# 签名绕过

发现这个签名绕过，是在kingroot原版root后卸载可用，而重启后不可用。

## java绕过

搜索Landroid/content/pm/PackageInfo;->signatures:[Landroid/content/pm/Signature;定位。

定位到的方法为：

Lcom/kingroot/kinguser/ace;->dx(Ljava/lang/String;)Landroid/content/pm/Signature; = is call

Lcom/kingroot/kinguser/ace;->dv(Ljava/lang/String;)Landroid/content/pm/Signature; = is called

进一步查找：

Lcom/kingroot/kinguser/ace;->dx(Ljava/lang/String;)Landroid/content/pm/Signature; = is call----------------------------为获取手机已安装程序的

Lcom/kingroot/kinguser/ace;->du(Ljava/lang/String;)Ljava/lang/String; = is called

param = /data/app/com.kingroot.kinguser-1/base.apk-----------为本apk的签名信息获取

Lcom/kingroot/kinguser/ace;->dv(Ljava/lang/String;)Landroid/content/pm/Signature; = is called

Lcom/kingroot/kinguser/ace;->du(Ljava/lang/String;)Ljava/lang/String; : res = E89B158E4BCF988EBD09EB83F5378E87

因此，只需要将程序的返回值永远是真是的apk签名“191240FCB048127DB9110D1B30537FDE”信息就好了。

### other

在上面两个都搞定了以后，发现在kingroot(原始)root完毕，再安装我的包，还是出现下面的：



界面为：dialog\_repackage\_warning.xml

activity为：Lcom/kingroot/kinguser/activitys/RePackageWarningActivity;

Lcom/kingroot/kinguser/oq;->a(Lcom/kingroot/kinguser/xp;)V

->Lcom/kingroot/kinguser/atg;->w(Z)V======== Lcom/kingroot/kinguser/or;->w(Z)V

->Lcom/kingroot/kinguser/activitys/RePackageWarningActivity;

Lcom/kingroot/kinguser/oq;->a(Lcom/kingroot/kinguser/xp;)V

->Lcom/kingroot/kinguser/op;->c(Lcom/kingroot/kinguser/or;)Z

->Lcom/kingroot/kinguser/op;->b(Lcom/kingroot/kinguser/or;)Z

-> Lcom/kingroot/kinguser/op;->fX()Ljava/lang/String;

-> Lcom/kingroot/kinguser/ace;--------3种方式验证签名

其中一种能够方式ace.bk

Lcom/kingroot/kinguser/acu;->a(Lcom/kingroot/kinguser/acx;J)Lcom/kingroot/kinguser/acw;

-->Lcom/kingroot/kinguser/acw;-><init>(Ljava/lang/String;Ljava/lang/Integer;Ljava/lang/String;Ljava/lang/String;)V

-->iput-object p3, p0, Lcom/kingroot/kinguser/acw;->Hi:Ljava/lang/String;

在acu.a中

调用[Hh:export CLASSPATH=/data/data/com.kingroot.kinguser/applib/tm.dex && /system/bin/app\_process /system/bin com.kingroot.kinguser.TellMe 10124,Hl:export CLASSPATH=/data/data/com.kingroot.kinguser/applib/tm.dex && /system/bin/app\_process /system/bin com.kingroot.kinguser.TellMe 10124,mTimeout:120000,]

获取程序的签名md5

将命令的执行结果替换掉签名“191240FCB048127DB9110D1B30537FDE”就可以了。

## so绕过

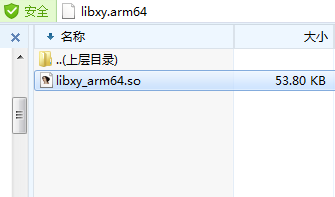
native 方法

load/loadLibrary

Lcom/kingroot/kinguser/zu;->F(Ljava/lang/String;Ljava/lang/String;)Z:load = /data/data/com.kingroot.kinguser/applib/libpu.so------------------------------------初步确定为反调试的so

Lcom/kingroot/kinguser/zu;->F(Ljava/lang/String;Ljava/lang/String;)Z:load = /data/data/com.kingroot.kinguser/applib/libhxy64.so -------------------------------初步确定为签名验证的so

进一步分析上面libhxy64.so,发现是libxy.arm64解压(为gzip格式)后得到的。



通过ida逆向，



因此将so修改后，重新打包为gz格式的压缩包，放到assets目录下。

## plugin java检查签名绕过

在Lcom/kingroot/kinguser/bqs;->ff(I)I中，加载dex文件：

DexClassLoaderHelper = init------------start--------

dexPath = /data/data/com.kingroot.kinguser/app\_workspace/app/com.kingroot.RushRoot-6122de20a94ae6d97defb274e7cccc84.apk

optimizedDirectory = /data/data/com.kingroot.kinguser/app\_workspace/dalvik-cache

libraryPath = /data/data/com.kingroot.kinguser/app\_workspace/data/com.kingroot.RushRoot/lib/armeabi

DexClassLoaderHelper = init------------end--------

因此，替换掉此Dex文件，并加入日志，发现在：Lcom/kingroot/RushRoot/ew;->b中，进行签名校验：

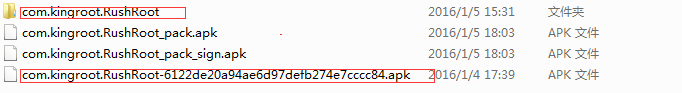
Lcom/kingroot/RushRoot/ew;->b : getPackageName = com.kingroot.kinguser

Lcom/kingroot/RushRoot/ew;->b : signature = E89B158E4BCF988EBD09EB83F5378E87

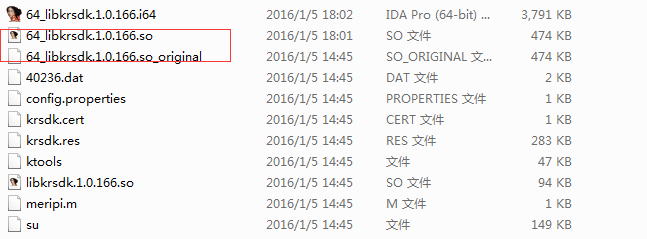
让其返回常量签名“191240FCB048127DB9110D1B30537FDE”

## plugin so检查签名绕过

com.kingroot.kinguser\app\_workspace\app，下有apk文件，反汇编为com.kingroot.RushRoot



在asserts目录下：





# 执行

对RushRoot.apk的所有的ProcessBuilder和Runtime进行监控，发现在Lcom/kingroot/sdk/root/d;->a的ProcessBuilder中执行krmain。

在Lcom/kingroot/sdk/root/d;->a的ProcessBuilder执行前，对data目录进行copy，获取运行时的整个环境。

/data/data/com.kingroot.kinguser/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/krmain -k /data/data/com.kingroot.kinguser/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/krcfg.txt

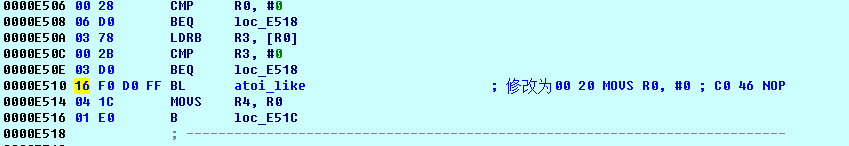
ProcessBuilder directory = /data/data/com.kingroot.kinguser/app\_workspace/data/com.kingroot.RushRoot/krsdk/play

ProcessBuilder redirectErrorStream = true

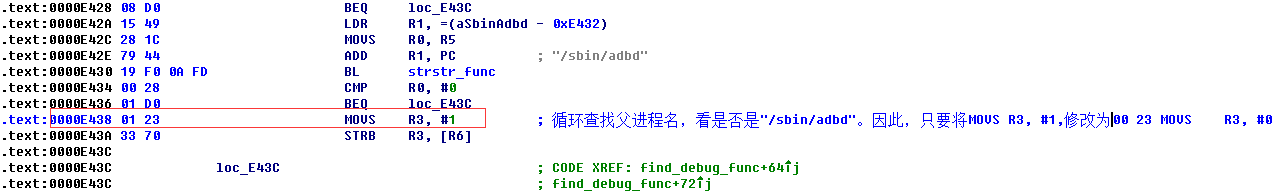
## krmain反调试

### /proc/self/status

通过查找"TracerPid:"，定位反调试函数，发现此函数返回/proc/self/status中的TracerPid的值，我们让其永远返回0就可以了。



### 循环查找父进程名，是否是/sbin/adbd

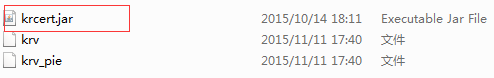


## krmain签名校验

过了krmain反调试之后，使用strace 跟踪kramin。搜索krmain的execve/open/read/write函数。发现文件最后释放krv.tgz、minitar两个文件。

然后用pull.py程序pull出这两个文件（运行时实时pull）。

其中krv.tgz为压缩文件，解压后里面有个jar文件，jar文件验证程序签名。



通过对比krmain中的二进制和krv.tgz/minitor这两个文件，发现他们是从krmain中释放出来的（krmain中的数据并没有加密，只是将这2个文件的二进制赋给了一个变量）。

利用ps.py获取运行进程的cmdline:

/data/data/com.kingroot.kinguser/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/krv\_pie /data/data/com.kingroot.kinguser/app\_workspace/data/com.kingroot.RushRoot/krsdk/play /data/data/com.kingroot.kinguser/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/krcert.jar

调用过程：

u0\_a159 5475 472 2240092 172340 ffffffff 00000000 S com.kingroot.kinguser

u0\_a159 7840 5475 3072 176 ffffffff 00000000 S krs\_100170

u0\_a159 7933 7840 3072 204 ffffffff 00000000 S krs\_100170

u0\_a159 7947 7933 31648 8420 ffffffff 00000000 S /data/data/com.kingroot.kinguser/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/krv\_pie

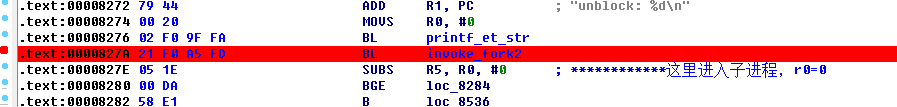
因此krmain先fork，然后子进程再fork，调用krv\_pie进行签名验证。

通过逆向krcert.jar，发现传递给它3个参数：apk路径，待验证包名，待验证md5。验证过程为从apk路径中拿到assets/ krsdk.cert文件，从里面解析出内置包名/内置md5，与待验证包名/待验证md5比较返回结构。

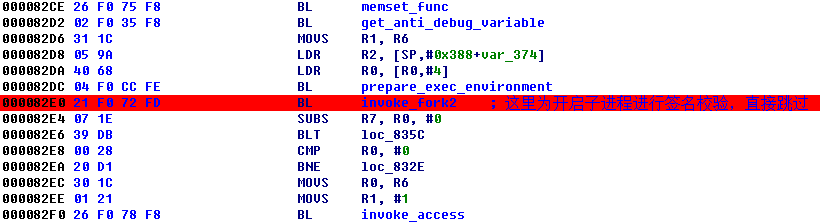
绕过这个签名验证，方法思路是不让krmain执行krv\_pie签名验证，并修改签名验证结果。

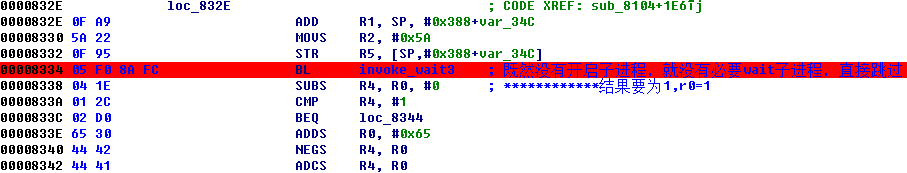
~~在/proc/kallsyms中查找hdd\_init\_tx\_rx字符串。~~

通过字符串定位到[et] unblock: 0，然后直接进入子进程：

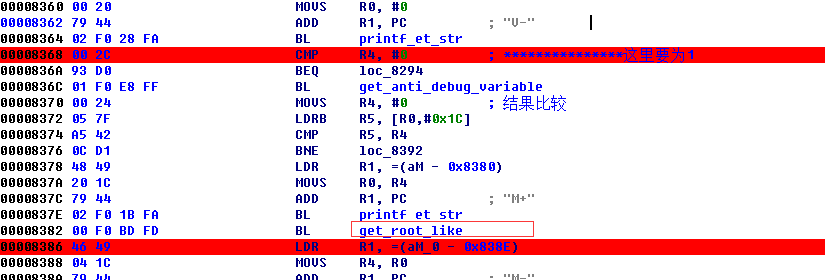


不进行签名校验：





修改签名验证结果，并进入root函数：



进入root函数，发现先释放文件，然后fork调用释放的文件

sh -c cd /data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play;

/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/winkle64

2

/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk

/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play

/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/krsdk.cert

## 调用方案

u0\_a159 5475 472 2240092 172340 ffffffff 00000000 S com.kingroot.kinguser

u0\_a159 7840 5475 3072 176 ffffffff 00000000 S krs\_100170

u0\_a159 7933 7840 3072 204 ffffffff 00000000 S krs\_100170

u0\_a159 8054 7933 5116 1288 ffffffff 00000000 S sh

root 8055 8054 4812 1400 ffffffff 00000000 R /data/data/com.kingroot.kinguser/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/winkle64

root 8277 8055 5116 1292 ffffffff 00000000 S sh

root 8278 8277 5116 1496 ffffffff 00000000 S sh

root 8337 8278 976 836 ffffffff 00000000 D /data/data/com.kingroot.kinguser/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/.68f7666/supolicy

其中：

7840=/data/data/com.kingroot.kinguser/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/krmain -k /data/data/com.kingroot.kinguser/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/krcfg.txt

因此krmain先fork，然后子进程再fork，调用sh /data/data/com.kingroot.kinguser/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/winkle64

## winkle64执行

cd /data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play

;

/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/winkle64

2

/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk

/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play

/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/krsdk.cert

### 调试命令：

cd /data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play

/data/local/tmp/gdbserver localhost:23946 /data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/winkle64 2 /data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk /data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play /data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/krsdk.cert

### 打印信息

[ZTE|ZTE A2015|Linux version 3.10.49-perf-ga5b53b6 (zte@scl\_xa243\_222) (gcc version 4.9.x-google 20140827 (prerelease) (GCC) ) #2 SMP PREEMPT Thu Sep 24 12:45:18 CST 2015]

0c540ac35bc597fa3611ef9d82ea6591912fab4e14caf70302ef37a2111810fb

KRSLOG: /data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play

TOY

1 4

rop 1: No such file or directory

0xffffffc01b804000

1 4

rop 2: Operation not permitted

uname

linux\_ver

##########

off\_set

KRSLOG: prepare done.

##########

##########

##########

##########

failed to open kptr\_restrict

##########

##########

##########

##########

##########

##########

### 调用过程：

**在下面的函数下断点：**

mkdir : b \*0x426A28

open : b \*0x41E2D0

unlink : b \*0x4267E8

socket : b \*0x42BC50

connect : b \*0x42BBD8

getuid : b \*0x4268C0

invoke\_clone: b \*0x426444

clone : b \*0x4264A0

ioctl : b \*0x42BBC0

fcntl : b \*0x426878

prctl : b \*0x4269B0

**调用过程为：**

socket:

int sockfd = socket(2, 1, 0)

open:

/system/build.prop

open:

/proc/version

open:

/system/build.prop

open:

/system/build.prop

open:

/system/build.prop

open:

/proc/version

open:

flintlock

mkdir:

/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/.4a4a107

unlink:

/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/.4a4a107/busybox

open:生成文件：

/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/.6962bc5/busybox

unlink:

/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/.4a4a107/toy

open:生成文件：

/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/.6962bc5/toy

fork进程后，用busybox解压文件：

命令：

/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/.4a4a107/busybox tar xzf /data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/.4a4a107/toy -C / data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/.4a4a107

ioctl:

ioctl(sockfd, 0x8bf7, "wlan0");

ioctl:

ioctl(sockfd, 0x8bf7, "wlan0");

修改进程名：

prctl(15, "n14PwfvAq8")

open:

/proc/iomem

open:

/proc/sys/kernel/kptr\_restrict

open:

/proc/sys/kernel/kptr\_restrict

open:root成功了

/proc/kallsyms

open:root成功了

/proc/kallsyms

open：

/proc/sys/kernel/kptr\_restrict

open:

/proc/sys/kernel/kptr\_restrict

open:

/proc/kallsyms

fork:

sh /data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/.697432e/postroot.sh /data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/.697432e 2 /data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk /data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play /data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/krsdk.cert >/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/postroot\_log.txt 2>&1

sh

/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/.697432e/postroot.sh

/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/.697432e

2

/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk

/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play

/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/krsdk.cert

>/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/postroot\_log.txt

2>&1

fork:

chmod 0666 /data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/postroot\_log.tx

#### 进一步分析

对vfork\_and\_execute\_something执行的地方下断点：

1). b \*0x40B650

2). b \*0x40BCC4

3). b \*0x40BD78

1).处执行命令

/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/.4a4a107/busybox tar xzf /data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/.4a4a107/toy -C / data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/.4a4a107

2).处执行命令

sh /data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/.697432e/postroot.sh /data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play/.697432e 2 /data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk /data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/play /data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/krsdk.cert >/data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/postroot\_log.txt 2>&1

3).处执行命令

chmod 0666 /data/local/tmp/vsnake\_dir/app\_workspace/data/com.kingroot.RushRoot/krsdk/postroot\_log.txt

chmod 0666 /data/local/tmp/winkle\_dir/postroot\_log.txt

### 当winkle64在干净的手机上不能root成功的原因

2次检查检查wlan0设备是否开着，只有开着的时候才执行方案。

ioctl(sockfd, 0x8BF7, "wlan0");

## 让root方案满足我们需求

在上面的调用过程分析中，我们看到了3). b \*0x40BD78执行的是我们root后需要做的事情，因此我们仅仅需要修改这个就ok了。



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 未加密数据 |  |  | 修改后未加密数据 |  |
| 0x63->c |  |  | 0x73->s |  |
| W10(解密数） | W11元数据 | 结果 | 修改后元数据 | 修改后结果 |
| 0xca | 0xa2 | 0x68->h | 0xa2 | 0x68->h |
| 0xcb | 0xa6 | 0x6d->m | 0xeb | 0x20-> |
| 0xcc | 0xa3 | 0x6f->o | 0xe9 | 0x25->% |
| 0xcd | 0xa9 | 0x64->d | 0xbe | 0x73->s |
| 0xce | 0xee | 0x20-> | 0xe1 | 0x2f->/ |
| 0xcf | 0xff | 0x30->0 | 0xbf | 0x70->p |
| 0xd0 | 0xe6 | 0x36->6 | 0xbf | 0x6f->o |
| 0xd1 | 0xe7 | 0x36->6 | 0xa2 | 0x73->s |
| 0xd2 | 0xe4 | 0x36->6 | 0xa6 | 0x74->t |
| 0xd3 | 0xf3 | 0x20-> | 0xd3 | 0x0->'\0' |
| 0xd4 | 0xf1 | 0x25->% |  |  |
| 0xd5 | 0xa6 | 0x73->s |  |  |
| 0xd6 | 0xf9 | 0x2f->/ |  |  |
| 0xd7 | 0xa7 | 0x70->p |  |  |
| 0xd8 | 0xb7 | 0x6f->o |  |  |
| 0xd9 | 0xaa | 0x73->s |  |  |
| 0xda | 0xae | 0x74->t |  |  |
| 0xdb | 0xa9 | 0x72->r |  |  |
| 0xdc | 0xb3 | 0x6f->o |  |  |
| 0xdd | 0xb2 | 0x6f->o |  |  |
| 0xde | 0xaa | 0x74->t |  |  |
| 0xdf | 0x80 | 0x5f->\_ |  |  |
| 0xe0 | 0x8c | 0x6c->l |  |  |
| 0xe1 | 0x8e | 0x6f->o |  |  |
| 0xe2 | 0x85 | 0x67->g |  |  |
| 0xe3 | 0xcd | 0x2e->. |  |  |
| 0xe4 | 0x90 | 0x74->t |  |  |
| 0xe5 | 0x9d | 0x78->x |  |  |
| 0xe6 | 0x92 | 0x74->t |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 地址 | 原指令 | 原二进制 | 修改后指令 | 修改后二进制 |
| 40BCD0 | MOV X10, #0xFFEE000000000000 | CA FD FF D2 | MOV X10, #0xbfe1000000000000 | 2A FC F7 D2 |
| 40BCD4 | MOVK X10, #0xA9A3,LSL#32 | 6A 34 D5 F2 | MOVK X10, #0xbee9,LSL#32 | 2A DD D7 F2 |
| 40BCD8 | MOVK X10, #0xA6A2,LSL#16 | 4A D4 B4 F2 | MOVK X10, #0xeba2,LSL#16 | 4A 74 BD F2 |
| 40BCDC | MOVK X10, #0x63C9 | 2A 79 8C F2 | MOVK X10, #0x73C9 | 2A 79 8E F2 |
|  |  |  |  |  |
| 40BCF4 | MOVK X10, #0xF3E4,LSL#16 | 8A 7C BE F2 | MOVK X10, #0xd3a6,LSL#16 | CA 74 BA F2 |
| 40BCF8 | MOVK X10, #0xE7E6 | CA FC 9C F2 | MOVK X10, #0xa2bf | EA 57 94 F2 |
|  |  |  |  |  |

c嵌入汇编，然后ndk编译获取机器码值

# 方案原理：

预先一系列操作。

validate\_0\_func：

read：将内核地址内容写入pipe，然后从pipe读出。

int filedes[0];

pipe(filedes);

write(filedes[1], kernelAddr, len);

read(filedes[0], processLocalAddr, len);

validate\_1\_func：

write:将本地数据写入管道，然后将数据读出到内核地址

int filedes[0];

pipe(filedes);

write(filedes[1], processLocalAddr, len);

read(filedes[0], kernelAddr, len);