




Android逆向之旅---应用的"反调试"方案解析(附加修改IDA调试端口和修改内核信息)

原创

尼古拉斯.赵四  于 2020-04-19 12:14:07 发布  964  收藏 4

文章标签: [android](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: https://blog.csdn.net/m0_46204016/article/details/105607778

版权

一、前言

在前一篇文章中详细介绍了 [Android现阶段可以采用的几种反调试方案策略](#), 我们在破解逆向应用的时候, 一般现在第一步都回去解决反调试, 不然后续步骤无法进行, 当然如果你是静态分析的话获取就没必要了。但是有时候必须要借助动态调试方可破解, 就需要进行操作了。现阶段反调试策略主要包括以下几种方式:

第一、自己附加进程, 先占坑, `ptrace(PTRACE_TRACEME, 0, 0, 0)`!

第二、签名校验不可或缺的一个选择, 本地校验和服务端校验双管齐下!

第三、借助系统api判断应用调试状态和调试属性, 最基础的防护!

第四、轮训检查android_server调试端口信息和进程信息, 防护IDA的一种有效方式!

第五、轮训检查自身status中的TracerPid字段值, 防止被其他进程附加调试的一种有效方式!

所以本文就来一一讲解如何解决这几种方式的反调试方案。

二、方法总结

第一种: 找到关键方法, 注释代码

这种方式需要采用静态方式分析代码, 找到关键方法进行反调试代码功能注释, 这种方式可以应对与上面所有的反调试方案, 但是对于轮训检查这种方式就不太适合了, 为什么? 操作过的同学会发现, 在去除反调试功能的时候那种痛苦了。所以这种注释代码, 个人觉得只适用于以下几种反调试:

第一、自己附加进程

这个可以IDA打开关键so代码, 找到这段代码处: `ptrace(PTRACE_TRACEME, 0, 0, 0)`, 直接nop掉即可。这个没什么难度, 因为就一行代码, 说白了就几条arm指令罢了。IDA静态分析so也是无压力的。

第二、签名校验

最后总结了一个比较简单的过滤签名校验的方法: 先在Jadx中打开应用之后, 全局搜字符串内容: "**signatures**", 这个就可以定位到获取应用签名信息的地方了, 然后可以依次跟踪校验的地方了。找到具体地方代码直接注释即可。但是如果服务端交互信息中携带了签名校验, 而签名校验又在so中, 那么就需要另外操作了, 这部分知识点将在后面单独一篇文章详细介绍如何破解。

第三、调用系统api判断当前应用是否处于调试状态

这种方式看到我们实现的逻辑还是比较简单的，直接调用系统的 `android.os.Debug.isDebuggerConnected()` 方法和判断当前应用属性: `ApplicationInfo.FLAG_DEBUGGABLE`，那么可以依然采用全局搜索反编译之后的应用内容，找到这部分内容，然后直接注释代码即可。

第二种：修改IDA通信端口

上面分析完了，直接使用静态方式+注释代码功能解决了之前提到的三种反调试方案。但是还有两种没有解决，下面就会详细介绍一种非常靠谱方便永久的方法。而这部分内容才是本文的重点。首先来看看如何解决之前提到的利用检查IDA调试端口 `23946` 这个反调试方案。这个其实思路很简单，因为你检查的端口号是默认的 `23946`，所以我们如果能把这个端口号改成其他值，那么其实就解决了。修改这个端口号，也比较简单：网上有一种方案就是 `android_server` 本身支持自定义端口号的，命令很简单：`./android_server -p12345`；直接加上 `-p` 参数即可，注意端口号和参数之间没空格：

```
root@android:/data # ./android_server -p12345
IDA Android 32-bit remote debug server(ST) v1.19. Hex-Rays (c) 2004-2015
Listening on port #12345...
```

有的人说，这方法这么简单，那下面就不介绍了，当然不是，我写文章的目的不是为了简单，而是为了让大家了解更多的知识，宁愿多走弯路，走多条路出来。而且上面的这种方式每次都加 `-p` 比较麻烦，我想用另外一种方式去一次性解决问题，同时我更想在这个过程中熟悉一下IDA的使用，使用IDA打开 `android_server` 文件，其实他是 `elf` 格式的，打开无压力，打开之后使用 `Shift+12` 查看字符串内容界面：

Address	Length	Type	String
.text:0003B059	00000009	C	h\r\$xhtC\r<
.text:0003B0A5	00000009	C	h\r\$xhtC\r<
.text:0003B0EB	00000006	C	xaxey
.text:0003B10A	00000005	C	\rClz\t
.text:0003B14F	0000000A	C	h\r hXC\r8
.text:0003BBAB	00000007	C	#cU\a#\vp
.text:0003BC5B	00000009	C	h\r zhHC\r8
.text:0003BC9F	00000009	C	3(c\r#KC\r;
.note.android.i...	00000008	C	Android
.ARM.extab:00...	00000005	C	\r \n`
.rodata:00073F...	0000000A	C	(unknown)
.rodata:00073F38	00000060	C	=====
.rodata:00073F98	00000029	C	[%d] Could not establish the connection\n
.rodata:00073F...	0000001F	C	[%d] Incompatible IDA version\n
.rodata:00073FE4	00000013	C	[%d] Bad password\n
.rodata:00073FF8	00000024	C	[%d] Closing connection from %s...\n
.rodata:000740...	00000032	C	[%d] Debugged session entered into sleeping mode\n
.rodata:000740...	00000008	C	%s: %s\n
.rodata:000740...	0000001D	C	got signal #%d, terminating\n
.rodata:000740...	00000022	C	-k keep broken connections\n
.rodata:000740...	0000003C	C	Sorry debugger doesn't support reusing broken connections\n
.rodata:000740...	00000088	C	usage: ida_remote [switches]\n -i... IP address to bind to (default to any)\n -v verbose\n -p....
.rodata:000741...	0000000D	C	init_sockets
.rodata:000741...	00000007	C	socket
.rodata:000741...	0000003B	C	Cannot parse IP v4 address %s, falling back to INADDR_ANY\n
.rodata:000741...	00000005	C	bind
.rodata:000741...	00000007	C	listen
.rodata:000741...	0000000F	C	Host %s (%s):
.rodata:000741...	0000000A	C	Host %s:
.rodata:000741...	0000001A	C	Listening on port #%u...\n
.rodata:000741...	00000007	C	accept
.rodata:000742...	00000020	C	Could not initialize subsystem!
.rodata:000742...	0000004A	C	IDA Android 32-bit remote debug server(ST) v1.%d. Hex-Rays (c) 2004-2015\n

找到这三处关键字符串内容，我们可以通过以往运行过 `android_server` 之后的提示信息察觉：

```
root@android:/data # ./android_server
IDA Android 32-bit remote debug server(ST) v1.:
```

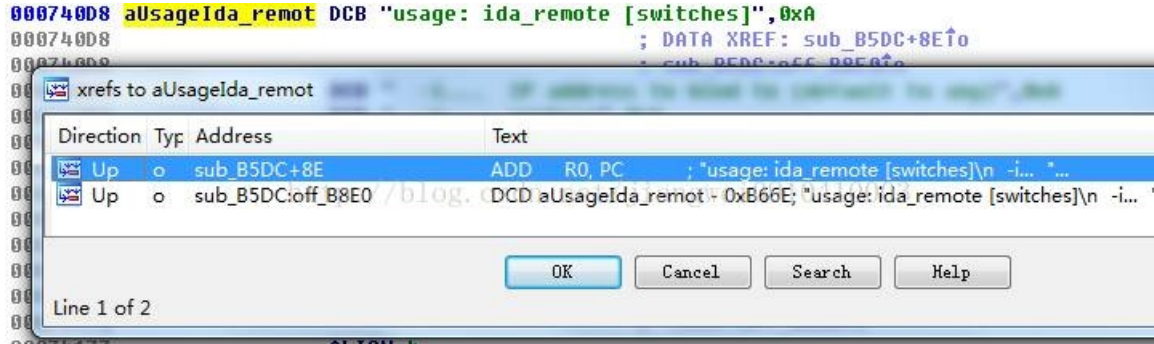
找到这三处字符串内容，下面就简单了，一处一处进行修改，双击字符串条目内容：

```

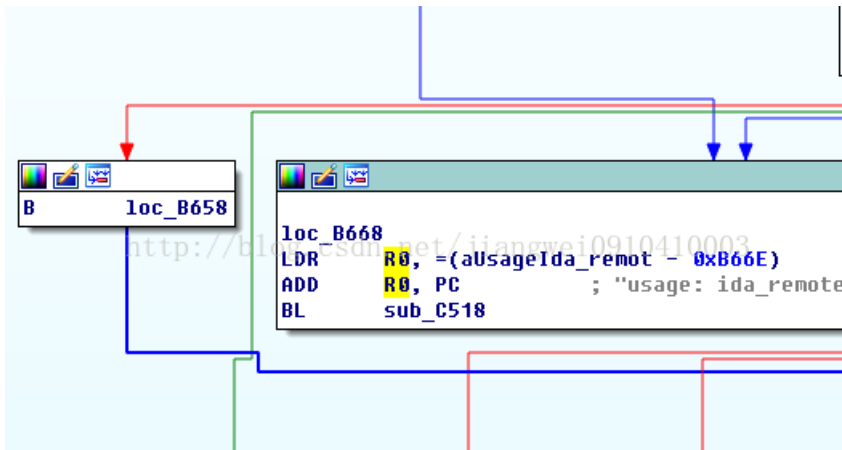
rodata:0007409C ; .text:off_BA58To
rodata:000740D8 aUsageIda_remot DCB "usage: ida_remote [switches]",0xA
rodata:000740D8 ; DATA XREF: sub_B5D0
rodata:000740D8 ; sub_B5DC:off_B8E0
rodata:000740D8 选中按X即可
rodata:000740D8 http://blog.csdn.net/jiangy10910410003
rodata:000740D8 DCB " -i... IP address to bind to (default
rodata:000740D8 DCB " -v... verbose",0xA
rodata:000740D8 DCB " -p... port number",0xA
rodata:000740D8 DCB " -P... password",0xA

```

选中按X键，进行切换：



选择第一个跳转到arm指令处：



这是graph视图，可以使用空格键进行切换：

```

.text:0000B668 loc_B668 ; CODE XREF: sub_B5DC+422↓j
.text:0000B668 LDR R0, =(aUsageIda_remot - 0xB66E)
.text:0000B66A ADD R0, PC ; "usage: ida_remote [switches]\n -i... "...
.text:0000B66C BL sub_C518
-----
.text:0000B670 ;
.text:0000B670 5D8A转化成十进制就是23946端口
.text:0000B670 loc_B670 ; CODE XREF: sub_B5DC+58↑j
.text:0000B670 ; sub_B5DC+64↑j
.text:0000B670 LDR R5, =0x5D8A
.text:0000B672
.text:0000B672 loc_B672 ; CODE XREF: sub_B5DC+2AA↓j
.text:0000B672 ; sub_B5DC:loc_B898↓j ...
.text:0000B672 LDR R4, =(sub_C3A0+1 - 0xB67A)
.text:0000B674 MOVS R0, #1

```

看到arm指令了，LDR R5, =0x5D8A；其中0x5D8A就是十进制的23946，也就是默认端口号，所以这里我们只需要将这个arm指令，改成MOVS R5, #0xDD；可对R5进行重新赋值，这里赋值为DD,也就是221；这里有个小问题就是如何进行修改，IDA中可以切换到Hex View视图进行修改编辑二进制，但是这样修改不会生效到源文件中，所以我们这里还得借助一个二进制编辑工具010Editor，我们使用这个软件打开android_server之后，使用Ctrl+G可以直接跳转到指定地址，使用Ctrl+F可以跳转到搜索内容处，记住以下这两个快捷键。

```

B660h: 00 D8 E0 E0 9D 49 79 44 9D 48 78
B670h: 99 4D 9C 4C 01 20 7C 44 21 1C 60
B680h: 02 20 60 F0 85 FC 21 1C 0F 20 60
B690h: 0B 20 60 F0 7D FC 22 F0 9B FF 00

```

这里看到了99 4D就是：LDR R5, =0x5D8A 对应的指令十六进制值，关于指令和十六进制值之间转化可以去网上搜一个小工具即可。我们想将其变成 MOVS R5, #0xDD 指令，对应的十六进制是：DD 25，其中DD就是立即数值，25表示MOVS R5指令。所以下面就可以直接进行修改即可：

```

B660h: 00 D8 E0 E0 9D
B670h: DD 25 9C 4C 01
B680h: 02 20 60 F0 85
B690h: 0B 20 60 F0 7D

```

修改完成之后，进行保存即可，这样我们就修改好了一处，还有两处操作一模一样：

```

.text:0000B97E loc_B97E ; CODE XREF: sub_B5DC+C2↑j
.text:0000B97E LDR R1, =(aInit_sockets - 0xB986)
.text:0000B980 MOVS R0, #0
.text:0000B982 ADD R1, PC ; "init_sockets"
.text:0000B984 BL sub_C360
.text:0000B988 ; -----
.text:0000B988 loc_B988 ; CODE XREF: sub_B5DC+6A↑j
.text:0000B988 LDR R3, =(dword_8074C - 0xB992)
.text:0000B98A LDR R5, =0x5D8A
.text:0000B98C MOVS R2, #0x2C
.text:0000B98E ADD R3, PC ; dword_8074C
.text:0000B990 MOV R9, R3
.text:0000B992 MOV R8, R2
.text:0000B994 MOVS R7, #1

```

这里的5D8A转化成十进制是23946

继续修改init_sockets处，命令都是一样的，记住地址：B98A，去010Editor中进行修改即可：

```

B970h: 79 44 00 F0 F5 FC 20 1C 03 F0 58 FE E7 E7 30 49
B980h: 00 20 79 44 00 F0 EC FC 2E 4B 2F 4D 2C 22 7B 44
B990h: 99 46 90 46 01 27 60 68 43 78 1A 06 13 0E 6B 2B
B9A0h: 37 D0 2D D8 50 2B 3D D0 69 2B 26 D1 27 4B 02 30

```

然后继续修改IDA Android 32-bit...处

```

.text:0000B628 LDR R3, [R3] ; unk_80BB0
.text:0000B62A ADD R0, PC ; "IDA Android 32-bit remote debug server(...
.text:0000B62C LDRB R5, [R3]
.text:0000B62E BL sub_FF88
.text:0000B632 CMP R6, #1
.text:0000B634 BLE loc_B670
.text:0000B636 LDR R2, [R4, #4]
.text:0000B638 LDRB R3, [R2]
.text:0000B63A MOVS R2, #2
.text:0000B63C BICS R3, R2
.text:0000B63E CMP R3, #0x2D
.text:0000B640 BNE loc_B670
.text:0000B642 CMP R5, #0
.text:0000B644 BNE loc_B648
.text:0000B646 B loc_B988
.text:0000B648 ; -----
.text:0000B648 loc_B648 ; CODE XREF: sub_B5DC+68↑j
.text:0000B648 LDR R3, =(dword_8074C - 0xB652)
.text:0000B64A LDR R2, =(dword_8074C - 0xB654)
.text:0000B64C LDR R5, =0x5D8A
.text:0000B64E ADD R3, PC ; dword_8074C
.text:0000B650 ADD R2, PC ; dword_8074C
.text:0000B652 MOV R9, R3
.text:0000B654 MOVS R7, #1
.text:0000B656 MOV R8, R2

```

这里的5D8A

记住地址：B64C，去010Editor进行修改即可：

```

B630h: AB FC 01 2E 1C DD 62 68 13 78 02 22 93 43 2D 2B
B640h: 16 D1 00 2D 00 D1 9F E1 A1 4B A2 4A A2 4D 7B 44
B650h: 7A 44 99 46 01 27 90 46 63 68 58 78 50 38 26 28
B660h: 00 D8 E0 E0 9D 49 79 44 9D 48 78 44 00 F0 54 FF

```


这样我们就全部改好了，保存android_server文件，再次使用IDA打开，找到一个地方查看修改是否成功：

```
.text:0000B97E          LDR    R1, =(aInit_sockets - 0xB986)
.text:0000B980          MOVS   R0, #0
.text:0000B982          ADD    R1, PC          ; "init_sockets"
.text:0000B984          BL     sub_C360
.text:0000B988          ; -----
.text:0000B988          loc_B988                ; CODE XREF: sub_B5DC+6A↑j
.text:0000B988          LDR    R3, =(dword_8074C + 0xB992)
.text:0000B98A          MOVS   R5, #0xDD
.text:0000B98C          MOVS   R2, #0x2C
.text:0000B98E          ADD    R3, PC          ; dword_8074C
.text:0000B990          MOU    R9, R3
.text:0000B992          MOU    R8, R2
.text:0000B994          MOVS   R7, #1
```

可以看到这里已经成功修改了

的确修改成功了，下面我们把android_server拷贝到设备中运行，看看端口是否为221(0xDD)：

```
root@android:/data # ./android_server
IDA Android 32-bit remote debug server(ST) v1.19. Hex-Rays (c) 2004-2015
Listening on port #221...
```

看到了，这里成功的修改了，android_server监听端口了，主要当打开IDA进行连接的时候需要注意端口是221，而不是23946了，或者你可以用adb forward tcp:221...命令进行转发也可以！

第三种：修改boot.img文件，跳过反调试

这种方式是为了解决现在常用的反调试策略，就是轮训检查进程的TracerPid值，所以我们需要修改设备的boot.img文件，将这个值直接写死为0即可。关于如何修改操作，看雪上已经有大神讲解了非常详细的过程，我就是按照这个流程进行操作的：<http://bbs.pediy.com/thread-207538.htm>，因为每个设备的boot.img都不一样，所以在操作的过程中可能遇到很多问题，所以下面就把我操作的过程中遇到的问题讲解一下，顺便精炼的说一下步骤：

第一步，你得有一个可以折腾的root手机

因为现在是在玩boot.img了，后面得刷机，所以你得搞一个你觉得没多大意义的设备，即使成砖头了也无妨。当然一般不会成为砖头。

第二步：root环境下提取zImage内核文件

这里我用的是三星note2设备，自己刷了一个CM4.4系统，按照大神的贴中先去找找到系统boot的文件位置，这个路径一定要注意：**/dev/block/platform/[每个设备目录不一样]/by-name**；其中platform目录中的子目录因为每个设备都不一样，所以需要注意，查看自己设备目录名称，然后进入到by-name之后，使用**ls -l**命令查看详情，找到一项BOOT，记住link的路径地址，这里是/dev/block/mmcblk0p8，然后使用命令，将boot导出为boot.img

```
dd if=/dev/block/mmcblk0p8 of=/data/local/boot.img
```

```
adb pull /data/local/boot.img boot.img
```

```

shell@android:/ $
shell@android:/ $ su
root@android:/ #
root@android:/ # cd /dev/block/platform
root@android:/dev/block/platform # ls
dw_mmc
root@android:/dev/block/platform # cd dw_mmc/
root@android:/dev/block/platform/dw_mmc # cd by-name
root@android:/dev/block/platform/dw_mmc/by-name # ls -l
lrwxrwxrwx root root 2012-01-13 04:19 BOOT -> /dev/block/mmcblk0p8
lrwxrwxrwx root root 2012-01-13 04:19 BOOT0 -> /dev/block/mmcblk0p1
lrwxrwxrwx root root 2012-01-13 04:19 BOOT1 -> /dev/block/mmcblk0p2
lrwxrwxrwx root root 2012-01-13 04:19 CACHE -> /dev/block/mmcblk0p12
lrwxrwxrwx root root 2012-01-13 04:19 EFS -> /dev/block/mmcblk0p3
lrwxrwxrwx root root 2012-01-13 04:19 EFS2 -> /dev/block/mmcblk0p17
lrwxrwxrwx root root 2012-01-13 04:19 HIDDEN -> /dev/block/mmcblk0p14
lrwxrwxrwx root root 2012-01-13 04:19 OTA -> /dev/block/mmcblk0p15
lrwxrwxrwx root root 2012-01-13 04:19 PARAM -> /dev/block/mmcblk0p7
lrwxrwxrwx root root 2012-01-13 04:19 RADIO -> /dev/block/mmcblk0p10
lrwxrwxrwx root root 2012-01-13 04:19 RADIO2 -> /dev/block/mmcblk0p18
lrwxrwxrwx root root 2012-01-13 04:19 RECOVERY -> /dev/block/mmcblk0p9
lrwxrwxrwx root root 2012-01-13 04:19 SYSTEM -> /dev/block/mmcblk0p13
lrwxrwxrwx root root 2012-01-13 04:19 TOMBSTONES -> /dev/block/mmcblk0p11
lrwxrwxrwx root root 2012-01-13 04:19 USERDATA -> /dev/block/mmcblk0p16
lrwxrwxrwx root root 2012-01-13 04:19 m9keys1 -> /dev/block/mmcblk0p4
lrwxrwxrwx root root 2012-01-13 04:19 m9keys2 -> /dev/block/mmcblk0p5
lrwxrwxrwx root root 2012-01-13 04:19 m9keys3 -> /dev/block/mmcblk0p6
root@android:/dev/block/platform/dw_mmc/by-name #

```

这里可能有人会遇到一个问题就是，看到多个BOOT，比如BOOT1,BOOT2，这里可以选择BOOT即可，有的人会发现没这个选项，那么只能在刷个其他系统进行操作了。

第三步：借助bootimg.exe工具解压boot.img文件

这个工具我会在后面一起给出压缩包，命令用法很简单，

解包是：`bootimg.exe --unpack-boot boot.img`

压包是：`bootimg.exe --repack-boot`

这里有一个坑，我找到两个版本，第一个版本工具操作之后刷机总是黑屏启动失败，最后找到了第二个版本工具才成功的。其实这些工具原理很简单，就是解析boot.img文件格式罢了，因为boot和recovery映像并不是一个完整的文件系统，它们是一种android自定义的文件格式，该格式包括了2K的文件头，后面紧接着是用gzip压缩过的内核，再后面是一个ramdisk内存盘，然后紧接着第二阶段的载入器程序(这个载入器程序是可选的，在某些映像中或许没有这部分)：

```

** +-----+
** | boot header | 1 page
** +-----+
** | kernel      | n pages
** +-----+
** | ramdisk     | m pages
** +-----+
** | second stage | o pages
** +-----+

```

我们想要的是kernel内核信息。所以用这个工具进行操作之后，会发现有这么几个目录和文件：

initrd	2017/3/2...	文件夹	
boot.img	2017/3/2...	光盘映像文件	16,384 KB
bootimg.exe	2015/6/2...	应用程序	3,912 KB
bootinfo.txt	2017/3/2...	文本文档	1 KB
boot-new.img	2017/3/2...	光盘映像文件	6,908 KB
boot-new_succ.img	2017/3/2...	光盘映像文件	6,908 KB
boot-old.img	2017/3/2...	光盘映像文件	16,384 KB
cpio.lst.txt	2017/3/2...	文本文档	2 KB
help.png	2015/6/2...	Kankan PNG 图像	50 KB
kernel	2017/3/2...	文件	6,539 KB
ramdisk.gz	2017/3/2...	WinRAR 压缩文...	366 KB

解压之后有一个kernel文件，这个就是内核文件，而ramdisk.gz就是释放到设备目录内容，也就是initrd目录，进入查看内容：

data	2017/3/2...	文件夹	
dev	2017/3/2...	文件夹	
proc	2017/3/2...	文件夹	
sbin	2017/3/2...	文件夹	
sys	2017/3/2...	文件夹	
system	2017/3/2...	文件夹	
default.prop	2017/3/2...	PROP 1	
file_contexts	2017/3/2...	文件	
fstab.pisces	2017/3/2...	PISCES	
init	2017/3/2...	文件	
init.cm.rc	2017/3/2...	Resour	
init.environ.rc	2017/3/2...	Resour	
init.hdcp.rc	2017/3/2...	Resour	
init.modem_jmc.rc	2017/3/2...	Resour	
init.modem_sprd.rc	2017/3/2...	Resour	
init.nv_dev_board.usb.rc	2017/3/2...	Resour	
init.pisces.rc	2017/3/2...	Resour	
init.pisces.usb.rc	2017/3/2...	Resour	
init.rc	2017/3/2...	Resour	
init.superuser.rc	2017/3/2...	Resour	
init.trace.rc	2017/3/2...	Resour	
init.usb.rc	2017/3/2...	Resour	
property_contexts	2017/3/2...	文件	
seapp_contexts	2017/3/2...	文件	
sepolicy	2017/3/2...	文件	
ueventd.pisces.rc	2017/3/2...	Resour	
ueventd.rc	2017/3/2...	Resour	

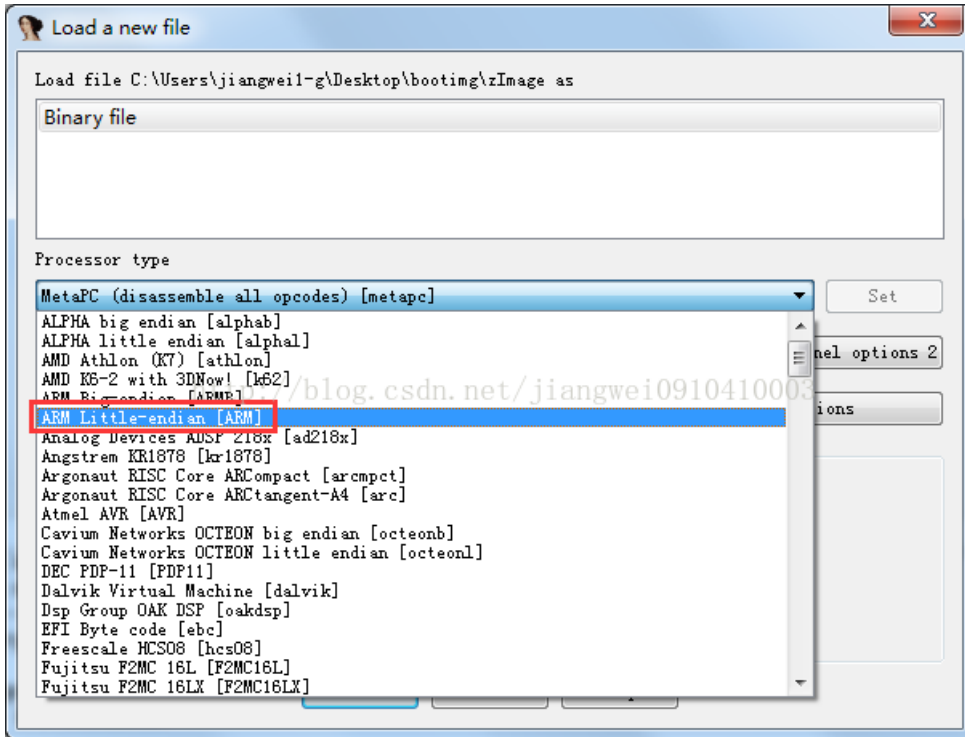
看到了吧，这就是最终设备的目录结构，可以看到这里有init.rc启动文件，default.prop配置文件等。

接下来我们就要对kernel内核文件进行特别处理了：将kernel文件复制为文件名为zimage.gz的文件，并使用010Editor工具，Ctrl+F快捷键查找十六进制内容1F 8B 08 00，找到后把前面的数据全删掉，使zimage.gz文件变成标准的gzip压缩文件，这样子就可以使用gunzip解压了。命令：gunzip zimage.gz；这时候获取到解压之后的zimage才是我们要处理的最终文件。

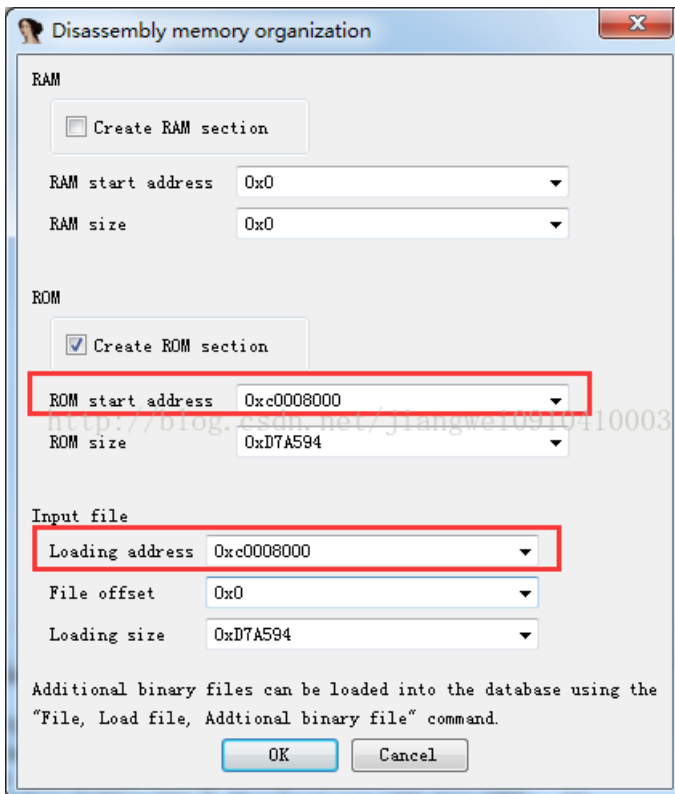
4790h:	6D 70 72 65 73 73 69 6F 6E 20 65 72 72 6F 72 00	mpressic
47A0h:	1F 8B 08 00 00 00 00 00 02 03 D4 FD 0B 78 54 D5	.<.....
47B0h:	D5 3F 8E 9F 33 97 64 32 19 C8 C9 0D 22 46 33 81	Ô?ŽŸ3-d2
47C0h:	A8 54 51 4E 00 95 DA 28 83 A2 D2 4A 75 B8 4A 2D	TQN.·Ů
47D0h:	6D 07 C5 96 F6 A5 35 5E DA FA BE B5 75 72 E1 52	m.Ā-ō¥5'
47E0h:	4A 6A 20 01 29 85 72 A8 DA 5A 6B DB 88 B6 A5 54	Jj .)....r
47F0h:	ED 41 F0 52 0B 1A 11 95 AA D4 73 E1 90 C8 58 8C	īA&R....

第四步：IDA打开zImage内核文件进行修改

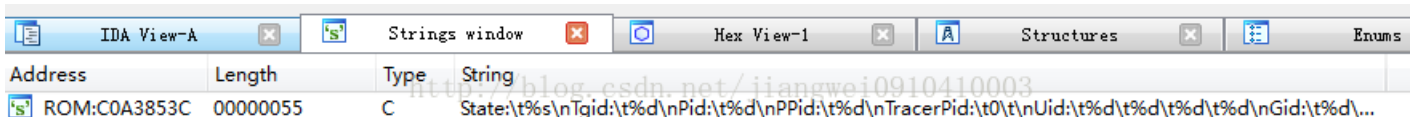
有了上面一步得到的内核文件zImage，直接使用IDA打开，但是打开的时候需要注意设置选项：



然后设置开始地址为0xC0008000:



这里为什么要设置成这个起始地址，因为Linux启动内核的地址为0xC0008000；打开之后，我们可以直接Shift+F12，查看字符串内容，因为我们想改TracerPid值，所以直接搜字符串"TracerPid"值：



双击进入，这时候我们可以记下这个地址，然后减去刚刚我们那个偏移地址0xC0008000:

```
ROM:C0A3853B DCB 0
ROM:C0A3853C aStateSTgidDPid DCB "State:",9,"%s",0xA ; DATA XREF: sub_C0186210+180fd
ROM:C0A3853C ; ROM:off_C01867C4f0
ROM:C0A3853C DCB "Tgid:",9,"%d",0xA
ROM:C0A3853C DCB "Pid:",9,"%d",0xA
ROM:C0A3853C DCB "PPid:",9,"%d",0xA
ROM:C0A3853C http:// DCB "TracerPid:",9,"0",9,0xA 0910410003
ROM:C0A3853C DCB "Uid:",9,"%d",9,"%d",9,"%d",9,"%d",0xA
ROM:C0A3853C DCB "Gid:",9,"%d",9,"%d",9,"%d",9,"%d",0xA,0
ROM:C0A38591 DCB 0, 0, 0
ROM:C0A38594 aFdsizedGroups DCB "FDSize:",9,"%d",0xA
```

也就是 $0xC0A3853C - 0xC0008000 = 0xA3053C$ ，这里没有像看雪大神操作那么复杂，先去定位函数位置，修改指令，因为每个设备不一样，指令代码就不一样，不具备通用性，所以这里有一个更好的方案：就是直接改TracerPid的格式字符串值，原始格式化字符串内容为：

```
%t%s\nTgid:%t%d\nPid:%t%d\nPPid:%t%d\nTracerPid:%t0t\nUid:%t%d\t%d\t%d\t%d\nGid:%t%d\t%d\t%d\t%d%
```

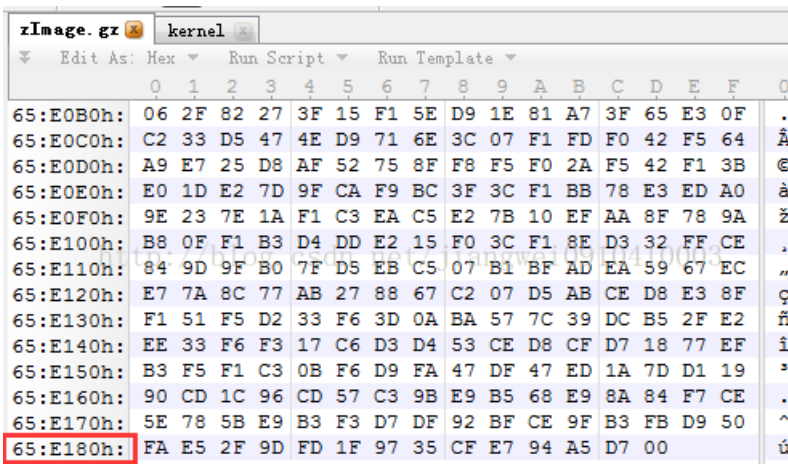
这里应该用到了C语言中的占位符%d，来进行值的填充，那么我们可以把TracerPid那一项的占位符%d，改成'0'，但是'%d'是两个字符，所以我们可以改成'00'，或者'0\t'，或者'0\n'；只要保证修改后的字符串内容对其就好。这样TracerPid这一项的值占位符就失效，值永远都是0了。而上面计算的地址就是我们要去010Editor中操作的地址，用010Editor打开zImage文件，Ctrl+G跳转到0xA3053C处：

```
A3:0540h: 65 3A 09 25 73 0A 54 67 69 64 3A 09 25 64 0A 50 e:.%s.Tgid:.%d.P
A3:0550h: 69 64 3A 09 25 64 0A 50 50 69 64 3A 09 25 64 0A id:.%d.PPid:.%d.
A3:0560h: 54 72 61 63 65 72 50 69 64 3A 09 30 09 0A 55 69 TracerPid:.0..Ui
A3:0570h: 64 3A 09 25 64 09 25 64 09 25 64 09 25 64 0A 47 (d:).%d.%d.%d.%d.G
A3:0580h: 69 64 3A 09 25 64 09 25 64 09 25 64 09 25 64 0A id:.%d.%d.%d.%d.
A3:0590h: 00 00 00 00 46 44 53 69 7A 65 3A 09 25 64 0A 47 ....FDSize:.%d.G
```

这里我们将其改成'0\t'值，对应的十六进制就是：30 09；这样我们就修改成功了。

第五步：生成修改后的boot.img文件

这里操作其实就是一个相反的过程，首先使用gzip命令压缩上面修改好的内核文件zImage：**gzip -n -f -9 zImage**；然后使用010Editor将压缩好的zImage.gz的二进制数据覆盖到原kernel文件的1F 8B 08 00处的位置(回写回去时不能改变原kernel文件的大小及修改原kernel文件后面的内容，否则会很麻烦)，这时得到了新的kernel文件内容。这里需要特别强调一下，也就是我踩过的坑：比如kernel原来是10M大小，1F8B0800之前删除的是1M，我们修改之后的zImage.gz大小是8M，那么我们回写覆盖的时候一定是1M~9M的位置，而kernel的前面1M内容和后面1M内容不能有任何改动，搞错的话，刷机会出现启动失败的情况。下面用我操作的案例讲解一下：



```
zImage.gz kernel
Edit As: Hex Run Script Run Template
0 1 2 3 4 5 6 7 8 9 A B C D E F 0
65:E0B0h: 06 2F 82 27 3F 15 F1 5E D9 1E 81 A7 3F 65 E3 0F .
65:E0C0h: C2 33 D5 47 4E D9 71 6E 3C 07 F1 FD F0 42 F5 64 Å
65:E0D0h: A9 E7 25 D8 AF 52 75 8F F8 F5 F0 2A F5 42 F1 3B ©
65:E0E0h: E0 1D E2 7D 9F CA F9 BC 3F 3C F1 BB 78 E3 ED A0 à
65:E0F0h: 9E 23 7E 1A F1 C3 EA C5 E2 7B 10 EF AA 8F 78 9A ž
65:E100h: B8 0F F1 B3 D4 DD E2 15 F0 3C F1 8E D3 32 FF CE .
65:E110h: 84 9D 9F B0 7F D5 EB C5 07 B1 BF AD EA 59 67 EC „
65:E120h: E7 7A 8C 77 AB 27 88 67 C2 07 D5 AB CE D8 E3 8F ç
65:E130h: F1 51 F5 D2 33 F6 3D 0A BA 57 7C 39 DC B5 2F E2 ñ
65:E140h: EE 33 F6 F3 17 C6 D3 D4 53 CE D8 CF D7 18 77 EF i
65:E150h: B3 F5 F1 C3 0B F6 D9 FA 47 DF 47 ED 1A 7D D1 19 ’
65:E160h: 90 CD 1C 96 CD 57 C3 9B E9 B5 68 E9 8A 84 F7 CE .
65:E170h: 5E 78 5B E9 B3 F3 D7 DF 92 BF CE 9F B3 FB D9 50 ^
65:E180h: FA E5 2F 9D FD 1F 97 35 CF E7 94 A5 D7 00 ú
```

这是我修改之后的压缩好的zImage.gz文件，最后一个数据是0x65E18D，然后全选内容复制好，记住之后，再去原来的kernel内容：

```

zImage.gz kernel
Edit As: Hex Run Script Run Template
0 1 2 3 4 5 6 7 8 9 A B C D E F 012345678
4710h: 6F 66 20 6D 65 6D 6F 72 79 20 77 68 69 6C 65 20 of memory
4720h: 61 6C 6C 6F 63 61 74 69 6E 67 20 7A 5F 73 74 72 allocatin
4730h: 65 61 6D 00 4F 75 74 20 6F 66 20 6D 65 6D 6F 72 eam.Out o
4740h: 79 20 77 68 69 6C 65 20 61 6C 6C 6F 63 61 74 69 y while a
4750h: 6E 67 20 77 6F 72 6B 73 70 61 63 65 00 00 00 00 ng worksp
4760h: 4E 6F 74 20 61 20 67 7A 69 70 20 66 69 6C 65 00 Not a gzi
4770h: 68 65 61 64 65 72 20 65 72 72 6F 72 00 00 00 00 header er
4780h: 72 65 61 64 20 65 72 72 6F 72 00 00 75 6E 63 6F read erro
4790h: 6D 70 72 65 73 73 69 6F 6E 20 65 72 72 6F 72 00 mpresion
47A0h: 1F 8B 08 00 00 00 00 00 02 03 D4 FD 0B 78 54 D5 .<.....
47B0h: D5 3F 8E 9F 33 97 64 32 19 C8 C9 0D 22 46 33 81 Ô?Žÿ3-d2.
47C0h: A8 54 51 4E 00 95 DA 28 83 A2 D2 4A 75 B8 4A 2D ``TQN.·Ú(f
47D0h: 6D 07 C5 96 F6 A5 35 5E DA FA BE B5 75 72 E1 52 m.Å-ô¥5^Ú

```

在kernel中的1F8B0800位置是0x47A0，那么我们就需要把刚刚赋值的内容从这里开始替换，到哪里结束呢？将这两个地址相加即可：0x65E18D+0x47A0=0x66292D；也就是到0x66292D结束：

```

zImage.gz kernel
Edit As: Hex Run Script Run Template
0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
66:2890h: 1F 56 2F 12 DF 83 78 57 5D C4 D3 C4 7D 88 9F A5 .V/.8fxW]AOA)^Y¥
66:28A0h: EE 11 2F 87 E7 89 77 9C 96 F9 77 26 EC FC 84 FD i./+çkwæ-ûw&iü,,ÿ
66:28B0h: AB 5E 27 3E 88 FD 6D 55 CF 3C 63 3F D7 63 BC 5B <<^'>^ymUI<c?>c4[
66:28C0h: 3D 41 3C 03 3E A8 5E 79 C6 1E 7F 8C 8F AA 97 9C =A<.>^yE.(E.ª-æ
66:28D0h: B1 EF 51 D0 BD E2 CB E1 AE 7D 11 F7 9C B1 9F BF ïiQD^sáEá@).-æ+ÿç
66:28E0h: 30 9E A6 9E 72 C6 7E BE C6 B8 67 9F AD 4F 00 5E 0ž!žrE~%E.gÿ-O.^
66:28F0h: B0 CF D6 3F FA 3E 6A D7 E8 8B CE 80 6C E6 B0 6C °IO?ú>j×è<IÈlæ°l
66:2900h: BE 1A DE 4C AF 45 4B 57 24 BC 77 F6 C2 DB 4A 9E % .PL EKW$4wóÄÜJž
66:2910h: 9D BF FE 96 FC 75 81 9C DD CF D6 BB 2F 7F E9 EC .çp-ûu.æÿIO»/.éi
66:2920h: FF 00 83 17 8D A9 94 A5 D7 00 00 00 00 00 00 00 Ÿ.f..@“¥*......
66:2930h: 00 00 00 00 00 00 00 00 2A 29 66 00 80 29 66 00 .....*)f.€)f.
66:2940h: A0 47 00 00 7C 29 66 00 68 29 66 00 94 08 00 00 G..|)f.h)f.“...
66:2950h: 70 29 66 00 74 29 66 00 78 29 66 00 6C 29 66 00 p)f.t)f.x)f.l)f.
66:2960h: 00 00 00 00 00 00 00 00 .....

```

这样原来的kernel内容大小肯定不会发生变化了，始终都是0x662967，所以在替换内容的时候内容一定不能发生变化。替换完成之后，将新的kernel文件替换原来的kernel文件，在使用之前提到的bootimg.exe工具生成新的boot.img文件即可。

第六步：刷机boot.img文件

这里有一个坑，在刷机的时候用到的是fastboot命令，但是遇到最多的问题就是这个错误：

```

C:\Users\jiangwei1-g>fastboot flash boot booting.img
< waiting for any device >

```

这个是因为设备还没有启动fastboot，关于每个设备启动fastboot不一样操作，比如小米是电源键+音量减，三星是音量减+HOME键+电源键；具体设备可以自行网上搜索即可。到了fastboot界面再次运行fastboot就可以了：

```

fastboot flash boot boot-new.img

```

```
C:\Users\jiangwei1-g\Desktop\booting\android_booting-master\android_booting-master>fastboot flash boot boot-new.img
target reported max download size of 713031680 bytes
sending 'boot' (6908 KB)...
OKAY [ 0.268s]
writing 'boot'...
OKAY [ 0.194s]
finished. total time: 0.463s
http://blog.csdn.net/jiangwei0910410003
C:\Users\jiangwei1-g\Desktop\booting\android_booting-master\android_booting-master>fastboot reboot
rebooting...

finished. total time: 0.000s

C:\Users\jiangwei1-g\Desktop\booting\android_booting-master\android_booting-master>
```

然后在运行fastboot reboot重启设备即可。有的同学在做操作的时候，始终进入fastboot失败，导致fastboot命令运行错误，这个真解决不了那就换个手机试一下吧。

这时候我们启动设备，然后调试一个app,发现他的TracerPid值永远都是0了，因为我之前将TracerPid改成'00'字符串了，也是可以：

```
root@pisces:/ # cat /proc/5966/status
Name:   ong.encryptdemo
State:  t (tracing stop)
Tgid:   5966
Pid:    5966
PPid:   215
TracerPid: 00
Uid:    4799    10077    10077    10077
Gid:    10077    10077    10077    10077
FDSize: 256
Groups: 50077
UmPeak: 893396 kB
UmSize: 893012 kB
UmLck:   0 kB
UmPin:   0 kB
UmHWM:   36728 kB
UmRSS:   36728 kB
```

因为感觉不正规，所以就有重新改成了'0t'值了。都是可以的。

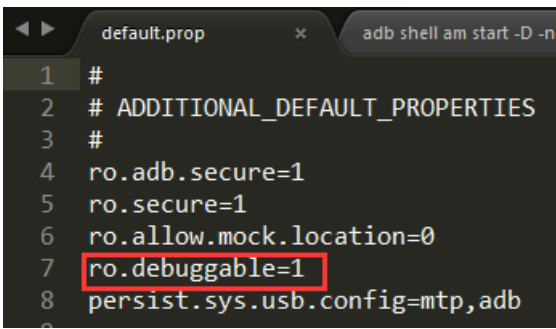
注意：一定要保存原始提取的内核文件boot.img，当你把设备弄成砖头启动失败的时候，可以在把这个原始的boot.img刷回去就可！

三、内容延展

不知道大家以前在看：[脱360加固应用的保护壳](#) 文章的时候当时说到了一个工具mprop，作用就是能够改写系统的内存中的ro.debuggable这个属性值，这样我们就没必要每次反编译app，然后在AndroidManifest.xml中添加android:debuggable="true"，让应用可调试了。

```
root@pisces:/ # getprop ro.debuggable
1
root@pisces:/ #
```

当时说到这个工具有一个弊端就是他只能修改内存中的值，当设备重启就会失效，那么现在我们可以让他永久有效，其实这个属性值，是在系统根目录下的default.prop文件中的，设备启动就会解析存入内存中。所以如果我们能够把这个文件中的值改成1，那么就永久有效了。在上面解包boot.img的时候，说到了有一个initrd目录，其实default.prop就是在这个目录下：



```
default.prop x adb shell am start -D -n
1 #
2 # ADDITIONAL_DEFAULT_PROPERTIES
3 #
4 ro.adb.secure=1
5 ro.secure=1
6 ro.allow.mock.location=0
7 ro.debuggable=1
8 persist.sys.usb.config=mtp,adb
```

这里我们直接将其改成1，因为我们现在已经进行了修改boot.img操作，那就顺便把这个功能也给改了。多方便呀！

四、提取内核操作总结

第一步：设备root之后，查看设备的内核文件路径：cd /dev/block/platform/[具体设备具体查看]/by-name，然后使用命令ls -l 查看boot属性的，记住路径

第二步：dd if=/dev/block/[你的内核路径] of=/data/local/boot.img

adb pull /data/local/boot.img boot.img

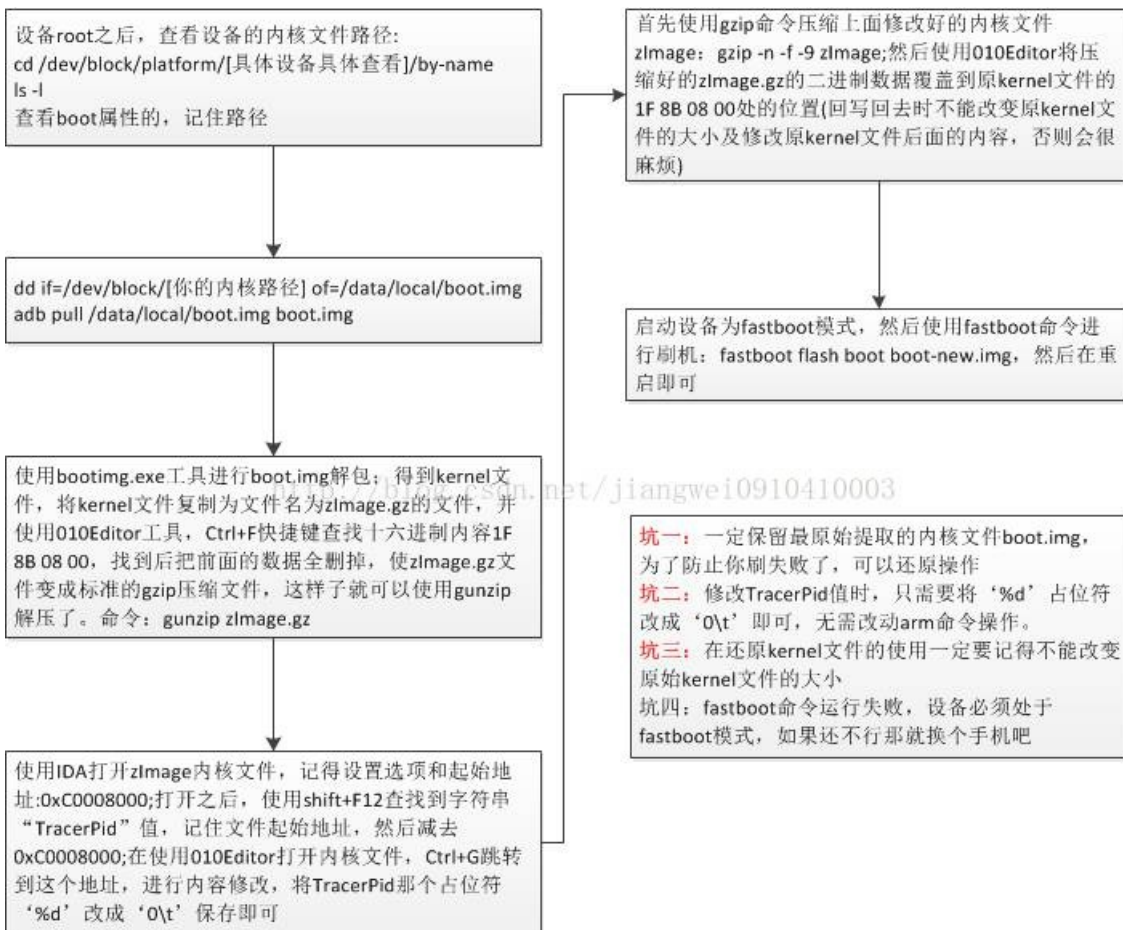
第三步：使用bootimg.exe工具进行boot.img解包；得到kernel文件，将kernel文件复制为文件名为zImage.gz的文件，并使用010Editor工具，Ctrl+F快捷键查找十六进制内容1F 8B 08 00，找到后把前面的数据全删掉，使zImage.gz文件变成标准的gzip压缩文件，这样子就可以使用gunzip解压了。命令：gunzip zImage.gz

第四步：使用IDA打开zImage内核文件，记得设置选项和起始地址:0xC0008000;打开之后，使用shift+F12查找字符串“TracerPid”值，记住文件起始地址，然后减去0xC0008000;在使用010Editor打开内核文件，Ctrl+G跳转到这个地址，进行内容修改，将TracerPid那个占位符'%d'改成'0\t'保存即可

第五步：首先使用gzip命令压缩上面修改好的内核文件zImage：gzip -n -f -9 zImage;然后使用010Editor将压缩好的zImage.gz的二进制数据覆盖到原kernel文件的1F 8B 08 00处的位置(回写回去时不能改变原kernel文件的大小及修改原kernel文件后面的内容，否则会很麻烦)

第六步：启动设备为fastboot模式，然后使用fastboot命令进行刷机：fastboot flash boot boot-new.img，然后在重启即可

总结一张图(点击查看高清无码大图):



踩过的坑

- 坑一：一定保留最原始提取的内核文件boot.img，为了防止你刷失败了，可以还原操作。
- 坑二：修改TracerPid值时，只需要将‘%d’占位符改成‘0\t’即可，无需改动arm命令操作。
- 坑三：在还原kernel文件的使用一定要记得不能改变原始kernel文件的大小。
- 坑四：fastboot命令运行失败，设备必须处于fastboot模式，如果还不行那就换个手机吧。

五、技术总结

- 第一：关于反调试的第一种解决方案比较简单，就是静态分析代码，找到反调试的位置，然后注释代码即可。
- 第二：对于监听IDA端口反调试，通过修改android_server的启动端口，这里也学会了如何修改端口号操作。
- 第三：修改内核文件，让TracerPid始终为0，ro.debuggable属性值始终为1，这个操作过程还是有点繁琐的，遇到的问题肯定很多，而且每个人遇到的问题可能不一样，但是这是一个锻炼的过程，如果成功了意味着你学会了提取内核操作，了解内核文件结构，学会分析内核文件，修改内核文件。意义重大。比如你还可以修改设备的启动图，慢慢的你可以定制自己的rom了。
- 第四：在以上操作中，也熟悉了IDA工具使用，了解到了字符串内容永远都是寻找问题的最好突破口，IDA中查找字符串Shift+F12即可，010Editor中Ctrl+G和Ctrl+F查找快捷键。

本文的目的只有一个就是学习更多的逆向技巧和思路，如果有人利用本文技术去进行非法商业获取利益带来的法律责任都是操作者自己承担，和本文以及作者没关系，本文涉及到的代码项目可以去编码美丽小密圈自取，欢迎加入小密圈一起学习探讨技术

编码美丽

微信扫一扫加入星球

 知识星球



解包boot.img文件的工具下载地址：

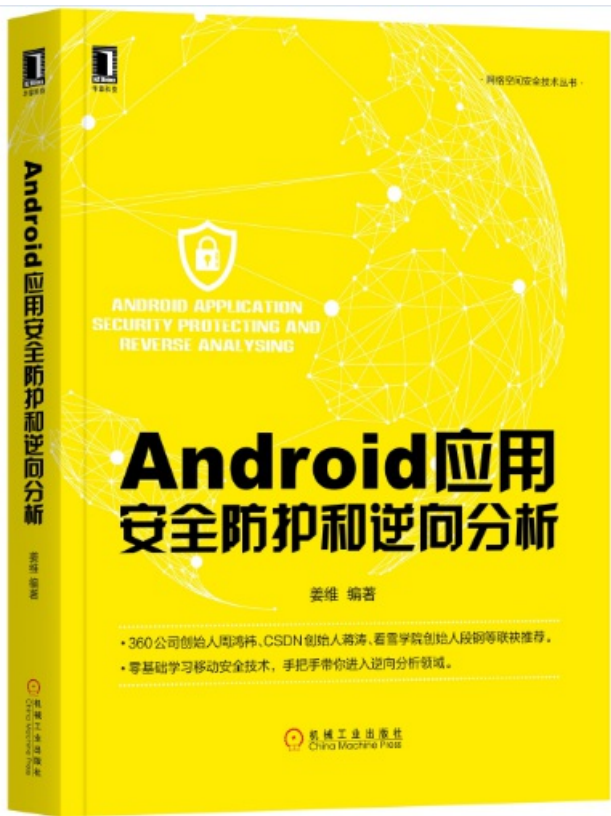
<http://download.csdn.net/detail/jiangwei0910410003/9793611>

六、总结

本文介绍的内容主要是如何解决反调试问题，主要是三种方案，最后一种修改手机内核文件的操作比较繁琐，遇到的问题也会比较多。但是如果要是成功了，以后进行破解逆向就方便多了。所以就努力看文章，自己手动操作一次。

《**Android应用安全防护和逆向分析**》

点击立即购买：[京东](#) [天猫](#)



更多内容：[点击这里](#)

关注[微信公众号](#)，最新技术干货实时推送



[创作打卡挑战赛](#) >
[赢取流量/现金/CSDN周边激励大奖](#)