**Root暗码实现全过程**

作者：李强 时间:2017/08/22

本文介绍暗码实现User->Root的实现全过程，从输入暗码到实现User和Root身份的转换。

基于项目平台：pop464g

Root暗码:\*#\*#212018#\*#\*

主要涉及代码目录：

Android /package/apps/Dialer/src/com/android/dialer

Android /package/apps/Dialer/src/com/android/dialer /dialpad/DialpadFragment.java

该函数实现了一个接口，该接口用于监听输入变化，该接口有是三个主要的函数：

beforeTextChanged、onTextChanged、afterTextChanged

在DialpadFragment.java中这三个函数的实现分别如下：

@Override

public void beforeTextChanged**(**CharSequence s**,** int start**,** int count**,** int after**)** **{**

mWasEmptyBeforeTextChange **=** TextUtils**.**isEmpty**(**s**);**

**}**

@Override

public void onTextChanged**(**CharSequence input**,** int start**,** int before**,** int changeCount**)** **{**

**if** **(**mWasEmptyBeforeTextChange **!=** TextUtils**.**isEmpty**(**input**))** **{**

final Activity activity **=** getActivity**();**

**if** **(**activity **!=** **null)** **{**

activity**.**invalidateOptionsMenu**();**

updateMenuOverflowButton**(**mWasEmptyBeforeTextChange**);**

**}**

**}**

**}**

关键是第三个函数与暗码相关：

@Override

public void afterTextChanged**(**Editable input**)** **{**

**if** **(!**mDigitsFilledByIntent **&&**

SpecialCharSequenceMgr**.**handleChars**(**getActivity**(),** input**.**toString**(),** mDigits**))** **{**

mDigits**.**getText**().**clear**();**

**}**

**if** **(**isDigitsEmpty**())** **{**

mDigitsFilledByIntent **=** **false;**

mDigits**.**setCursorVisible**(false);**

**}**

**if** **(**mDialpadQueryListener **!=** **null)** **{**

mDialpadQueryListener**.**onDialpadQueryChanged**(**mDigits**.**getText**().**toString**());**

**}**

updateDeleteButtonEnabledState**();**

updateActionButtonLabel**(null);**

mSuggestedAccountHandle **=** **null;**

**}**

在该函数中，调用了SpecialCharSequenceMgr**.**handleChars用来处理特殊字符串（也就是暗码）的相关处理，来看看handleChars方法都进行什么操作：

Android /package/apps/Dialer/src/com/android/dialer/ SpecialCharSequenceMgr.java

public static boolean handleChars**(**Context context**,** String input**,** EditText textField**)** **{**

//get rid of the separators so that the string gets parsed correctly

String dialString **=** PhoneNumberUtils**.**stripSeparators**(**input**);**

**if** **(**handleDeviceIdDisplay**(**context**,** dialString**)**

**||** handleRegulatoryInfoDisplay**(**context**,** dialString**)**

**||** handlePinEntry**(**context**,** dialString**)**

**||** handleAdnEntry**(**context**,** dialString**,** textField**)**

**||** handleSecretCode**(**context**,** dialString**)**

**||** handleDisplaySSVInfo**(**context**,**dialString**)**

**||** handleSalesTrackerDisplay**(**context**,** dialString**)**

**||** handleMmitestCode**(**context**,** dialString**)**

**||** showHSIInfo**(**context**,**dialString**)**

**||** handleChangeLanguage**(**context**,** dialString**)**

**||** handleSARCode**(**context**,**dialString**)** **||**handleSwVersionDisplay**(**context**,** dialString**)**

**||** handleJrdCode**(**context**,** dialString**)**

**||** handleUnlockSimLock**(**context**,** dialString**)**

**||** handleJrdDurationCode**(**context**,** dialString**)**

**||** handleEnableAutoAnswer**(**context**,** dialString**)**

**||** handleDisenableAutoAnswer**(**context**,** dialString**)**

**||** handlerGoogleApp**(**context**,** dialString**)**

**||**ExtensionManager**.**getInstance**().**getDialPadExtension**().**handleChars**(**context**,**

dialString**)**

**)** **{**

**return** **true;**

**}**

**return** **false;**

**}**

handleChars方法其实就是将不同的暗码分发给不同的函数进行处理（分析不同的暗码就从这找不同的方法就行），这里和root相关的处理函数handleSecretCode函数（也在SpecialCharSequenceMgr.java中），接着分析该函数源码：

static boolean handleSecretCode**(**Context context**,** String input**)** **{**

input **=** ExtensionManager**.**getInstance**().**getDialPadExtension**().**handleSecretCode**(**input**);**

int len **=** input**.**length**();**

**if** **(**len **>** 8 **&&** input**.**startsWith**(**"\*#\*#"**)** **&&** input**.**endsWith**(**"#\*#\*"**))** **{**

**if** **(**input**.**equals**(**ENGINEERMODE\_DISPLAY**))** **{**

Intent intent **=** **new** Intent**();**

ComponentName toActivity **=** **new** ComponentName**(**"com.mediatek.engineermode"**,**

"com.mediatek.engineermode.EngineerMode"**);**

intent**.**setComponent**(**toActivity**);**

intent**.**addFlags**(**Intent**.**FLAG\_ACTIVITY\_NEW\_TASK**);**

intent**.**addFlags**(**Intent**.**FLAG\_ACTIVITY\_SINGLE\_TOP**);**

context**.**startActivity**(**intent**);**

**return true;**

**}** **else** **{**

final Intent intent **=** **new** Intent**(**SECRET\_CODE\_ACTION**,**

Uri**.**parse**(**"android\_secret\_code://"

**+** input**.**substring**(**4**,** len **-** 4**)));**

context**.**sendBroadcast**(**intent**);**

**return** **true;**

**}**

**}**

**return** **false;**

**}**

ENGINEERMODE\_DISPLAY是\*#\*#3646633#\*#\*，判断是不是要暗码启动Engineer Mode（这里不是分析root的）。所以进入else，里面是发送了一个final类型的Intent，里面加了参数SECRET\_CODE\_ACTION(android.provider.Telephony.SECRET\_CODE)、Uri**.**parse**(**"android\_secret\_code://"**+** input**.**substring**(**4**,** len **-** 4**))**两个参数，最后发送了一个广播，有广播总有广播接受者吧，先给出AndroidManifest.Xml

<receiver android:name=**".User2RootReceiver"**>

<intent-filter>

<action android:name=**"android.provider.Telephony.SECRET\_CODE"** />

<data android:scheme=**"android\_secret\_code"** android:host=**"212018"** />

</intent-filter>

</receiver>

当输入的暗码是\*#\*#212018#\*#\*，并且SECRET\_CODE\_ACTION和intent-filter里面的相同，说明最终找到的是该广播接受者，接下来来分析广播接收函数：

android/vendor/jrdcom/apps/JrdUser2Root/src/com/jrdcom/user2root/User2RootReceiver.java

只给出了关键代码：

Intent i **=** **new** Intent**(**Intent**.**ACTION\_MAIN**);**

i**.**setComponent**(new** ComponentName**(**"com.jrdcom.user2root"**,**

"com.jrdcom.user2root.JrdUser2Root"**));**

i**.**setFlags**(**Intent**.**FLAG\_ACTIVITY\_NEW\_TASK**);**

Log**.**i**(**TAG**,**"Before start User2Root activity"**);**

context**.**startActivity**(**i**);**

启动了同一个包下的JrdUser2Root.java这个activity，该函数是具体实现user和root之间转换的核心函数。

/pop464g/vendor/jrdcom/apps/JrdUser2Root/src/com/jrdcom/user2root/User2RootReceiver.java

接下来开始源码的分析：

boolean mIsEnable**=false;**

private static final String ANDROID\_BUILD\_VERSION **=** "ro.build.version.sdk"**;**

private static final int ANDROID\_BUILD\_ICS **=** 14**;**

private static final String RO\_SECURE **=** "ro.secure"**;**

private static final String RO\_ALLOW\_MOCK\_LOCATION **=** "ro.allow.mock.location"**;**

private static final String RO\_DEBUG **=** "ro.debuggable"**;**

private static final String ADB\_ENABLE\_GB **=** "persist.service.adb.enable"**;**

private static final String ADB\_ENABLE\_ICS **=** "persist.sys.usb.config"**;**

mIsEnable：标识USB是否处于可调试状态

ANDROID\_BUILD\_VERSION：android编译的版本号

ANDROID\_BUILD\_ICS：android ICS（冰淇淋三明治）版本号14，因为在14之后添加了SEAndroid

RO\_SECURE、RO\_DEBUG：分别存储对应的系统属性，它们的值关系到ADBD的权限

RO\_ALLOW\_MOCK\_LOCATION：存储系统是否允许模拟位置的系统属性

ADB\_ENABLE\_GB：存储adb是否使能（该值是在android 4.0之前使用）

ADB\_ENABLE\_ICS：存储adb是否使能（该值是在android 4.0之后使用）

找到onCreate()函数：

@Override

public void onCreate**(**Bundle savedInstanceState**)** **{**

**super.**onCreate**(**savedInstanceState**);**

setContentView**(**R**.**layout**.**main**);**

**this.**findViews**();**

**this.**setActionListener**();**

**}**

调用了findViews**()**函数、setActionListener**()函数**

protected void findViews**(){**

**this.**mRootButton **=** **(**Button**)** **this.**findViewById**(**R**.**id**.**root**);**

**this.**mUserButton **=** **(**Button**)** **this.**findViewById**(**R**.**id**.**user**);**

**}**

protected void setActionListener**()** **{**

**this.**mRootButton**.**setOnClickListener**(this.**mRootListener**);**

**this.**mUserButton**.**setOnClickListener**(this.**mUserListener**);**

**}**

该activity上有两个button，分别是root，user表示要获取什么权限的。对这两个button进行监听。所以接下来看mRootListener

private OnClickListener mRootListener **=** **new** Button**.**OnClickListener**()** **{**

@Override

public void onClick**(**View v**)** **{**

int sdkVersion **=** SystemProperties**.**getInt**(**ANDROID\_BUILD\_VERSION**,** 10**);**

**if** **(**sdkVersion **>=** ANDROID\_BUILD\_ICS**)** **{**

toRoot\_ics**();**

**}** **else** **{**

toRoot\_gb**();**

**}**

**}**

**};**

对于mUserListener 的函数也是类似的分析，也就是在本activity里面有以下4个核心的代码块：

toRoot\_ics**()：android ICS以及之后的版本获取root权限**

toRoot\_gb**()：android gb以及之前的版本获取root权限**

toUser\_ics**()：android ICS以及之后的版本获取user权限**

toUser\_gb**()：android gb以及之前的版本获取user权限**

接下来逐一分析：

private void toRoot\_ics**()** **{**

usbdebug**();**

SystemProperties**.**set**(**RO\_SECURE**,** "0"**);**

**if(**"0"**.**equals**(**SystemProperties**.**get**(**RO\_SECURE**))){**

Log**.**i**(**TAG**,**"ro.secure is" **+** SystemProperties**.**get**(**RO\_SECURE**));**

**}else{**

Log**.**i**(**TAG**,**"can not set ro.secure"**);**

**}**

SystemProperties**.**set**(**RO\_ALLOW\_MOCK\_LOCATION**,** "1"**);**

**if(**"1"**.**equals**(**SystemProperties**.**get**(**RO\_ALLOW\_MOCK\_LOCATION**))){**

Log**.**i**(**TAG**,**"ro.allow.mock.location is" **+** SystemProperties**.**get**(**RO\_ALLOW\_MOCK\_LOCATION**));**

**}else{**

Log**.**i**(**TAG**,**"can not set ro.allow.mock.location"**);**

**}**

SystemProperties**.**set**(**RO\_DEBUG**,** "1"**);**

**if(**"1"**.**equals**(**SystemProperties**.**get**(**RO\_DEBUG**))){**

Log**.**i**(**TAG**,**"ro.debuggable is" **+** SystemProperties**.**get**(**RO\_DEBUG**));**

**}else{**

Log**.**i**(**TAG**,**"can not set ro.debuggable"**);**

**}**

SystemProperties**.**set**(**"ctl.start"**,** "disable\_selinux"**);** //set selinux passive

SystemProperties**.**set**(**"ctl.stop"**,**"adbd"**);**

SystemProperties**.**set**(**"ctl.start"**,**"adbd"**);**

**if(**mIsEnable**){**

Settings**.**Secure**.**putInt**(this.**getContentResolver**(),**

Settings**.**Secure**.**ADB\_ENABLED**,** 1**);**

**}**

Toast**.**makeText**(**JrdUser2Root**.this,** "Update to Root Success"**,** Toast**.**LENGTH\_LONG**).**show**();**

**}**

函数的开始就调用了方法usbdebug**()**：

/\*for update usbdebug state\*/

private void usbdebug**(){**

**try** **{**

mIsEnable **=true;**

int ENABLED **=** Settings**.**System**.**getInt**(this.**getContentResolver**(),**Settings**.**Secure**.**ADB\_ENABLED**);**

**if(**ENABLED**==**1**){**

Settings**.**Secure**.**putInt**(this.**getContentResolver**(),**

Settings**.**Secure**.**ADB\_ENABLED**,** 0**);**

**}**

**}** **catch** **(**Exception e**)** **{**

// TODO: handle exception

**}**

**}**

这段代码功能就是实现了usb调试功能的打开。为了该功能的实现还要在AndroidManifest.xml文件中添加相对应的权限：

<uses-permission android:name=**"android.permission.WRITE\_SETTINGS"**></uses-permission>

<uses-permission android:name=**"android.permission.WRITE\_SECURE\_SETTINGS"** />

回到toRoot\_ics**()**代码中，代码就是相关的属性进行了设置，以获取root权限，介绍修改这些属性之前，先介绍一下android属性系统的属性前缀，帮助对代码的理解：

系统属性根据不同的应用类型，分为不可变型，持久型，网络型，启动和停止服务等。

特别属性：

* 属性名称以“ro.”开头，那么这个属性被视为只读属性。一旦设置，属性值不能改变。
* 属性名称以“persist.”开头，当设置这个属性时，其值也将写入/data/property。
* 属性名称以“net.”开头，当设置这个属性时，“net.change”属性将会自动设置，以加入到最后修改的属性名。（这是很巧妙的。 netresolve模块的使用这个属性来追踪在net.\*属性上的任何变化。）
* 属性“ ctrl.start ”和“ ctrl.stop ”是用来启动和停止服务。每一项服务必须在/init.rc中定义.系统启动时，与init守护进程将解析init.rc和启动属性服务。一旦收到设置“ ctrl.start ”属性的请求，属性服务将使用该属性值作为服务名找到该服务，启动该服务。这项服务的启动结果将会放入“ init.svc.<服务名>“属性中 。客户端应用程序可以轮询那个属性值，以确定结果。

此处代码使用了SystemProperties类的get、set方法，该类是在android.os下的，是一个隐藏类，上层程序无法直接使用，修改的系统权限必须是在/system/core/init/property\_service.c中定义过的，系统属性设置还需要设置程序有system或root权限，

* 在AndroidManifest.xml中添加android:sharedUserId="android.uid.system"
* Android.mk中修改LOCAL\_CERTIFICATE := xxx为LOCAL\_CERTIFICATE := platform

通过以上两步app权限就提高到了system的权限。

修改了ro.secure、ro.debuggable这两个属性。adbd在开机的时候是root，后来被降级了，降级函数就是在这个adb\_main.cpp(/pop464g/system/core/adb)中的函数的should\_drop\_privileges()，源码如下：

static bool should\_drop\_privileges**()** **{**

#ifdef MTK\_ALLOW\_ADBD\_ROOT

**return** **false;**

#endif

#if defined(ALLOW\_ADBD\_ROOT)

char value**[**PROPERTY\_VALUE\_MAX**];**

// The emulator is never secure, so don't drop privileges there.

// TODO: this seems like a bug --- shouldn't the emulator behave like a device?

property\_get**(**"ro.kernel.qemu"**,** value**,** ""**);**

**if** **(**strcmp**(**value**,** "1"**)** **==** 0**)** **{**

**return** **false;**

**}**

// The properties that affect `adb root` and `adb unroot` are ro.secure and

// ro.debuggable. In this context the names don't make the expected behavior

// particularly obvious.

//

// ro.debuggable:

// Allowed to become root, but not necessarily the default. Set to 1 on

// eng and userdebug builds.

//

// ro.secure:

// Drop privileges by default. Set to 1 on userdebug and user builds.

property\_get**(**"ro.secure"**,** value**,** "1"**);**

bool ro\_secure **=** **(**strcmp**(**value**,** "1"**)** **==** 0**);**

property\_get**(**"ro.debuggable"**,** value**,** ""**);**

bool ro\_debuggable **=** **(**strcmp**(**value**,** "1"**)** **==** 0**);**

// Drop privileges if ro.secure is set...

bool drop **=** ro\_secure**;**

property\_get**(**"service.adb.root"**,** value**,** ""**);**

bool adb\_root **=** **(**strcmp**(**value**,** "1"**)** **==** 0**);**

bool adb\_unroot **=** **(**strcmp**(**value**,** "0"**)** **==** 0**);**

// ...except "adb root" lets you keep privileges in a debuggable build.

**if** **(**ro\_debuggable **&&** adb\_root**)** **{**

drop **=** **false;**

**}**

// ...and "adb unroot" lets you explicitly drop privileges.

**if** **(**adb\_unroot**)** **{**

drop **=** **true;**

**}**

**return** drop**;**

#else

**return** **true;** // "adb root" not allowed, always drop privileges.

#endif /\* ALLOW\_ADBD\_ROOT \*/

**}**

#endif /\* ADB\_HOST \*/

判断是否需要降权是依据的ro.secure、ro.debuggable的属性值，所以为了root权限是需要修改这两个值的。防止服务再次启动的时候进行降权。

修改了ro.allow.mock.location属性值，原因是在非root权限的条件下是不允许使用模拟位置功能的（不知道是否还有其他原因？）。只有root权限的才有该权限。在编译文件中有相关的说明：

pop464g/build/core/main.mk：

# Disallow mock locations by default for user builds

ADDITIONAL\_DEFAULT\_PROPERTIES += ro.allow.mock.location=0

修改了disable\_selinux服务的属性值，由于SELinux的强制访问控制，获取root是目的之一可能是要修改系统在user权限下不能修改的文件。但是由于强制访问控制的存在即使是有root权限也是不能达到目的的，所以修改了disable\_selinux的值关闭SELinux。

SystemProperties**.**set**(**"ctl.stop"**,**"adbd"**);**

SystemProperties**.**set**(**"ctl.start"**,**"adbd"**);**

由于修改了adbd的参数，为了让adbd不降权，所以此处重启adbd服务。最后设置Settings.Secure.ADB\_ENABLED为1，将自动打开USB调试关闭。并且通知用户root权限获取成功。

toUser\_gb()：

由于gb以及之前的版本没有SELinux，所以此处的代码就很简单，设置了persist.service.adb.enable（4.0之前用与控制adb的）、ro.secure、ro.allow.mock.location、ro.debuggable四个系统属性。

user到root分析完了，对应的root到user就简单了，把root修改过的属性值再修改回来即可。

在代码中属性值persist.sys.usb.config定义了，可是后来又注释掉了，原因是：android 4.0之后统一使用了persist.sys.usb.config来控制需要链接usb一些设置包括adb、mtp。可是这个persist.sys.usb.config的值又是依据ro.debuggable的值来设置的，设置的代码在pop464g/build/tools/post\_process\_props.py中，而ro.debuggable是在/pop464g/build/core/main.mk中设置的，所以我们在user和root之间转换只需要修改ro.debuggable的值即可。

属性介绍的时候不是说了ro属性是不能修改的吗？这里怎么能修改了，其实为了让该函数能够实现修改ro属性，在属性服务的代码中进行了相关的修改:

Pop464g/system/core/init/property\_service.cpp

/\*

\* Checks permissions for setting system properties.

\* Returns 1 if uid allowed, 0 otherwise.

\*/

check\_perms**：**

#ifdef USER2ROOT

**if(!**strcmp**(**name**,**"ro.secure"**)** **||** **!**strcmp**(**name**,**"ro.allow.mock.location"**)**

**||** **!**strcmp**(**name**,**"ro.debuggable"**)||!**strcmp**(**name**,**"persist.sys.usb.config"**)){**

**return** 1**;**

**}**

property\_set\_impl()：

**if(!**strncmp**(**name**,** "ro."**,** 3**)**

#ifdef USER2ROOT

**&&** **(**strcmp**(**name**,**"ro.secure"**))** **&&** **(**strcmp**(**name**,**"ro.allow.mock.location"**))** **&&** **(**strcmp**(**name**,**"ro.debuggable"**))** **&&** **(**strcmp**(**name**,**"ro.adb.secure"**))**

#endif

**)** **{**

#ifdef MTK\_INIT

ERROR**(**"PropSet Error:[%s:%s] ro.\* properties may NEVER be modified once set\n"**,** name**,** value**);**

#endif

**return** **-**1**;**

**}**

修改这两处的代码，对应的需要修改的4个属性就可以进行更改值了。