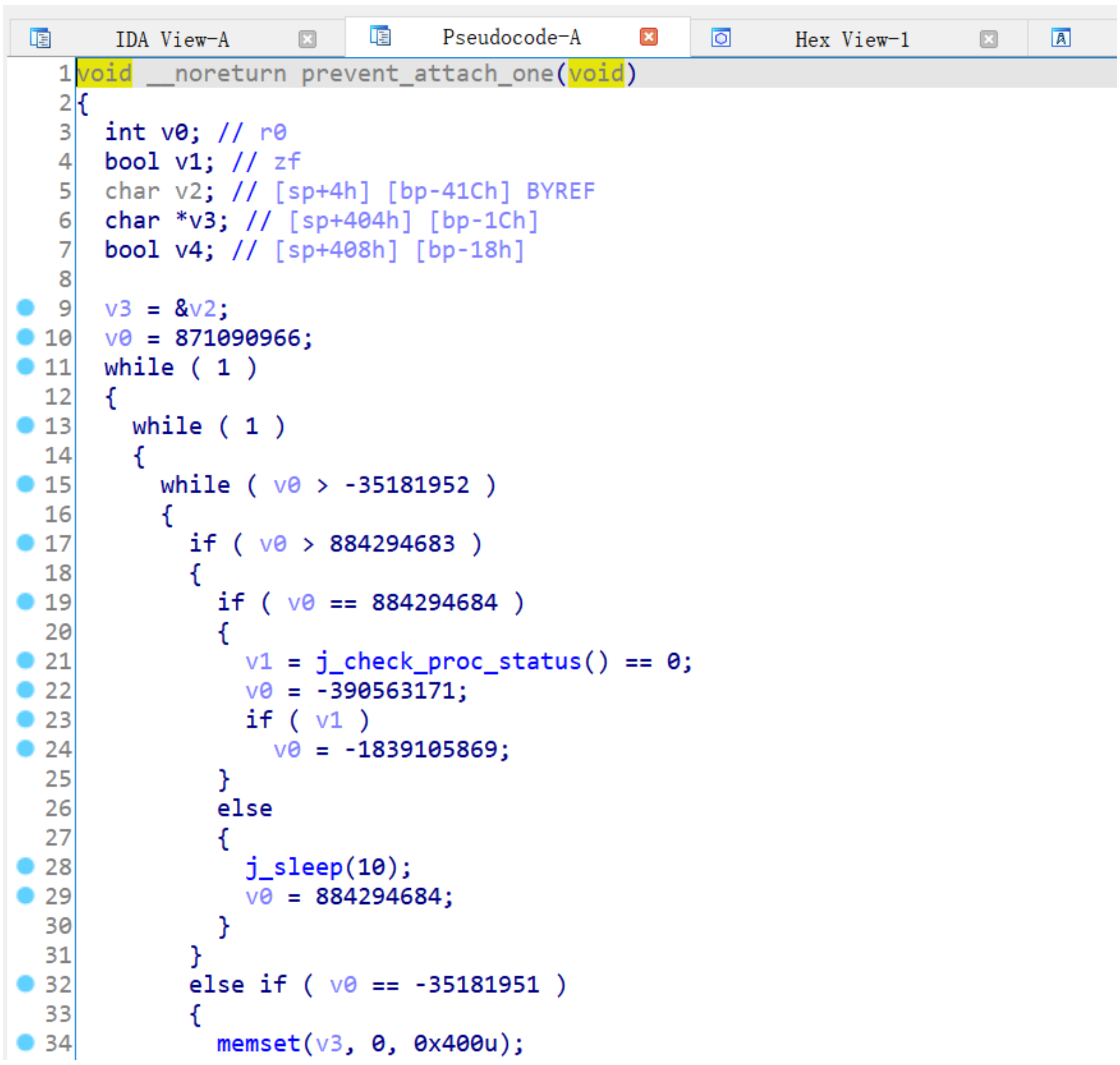
```
LOAD:00010D04
LOAD:00010D04                 ; int pthread_create(pthread_t *newthread, const pthread_attr_t *attr, void *(*start_rou
LOAD:00010D04                 pthread_create                          ; DATA XREF: j_pthread_create:loc_5AA74↓o
LOAD:00010D04                                                         ; j_pthread_create+8↓o ...
LOAD:00010D04 00 C6 8F E2             ADR             R12, 0x10D0C
LOAD:00010D08 50 CA 8C E2             ADD             R12, R12, #0x50000
LOAD:00010D0C A8 FF BC E5             LDR             PC, [R12,#(off_61CB4 - 0x60D0C)]! ; __imp_pthread_create
LOAD:00010D0C                 ; End of function pthread_create
LOAD:00010D0C
LOAD:00010D10                 ; [0000000C BYTES: COLLAPSED FUNCTION check_proc_status(void). PRESS CTRL-NUMPAD+ TO EXP
```

在目标SO的pthread_create处下断，你会发现，一共调用了两次，看它的参数3，即寄存器r2你会发现



```
pathName:/proc/9665/cmdline
pathName:/proc/self/maps
debugger break at: 0x40010d04
>>> r0=0xbfffe74c(-1073748148) r1=0x0 r2=0x400449d5 r3=0x0 r4=0x5d907a24 r5=0x75d83450 r6=0x1241b325 r7=0x38f21b8c r8=0x0 sb=0x0 sl=0x0 fp=0x0 ip=0xfffb6284 SP=0xbfffe740 LR=RX@0x40044bff[libvdog.so]
>>> q0=0x0 q1=0x0 q2=0x0 q3=0x0 q4=0x0 q5=0x0 q6=0x0 q7=0x0 q8=0x0 q9=0x0 q10=0x0 q11=0x0 q12=0x0 q13=0x0 q14=0x0 q15=0x0
=> *[  libvdog.so]*[0x010d04]*[ 00 c6 8f e2 ]*0x40010d04:*add ip, pc, #0, #12
    [  libvdog.so] [0x010d08] [ 50 ca 8c e2 ] 0x40010d08: add ip, ip, #80, #20
    [  libvdog.so] [0x010d0c] [ a8 ff bc e5 ] 0x40010d0c: ldr pc, [ip, #0xfa8]!
    [  libvdog.so] [0x010d10] [ 00 c6 8f e2 ] 0x40010d10: add ip, pc, #0, #12
    [  libvdog.so] [0x010d14] [ 50 ca 8c e2 ] 0x40010d14: add ip, ip, #80, #20
    [  libvdog.so] [0x010d18] [ a0 ff bc e5 ] 0x40010d18: ldr pc, [ip, #0xfa0]!
    [  libvdog.so] [0x010d1c] [ 00 c6 8f e2 ] 0x40010d1c: add ip, pc, #0, #12
    [  libvdog.so] [0x010d20] [ 50 ca 8c e2 ] 0x40010d20: add ip, ip, #80, #20
    [  libvdog.so] [0x010d24] [ 98 ff bc e5 ] 0x40010d24: ldr pc, [ip, #0xf98]!
    [  libvdog.so] [0x010d28] [ 00 c6 8f e2 ] 0x40010d28: add ip, pc, #0, #12
    [  libvdog.so] [0x010d2c] [ 50 ca 8c e2 ] 0x40010d2c: add ip, ip, #80, #20


c
[15:27:00 756]  INFO [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:821) - pthread_clone child_stack=RW@0x4047e930, thread_id=1, fn=unicorn@0xa06a2120, arg=RW@0x4047e930, flags=[C
debugger break at: 0x40010d04
>>> r0=0x40063990 r1=0x0 r2=0x40043221 r3=0x0 r4=0x10df1bff r5=0xe6a44fbb r6=0x3b7cf56a r7=0x46c34d5c r8=0x0 sb=0x0 sl=0x0 fp=0x0 ip=0xfffb6284 SP=0xbfffe768 LR=RX@0x4004343b[libvdog.so]0x4343b PC=RX
>>> q0=0x0 q1=0x0 q2=0x0 q3=0x0 q4=0x0 q5=0x0 q6=0x0 q7=0x0 q8=0x0 q9=0x0 q10=0x0 q11=0x0 q12=0x0 q13=0x0 q14=0x0 q15=0x0
=> *[  libvdog.so]*[0x010d04]*[ 00 c6 8f e2 ]*0x40010d04:*add ip, pc, #0, #12
    [  libvdog.so] [0x010d08] [ 50 ca 8c e2 ] 0x40010d08: add ip, ip, #80, #20
    [  libvdog.so] [0x010d0c] [ a8 ff bc e5 ] 0x40010d0c: ldr pc, [ip, #0xfa8]!
    [  libvdog.so] [0x010d10] [ 00 c6 8f e2 ] 0x40010d10: add ip, pc, #0, #12
    [  libvdog.so] [0x010d14] [ 50 ca 8c e2 ] 0x40010d14: add ip, ip, #80, #20
    [  libvdog.so] [0x010d18] [ a0 ff bc e5 ] 0x40010d18: ldr pc, [ip, #0xfa0]!
    [  libvdog.so] [0x010d1c] [ 00 c6 8f e2 ] 0x40010d1c: add ip, pc, #0, #12
    [  libvdog.so] [0x010d20] [ 50 ca 8c e2 ] 0x40010d20: add ip, ip, #80, #20
    [  libvdog.so] [0x010d24] [ 98 ff bc e5 ] 0x40010d24: ldr pc, [ip, #0xf98]!
    [  libvdog.so] [0x010d28] [ 00 c6 8f e2 ] 0x40010d28: add ip, pc, #0, #12
    [  libvdog.so] [0x010d2c] [ 50 ca 8c e2 ] 0x40010d2c: add ip, ip, #80, #20


c
```

一个线程用于反frida，另一个呢，则是反调试



```
void __noreturn prevent_attach_one(void)
{
  int v0; // r0
  bool v1; // zf
  char v2; // [sp+4h] [bp-41Ch] BYREF
  char *v3; // [sp+404h] [bp-1Ch]
  bool v4; // [sp+408h] [bp-18h]

  v3 = &v2;
  v0 = 871090966;
  while ( 1 )
  {
    while ( 1 )
    {
      while ( v0 > -35181952 )
      {
        if ( v0 > 884294683 )
        {
          if ( v0 == 884294684 )
          {
            v1 = j_check_proc_status() == 0;
            v0 = -390563171;
            if ( v1 )
              v0 = -1839105869;
          }
          else
          {
            j_sleep(10);
            v0 = 884294684;
          }
        }
        else if ( v0 == -35181951 )
        {
          memset(v3, 0, 0x400u);
```

具体内部逻辑，感兴趣的可以分析一下，我们直接用frida patch掉这两个检测线程的创建，看管不管用

```
var p_pthread_create = Module.findExportByName("libc.so", "pthread_create");
var pthread_create = new NativeFunction( p_pthread_create, "int", ["pointer",
"pointer", "pointer", "pointer"]);
Interceptor.replace( p_pthread_create, new NativeCallback(function (ptr0, ptr1,
ptr2, ptr3) {
    console.log("pthread_create() overloaded");
    var ret = ptr(0);
    var base_addr;
    try {
        base_addr = Module.getBaseAddress('libvdog.so');
    } catch(error) {
    }

    var funcAddress = ptr(0);
    if(base_addr != null){
        funcAddress = ptr2.sub(base_addr);
    }
    if(funcAddress.equals(0x449d5) || funcAddress.equals(0x43221)){
        return -1;
    } else {
        ret = pthread_create(ptr0,ptr1,ptr2,ptr3);
        return ret;
    }
}, "int", ["pointer", "pointer", "pointer", "pointer"]));
```

恰好管用，它没有更复杂的逻辑，好耶 = =

下篇再见。

样本链接：https://pan.baidu.com/s/1GcfbVs-cmWWb5LyAKtwxCg
提取码：017s