Android 单机小应用程序开发实验

实验名称|开机自启动程序管理器

实验日期|2017-04-24~2017-04-26

一 实验内容

写一个程序，管理所有的具有开机自启动权限的程序（验证方法：启动应用程序，禁止微信自启动，手机启动后，然后第三方向该微信账户发送消息，微信无法收到信息，手工启动微信后，微信能收到此信息）。

二 实验环境

操作系统：

Windows 7 64-bit

开发环境：

Android Studio 2.2.3

运行环境：

设备型号：FriendlyARM Tiny4412

Android 版本：5.0.2

内核版本：3.0.86

三 实验过程

原理

在自启动实验中，我们知道 apk 通过自定义广播类接收系统的广播来实现自启动。经过查阅资料，我们使用 pm disable 去禁用应用的自定义广播类，从而达到禁止自启动的目的。这项命令的执行要求 apk 具有 system 权限或 root 权限。我们的开发板允许 App 取得这些权限。

后面的实践部分将分三个部分依次讲述：

- 获取 system/root 权限
- 获取自启程序和禁止自启程序列表以及对应广播类
- 使用 pm disable(enable) package/class

本次实验建立在 Github 上一个自启动管理项目（见【参考资料】一）的基础之上，使用了其中的视图部分（不属于重点，实践过程中不进行解释）和获取应用信息部分（属于重点，后面将在理解的基础上进行解释），对获取应用信息部分进行了理解消化，对核心部分“启动项管理”按照 system 权限和 root 权限两种方式进行了重写。

实践

一 获取权限

① 获取 root 权限

网上大量资料显示 App 可以通过 `process = Runtime.getRuntime().exec("su");`来提升至 root 权限，然而经过尝试我们发现，以 Android 给普通 App 分配的权限，如 u0_a76，无法运行系统原装的 su 程序。

首先我们想到对 su 进行逆向工程（事实上，直接阅读源码即可，这一点后面会提到），使用 adb pull 命令，我们将 Android 中的 su 提取出来。借助相关工具，看到了 su 内部包含的一些条件判断：

```
.text:0000884E                  CBZ             R0, loc_8864
.text:00008850                  CMP.W           R0, #0x7D0
.text:00008854                  BEQ             loc_8864
.text:00008856                  LDR             R1, =(aSuUidDNotAllow - 0x8860)
.text:00008858                  ADD.W           R0, R4, #0xA8
.text:0000885C                  ADD             R1, PC  ; "su: uid %d not allowed to su\n"
.text:0000885E                  BL              sub_A8B0
.text:00008862 ; 41:     goto LABEL_20;
.text:00008862                  B               loc_8968
.text:00008864 ; ---------------------------------------------------------------------------
.text:00008864 ; 43:   if ( v6 > 1 )
.text:00008864
.text:00008864 loc_8864                                 ; CODE XREF: sub_8824+2A↑j
.text:0000889E loc_889E                                 ; CODE XREF: sub_8824+4A↑j
.text:0000889E                                          ; sub_8824+6A↑j ...
.text:0000889E                  LDR             R0, [R7,#8]
.text:000088A0                  BLX             sub_CA64
.text:000088A4                  CBZ             R0, loc_88AC
.text:000088A6 ; 61:     v10 = "su: permission denied\n";
.text:000088A6
.text:000088A6 loc_88A6                                 ; CODE XREF: sub_8824+92↓j
.text:000088A6                  LDR             R0, =(aSuPermissionDe - 0x88AC)
.text:000088A8                  ADD             R0, PC  ; "su: permission denied\n"
.text:000088AA                  B               loc_8960
.text:000088AC ; ---------------------------------------------------------------------------
.text:000088AC
.text:000088AC loc_88AC                                 ; CODE XREF: sub_8824+80↑j
.text:000088AC                  LDR             R0, [R7,#4]
.text:000088AE                  BLX             sub_C9C4
.text:000088B2                  MOV             R2, R0
.text:000088B4                  CMP             R0, #0
.text:000088B6                  BNE             loc_88A6
```

然后考虑进行二进制 patch，如下：

```
.text:0000884E                  CBZ             R0, loc_8864
.text:00008850                  CMP.W           R0, #0x7D0
.text:00008854                  BNE             loc_8864
.text:00008856                  LDR             R1, =(aSuUidDNotAllow - 0x8860)
.text:00008858                  ADD.W           R0, R4, #0xA8
.text:0000885C                  ADD             R1, PC  ; "su: uid %d not allowed to su\n"
.text:0000885E                  BL              sub_A8B0
.text:00008862                  B               loc_8968

.text:0000889E loc_889E                                 ; CODE XREF: sub_8824+4A↑j
.text:0000889E                                          ; sub_8824+6A↑j ...
.text:0000889E                  LDR             R0, [R7,#8]
.text:000088A0                  BLX             sub_CA64
.text:000088A4                  CBZ             R0, loc_88AC
.text:000088A6                  LDR             R0, =(aSuPermissionDe - 0x88AC)
.text:000088A8                  ADD             R0, PC  ; "su: permission denied\n"
.text:000088AA                  MOVS            R0, R0
.text:000088AC
.text:000088AC loc_88AC                                 ; CODE XREF: sub_8824+80↑j
.text:000088AC                  LDR             R0, [R7,#4]
.text:000088AE                  BLX             sub_C9C4
.text:000088B2                  MOV             R2, R0
.text:000088B4                  CMP             R0, #0
.text:000088B6                  MOVS            R0, R0
.text:000088B8                  CMP             R6, #3
.text:000088BA                  BNE             loc_88EA
.text:000088BC                  LDR             R0, [R5,#8]
.text:000088BE                  MOV             R1, R0
.text:000088C0                  BL              sub_A658
.text:000088C4                  CMP             R0, #0
```

然而，这样生成的新 su 不能达到要求。虽然 u0_a76 用户可以成功执行 su 并打开一个 shell，但这个 shell 依然是普通用户权限。（检查方法是，在 adb shell 中 ps | grep “autostartup”获得进程 UID，然后 su UID 到普通用户，普通用户再执行 su 看是否能够到 root 用户权限，这里是不能）

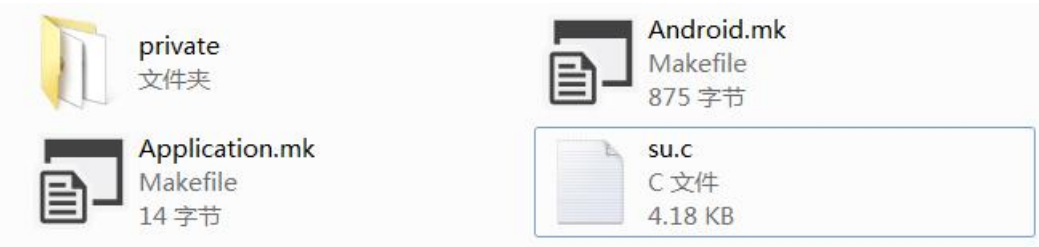
后来，在和李真希一起探索的过程中，我们在最开始编译 Android 系统的目录下的 system\extras\su 中找到了 su 的源码 su.c：

```
myuid = getuid();
if (myuid != AID_ROOT && myuid != AID_SHELL) {
    fprintf(stderr,"su: uid %d not allowed to su\n", myuid);
    return 1;
}

    if(setgid(gid) || setuid(uid)) {
        fprintf(stderr,"su: permission denied\n");
        return 1;
    }
```

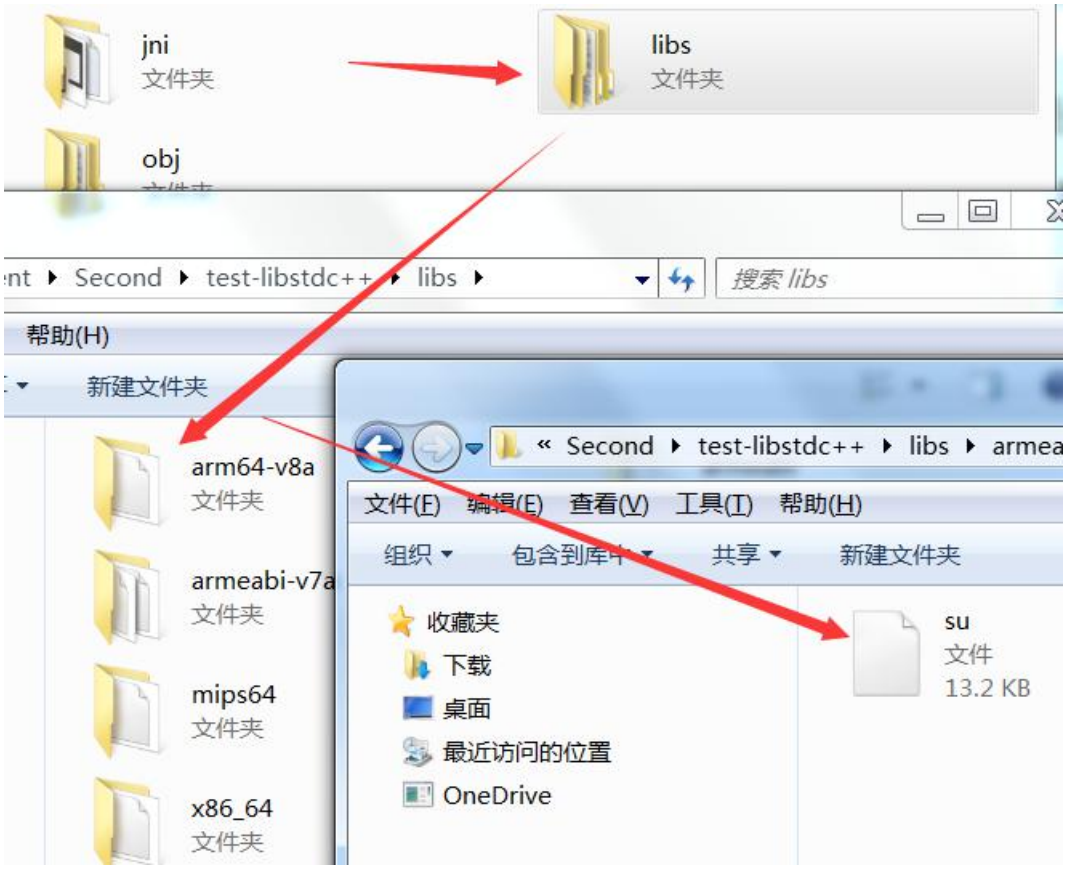
主要的限制为这两处。我们的应对措施是，将第一处的判断**注释掉**，因为 getuid 获取到的是 real UID，不是 effective UID，我们没办法通过置 suid 来绕过；但是对于 setgid 和 setuid，我们通过在 adb shell 中预先以 root 身份使用 chmod 06755 su 的方式给它加上 SUID 和 SGID 位来绕过。

所以在注释掉第一处后，我们找到它依赖的头文件，然后使用 NDK*进行编译：



```
D:\NewB\SystemExperiment\Second\test-libstdc++\jni>ndk-build
[armeabi-v7a] Install        : su => libs/armeabi-v7a/su
[armeabi] Install        : su => libs/armeabi/su
[x86] Install        : su => libs/x86/su
[mips] Install        : su => libs/mips/su
```

注意，在第一次进行编译的时候，可能由于头文件中 typedef 重定义问题报错，按照提示注释掉相关 typedef 内容即可。对于 warning 则不必理会。成功后可按下图找到 su 文件（如何获知 ABI 为 v7a？使用 adb shell "getprop ro.product.cpu.abi"我们可以获得开发板遵循的 ABI 标准）：



我们使用 adb push 命令把这个 su 放进开发板 Android 系统中，替代原有 su（首先需要 adb remount 将开发板挂载成读写模式）。接着，我们进入 adb shell，做以下操作：

```
D:\NewB\SystemExperiment\Second\test-libstdc++\jni>adb shell
root@tiny4412:/ # cd system/xbin
root@tiny4412:/system/xbin # chmod 06755 ./su
root@tiny4412:/system/xbin # chmod 06755 /system/
root@tiny4412:/system/xbin #
```

至此，root 权限获取成功。我们执行 `process = Runtime.getRuntime().exec("su");`即可以 root 权限执行操作。

*注：我们发现，使用 android-ndk-r10b 版本的 NDK 编译出来的 su 能够正常提权，使用较新版本的 NDK 编译出来的 su 在 adb shell 中能够正常提权，而 APP 通过 exec 执行却失败（似乎与 SELinux 有关，未查证）。

② 获取 system 权限

事实上，以 system 权限同样可以执行禁用/启用启动项的命令。另外，前面通过 su 获得 root 权限会给系统带来很大风险，使得任意应用都能够借助该 su 提权，带来很大风险。下面讲述通过 system 签名获取 system 权限的方法。这个方法把权限仅仅限制在我们的 APP 上。

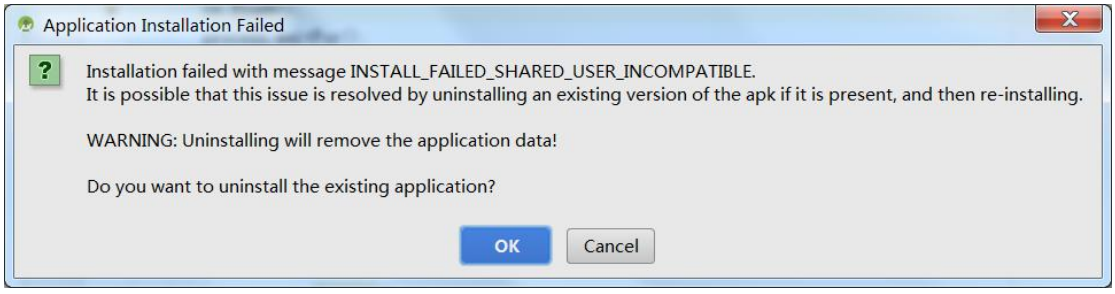
需要注意的是，由于我们不再要求获取 root 权限，所以要把 `process = Runtime.getRuntime().exec("su");`改成 `process = Runtime.getRuntime().exec("sh");`

获取 system 权限需要做两件事：

- 在 AndroidManifest.xml 中添加权限申请

```
android:sharedUserId="android.uid.system">
```

我们在 Android Studio 中生成使用 debug key 签名的 apk，会跳出一个安装失败的信息，因为我们还没有签名：



• 使用 system 签名对 APP 签名

我们找到上一步生成的 apk，到之前编译 Android 系统时的目录中找到签名文件 platform.pk8 和 platform.x509.pem，另外找到签名工具 signapk.jar，将它们和 apk 放在一起：

然后签名：

```
java -jar signapk.jar platform.x509.pem platform.pk8 app-debug.apk signed.apk
```

至此，system 权限获取成功。

二 获取自启程序广播类列表

如下图，通过调用 PackageManager 的方法来获取自启动列表，需要注意的是，我们把系统应用排除在外了：

```
allowInfoList = mPackageManager.queryBroadcastReceivers(intent,PackageManager.GET_RECEIVERS);
int k = 0;
//去除系统应用receiver
while(k < allowInfoList.size()){
    if((allowInfoList.get(k).activityInfo.applicationInfo.flags&ApplicationInfo.FLAG_SYSTEM)==1||
            (allowInfoList.get(k).activityInfo.applicationInfo.flags&ApplicationInfo.FLAG_UPDATED_SYSTEM_APP)==1){
        allowInfoList.remove(k);
    }else
        k++;
}
```

如下图，通过调用 PackageManager 的方法来获取禁止自启动列表：

```
forbidInfoList = mPackageManager.queryBroadcastReceivers(intent,PackageManager.GET_DISABLED_COMPONENTS);
int k = 0;
//去除系统应用receiver以及允许自启动的receiver
while(k < forbidInfoList.size()){
    if((forbidInfoList.get(k).activityInfo.applicationInfo.flags&ApplicationInfo.FLAG_SYSTEM)==1||
            (forbidInfoList.get(k).activityInfo.applicationInfo.flags&ApplicationInfo.FLAG_UPDATED_SYSTEM_APP)==1){
        forbidInfoList.remove(k);
    }else
        k++;
}
```

三 启用/禁用自启程序广播类

注意，上一步中我们已经得到了 package/name 这样的信息，所以，我们禁用的仅仅是自定义广播类，是 component，不是整个包，禁用整个包就不只是开机启不启动的问题了，是开机以后也启动不了。

禁用：

```
for(int j = 0; j < packageReceiverList.length; j++){
    cmd = "pm disable "+packageReceiverList[j];
    //部分receiver包含$符号，需要做进一步处理，用"$"替换掉$
    cmd = cmd.replace("$", "\""+"$"+"\"");
    //执行命令
    execCmd(cmd);
}
```

启用：

```
for(int j = 0; j < packageReceiverList.length; j++){
    cmd = "pm enable "+packageReceiverList[j];
    cmd = cmd.replace("$", "\""+"$"+"\"");
    execCmd(cmd);
}
```

执行命令的函数如下：

```
public static boolean execCmd(String cmd) {
    Process process = null;
    DataOutputStream os = null;
    try {
        process = Runtime.getRuntime().exec("sh");
        os = new DataOutputStream(process.getOutputStream());
        os.writeBytes(cmd + "\n");
        os.writeBytes("exit\n");
        os.flush();
        process.waitFor();
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    } finally {
        try {
            if (os != null) {
                os.close();
            }
            process.destroy();
        } catch (Exception e) {
        }
    }
    return true;
}
```

四 实验总结

这次实验用的时间比较长，主要耗在了将程序提升到 root 权限的问题上。总结一下，最终的解决方案就是上面提到的两种：

• 加系统签名，使用 system 权限

• 改掉 su 程序中的 uid 合法性检查步骤，并添加 suid 属性

感想也有很多，如下：

• 了解开源程序最省力的方法是源码审计而不是逆向

• 源码修改+重新编译和二进制 patch 是两种不同的思路

• 网上的文章抄来抄去，质量无法保证（如大家都提到要执行 su，可大多都省略了普通用户无法使用默认 su 这一点，也没有提出使用 system 权限就可以）

• 多用 Github 和 Google，少用百度

本次实验相当有意思，也让我们搞懂了 Android 在权限管理方面的一些机制，以及它与 Linux 权限管理的联系。也让我们学会了使用系统签名给应用签名使其具有 system 权限的方法，并掌握了使用 NDK 编译 C 语言程序并在 adb 中运行的技能，还有各种 adb 的小操作。总之，获益匪浅。

最后，感谢侯玉玺同学和李真希同学。前期对使用 su 提权无法成功的问题我们和侯玉玺同学进行了讨论，并实践了几种方法。后期修改 su 源码并使用 NDK 编译的过程主要是李真希同学完成的。我们组自己完成的是获取 system 权限禁用应用的实践。

五 参考资料

• Github: android 系统应用自启动管理

https://github.com/liwshuo/autorun

• Android 中获取应用程序(包)的信息--PackageManager 的使用

http://blog.csdn.net/qinjuning/article/details/6867806

• Android: write failed: EPIPE

http://www.it1352.com/87813.html

• 系统签名 APK，让应用获取系统级权限，后能强制开关 GPS

https://my.oschina.net/u/617626/blog/93183

• 编译调试 Android 系统原生 App － 以 Settings 为例

http://www.jianshu.com/p/691b2ad46e62

• android su 源码

http://blog.csdn.net/passerbysrs/article/details/46650253