# 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

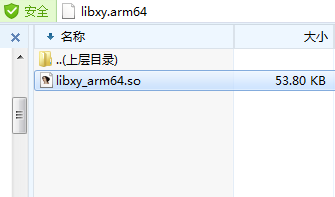
native 方法

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：

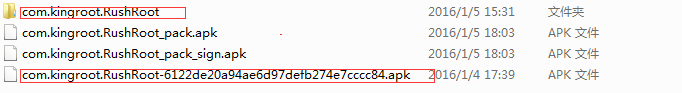
Lcom/kingroot/RushRoot/ew;->b : getPackageName = com.kingroot.kinguser

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

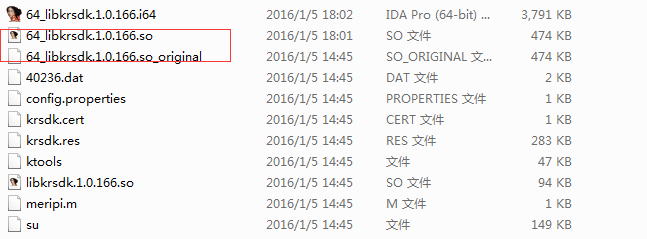
让其返回常量签名“191240FCB048127DB9110D1B30537FDE”

## 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

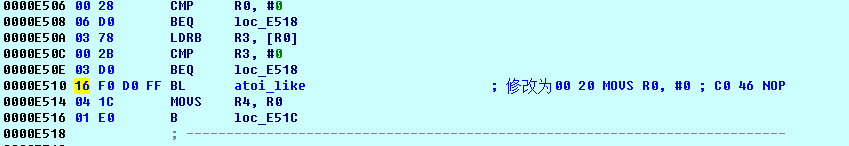
ProcessBuilder directory = /data/data/com.kingroot.kinguser/app\_workspace/data/com.kingroot.RushRoot/krsdk/play

ProcessBuilder redirectErrorStream = true

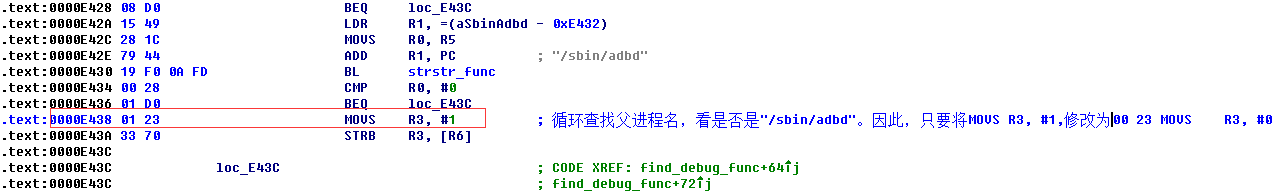
## krmain反调试

### /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

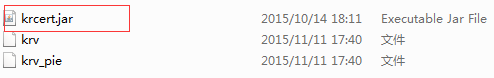


## krmain签名校验

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

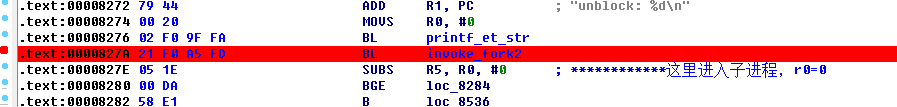
因此krmain先fork，然后子进程再fork，调用krv\_pie进行签名验证。

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。

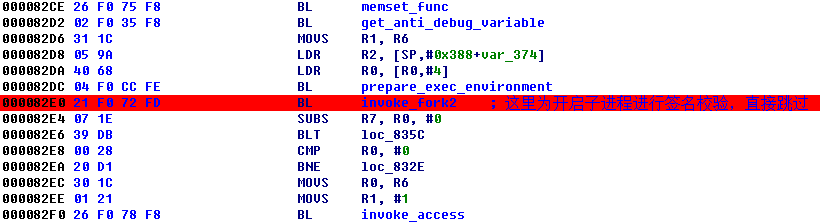
绕过这个签名验证，方法思路是不让krmain执行krv\_pie签名验证，并修改签名验证结果。

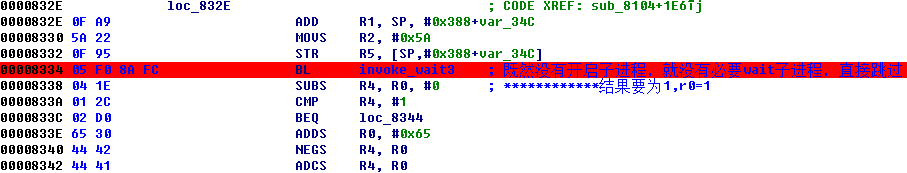
~~在/proc/kallsyms中查找hdd\_init\_tx\_rx字符串。~~

通过字符串定位到[et] unblock: 0，然后直接进入子进程：

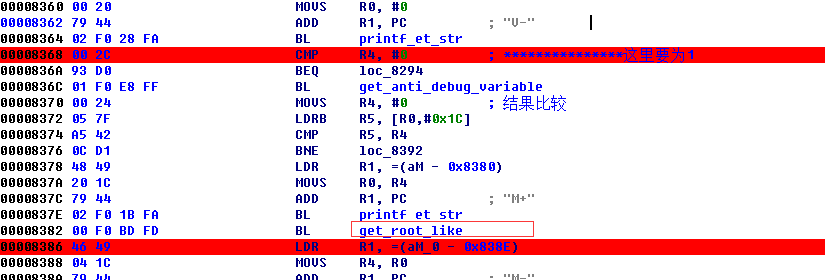


不进行签名校验：





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

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