### 实例破解——针对Google Play License方式的破解

通过前面的分析，我们可以知道，LVL的保护机制与In-app Billing一样，都是需要由开发人员来实现的框架库。由于框架库使用Java语言编写，而且是本地验证，导致了LVL被破解是轻而易举的事情。

破解LVL的思路是直接修改LVL中验证函数的返回结果，让其永远返回为真。网上已经有人开发出了自动破解LVL的工具AntiLVL。网址为http://androidcracking.blogspot.com/（国内访问需要翻墙），目前最新版本为AntiLVL 1.4。

本小节的实例修改自Android SDk\extras\google目录下的play\_licensing工程（需另外安装），程序中用到的BASE64\_PUBLIC\_KEY是笔者从网上搜索到的，为了避免测试时可能发生扣费现象，请读者尽量在AVD中测试本实例程序，在没有安装Google Play商店的AVD上，该实例中的运行效果如图9-9所示。

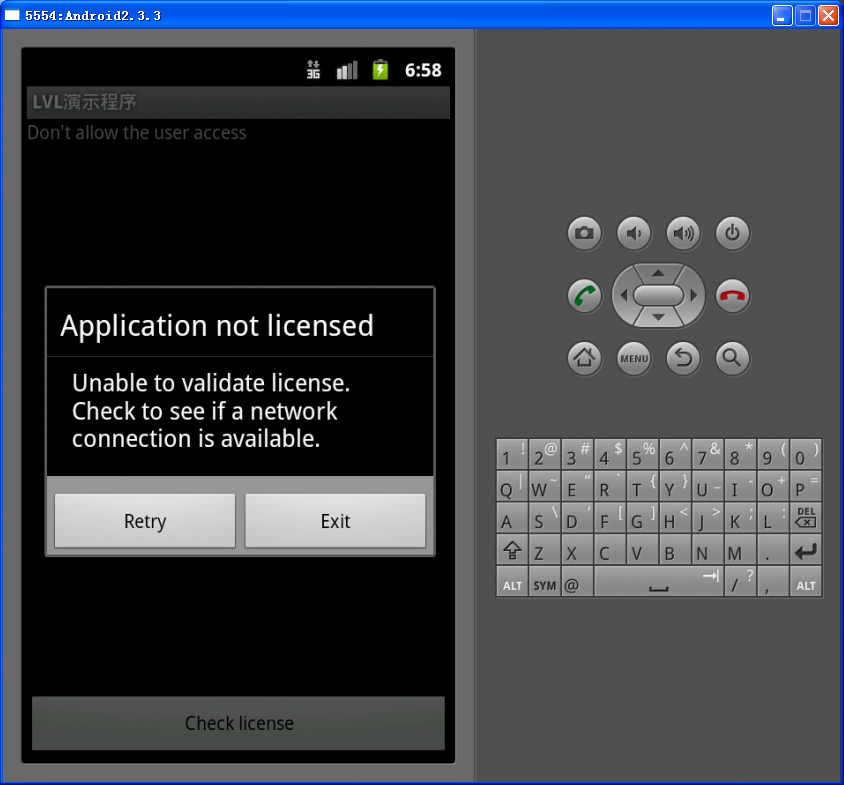


图9-9 LVL演示程序

接下来我们使用AntiLVL来破解该实例程序。将实例程序LVLSample.apk与AntiLVL放到同一目录，然后在命令提示符下执行命令：

java -jar antilvl.jar LVLSample.apk

命令执行后会生成破解后的LVLSample-antilvl.apk，并输出如下信息。

C:\Program Files\VStart50\Android\AntiLVL-1.4.0>**java -jar antilvl.jar LVLSample.apk**

-----------------------------------------------------------

AntiLVL - Android License Verification Library Subversion

Version: 1.4.0 Updated: October 18th, 2011 By: lohan+

For educational purposes only! :D

-----------------------------------------------------------

package: name='com.example.licensing' versionCode='2' versionName='1.1'

application-label:'LVLSample'

application-icon-120:'res/drawable-ldpi/ic\_launcher.png'

application-icon-160:'res/drawable-mdpi/ic\_launcher.png'

application-icon-240:'res/drawable-hdpi/ic\_launcher.png'

application-icon-320:'res/drawable-xhdpi/ic\_launcher.png'

application: label='LVLSample' icon='res/drawable-mdpi/ic\_launcher.png'

launchable-activity: name='com.example.licensing.MainActivity' label='LVLSample' icon=''

sdkVersion:'3'

uses-permission:'com.android.vending.CHECK\_LICENSE'

uses-feature:'android.hardware.touchscreen'

main

supports-screens: 'normal'

supports-any-density: 'false'

locales: '--\_--'

densities: '120' '160' '240' '320'

App package name: com.example.licensing

Uses Android licensing: yes

Determining protection mechanisms ...

Matching 41 files against 29 fingerprints

In smali\com\android\vending\licensing\APKExpansionPolicy.smali

found Server Managed Policy 1

In smali\com\android\vending\licensing\LicenseChecker.smali

found Hook Package Info

In smali\com\android\vending\licensing\LicenseValidator.smali

found License Validation Switch

In smali\com\android\vending\licensing\ServerManagedPolicy.smali

found Server Managed Policy 1

Installing hooks ..

Liberating ...

APKExpansionPolicy.smali .... success.

LicenseChecker.smali ... success.

ServerManagedPolicy.smali .... success.

Assembling ... done.

Signing ... done.

Zip aligning ... done.

Produced: LVLSample-antilvl.apk

Cleaning up ... done.

Finished.

从输出信息中可以看出，AntiLVL做了如下工作：

Determining：检测程序的LVL使用的保护类型。

Liberating：往反编译后的smali文件中插入桩代码来钩住关键方法。

Assembling：重新编译程序。

Signing：签名。

Zip aligning：对齐处理。

AntiLVL在检测LVL保护类型时，使用的方法是正则表达式匹配，具体的正则表达式是由指纹（fingerprint）配置文件来提供的。AntiLVL除了支持LVL的破解外，还支持亚马逊商店（Amazon DRM）、Verizon DRM以及自定义指纹规则形式的破解。使用jd-gui打开AntiLVL.jar，依次展开antilvl\fingerprints，会发现所有已经定义好的指纹配置文件，在fingerprints.xml文件中还描述了指纹的具体定义方法及含义，例如下面的指纹配置：

<fingerprint name="Trivial License Check 1">

<region name="tlcMethod">

<starts>(?im)^[ ]\*\.method (public |private |protected )

?(static )?(final )?(declared-synchronized )?

(hasUserDonated|isLicenseValid|isValidLicense|hasValidLicense|isLicensed|\S\*?licensed\S\*?|appLicensed|\S\*?license\S\*?valid\S\*?|\S\*?valid\S\*?license\S\*?)\(.\*?\)Z

</starts>

<ends>(?m)^\s\*\.end method</ends>

<op type="find" name="tlcMethodStarts">%!MethodStarts%</op>

<op type="insert" name="insertTLCCrack" afterOP="tlcMethodStarts">

const/4 v0, 0x1

return v0

</op>

</region>

</fingerprint>

每个fingerprint标签代表一个指纹，在指纹定义中region标签定义了一个区间，一个fingerprint可以定义多个region，在执行指纹匹配时，每个region都会被调用。region中starts标签指定了搜索的内容与起始位置，正则表达式中的相应符号含义为：

(?im)：不区分大小写，多行模式。

^[ ]\*：开头可以是多个空格

.method (public |private |protected )：方法声明开头可以是public 、private、protected之一。

?(static )?(final )?(declared-synchronized )?：可有的其它属性。

(hasUserDonated|isLicenseValid……\S\*?valid\S\*?license\S\*?)：要检查的方法名。

\(.\*?\)Z：方法的返回值必须是Z（smali语法中表示为Boolean类型）。

整段正则表达式的含义为：搜索返回值为Boolean类型的特定方法。

ends标签指定了搜索结束点（.end method），也就是当前方法声明的尾部。

op为Operation标签，即执行标签。type指定了op标签的类型，实例中前一个的op标签指定为find类型，即执行搜索功能，搜索的内容为“%!MethodStarts%”，此处的MethodStarts前后都被百分号包围，它在程序中有特殊的含义，代表方法的开始处。后一个op标签的type指定为insert，即执行插入操作，而插入的内容则为以下两行smali代码。

const/4 v0, 0x1

return v0

这两行代码转换成Java代码就是“return true”。综合上面的分析，整个fingerprint的作用就是搜索特定的方法，在方法的开头插上“return true”让它们永远返回真。

展开antilvl\hooks，可以发现AntiLVL核心功能的smali文件，这些smali文件在反编译apk时会被插入到目标apk的smali代码中，使用解压缩工具解压AntiLVL.jar，然后打开MainHook.smali文件，可以发现代码中有很多类似“%!AppPackage%”、“%!KeyPackage%”这样的字符串，这些字符串会由AntiLVL计算后替换成具体的值。

回到上面的分析，在AVD中安装破解后的LVLSample-antilvl.apk，运行后界面如图9-10所示。

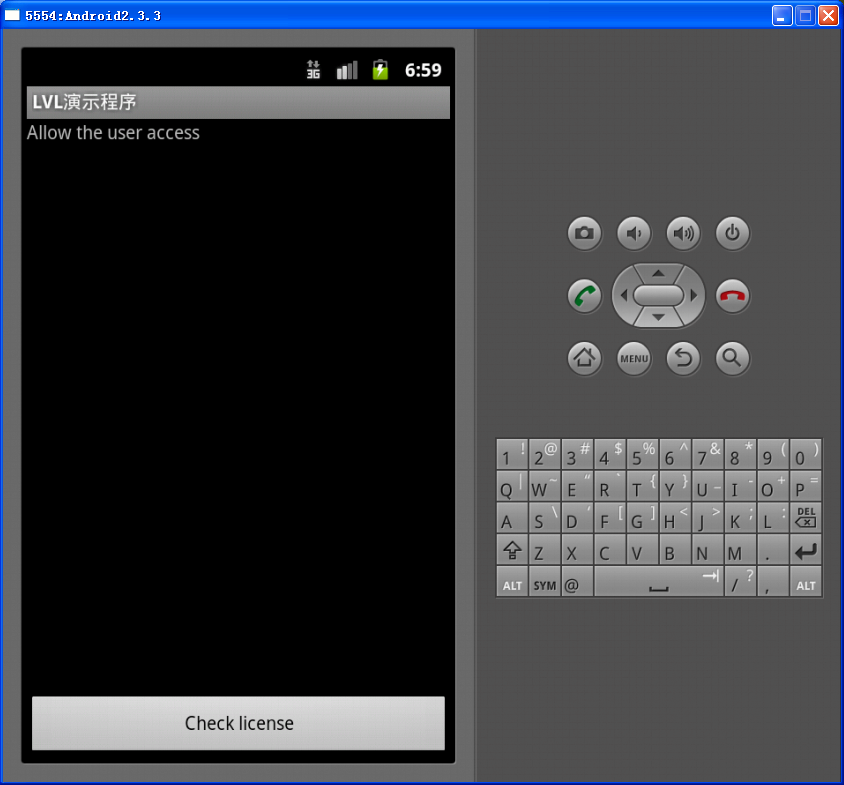


图9-10使用AntiLVL破解后的LVLSample

程序已经破解成功了！这太神奇了！AntiLVL究竟修改了哪些地方呢？从前面的分析结果中我们知道，被修改的文件有APKExpansionPolicy.smali、LicenseChecker.smali与ServerManagedPolicy.smali，具体被修改了哪里我们还需要手动的查看一下，下面请出Androguard工具。笔者启动在VirtualBox中安装有Androguard的Ubuntu虚拟机，将LVLSample.apk与LVLSample-antilvl.apk复制到Androguard安装目录下，然后打开终端提示符进入该目录并执行如下命令。

./androdiff.py -i ./LVLSample.apk ./LVLSample-antilvl.apk

androdiff.py脚本用于输出两个apk的不同点，在第5章中已经介绍过它使用方法，命令执行后输出信息如下：

<androguard.core.bytecodes.dvm.DalvikVMFormat object at 0x94c852c>

<androguard.core.analysis.analysis.VMAnalysis instance at 0x950e86c>

<androguard.core.bytecodes.dvm.DalvikVMFormat object at 0x94b86cc>

<androguard.core.analysis.analysis.VMAnalysis instance at 0x9d8dc2c>

Elements:

IDENTICAL: 104

SIMILAR: 2

NEW: 21

DELETED: 0

SKIPPED: 0

[ ('Lcom/android/vending/licensing/LicenseChecker;', 'getVersionCode',

'(Landroid/content/Context; Ljava/lang/String;)Ljava/lang/String;') ] <->

[ ('Lcom/android/vending/licensing/LicenseChecker;', 'getVersionCode',

'(Landroid/content/Context; Ljava/lang/String;)Ljava/lang/String;') ]

getVersionCode-BB@0x0 getVersionCode-BB@0x0

Added Elements(1)

0xa 3 invoke-static v1, v4, v2, LtleVBkT/znz1cznhA;->onz4n4q7IFz(

Landroid/content/pm/PackageManager; Ljava/lang/String; I)Landroid/content/pm/PackageInfo;

Deleted Elements(1)

0xa 3 invoke-virtual v1, v4, v2, Landroid/content/pm/PackageManager;->

getPackageInfo(Ljava/lang/String; I)Landroid/content/pm/PackageInfo;

Elements:

IDENTICAL: 2

SIMILAR: 1

NEW: 0

DELETED: 0

SKIPPED: 0

[ ('Lcom/android/vending/licensing/APKExpansionPolicy;', 'allowAccess', '()Z') ] <->

[ ('Lcom/android/vending/licensing/APKExpansionPolicy;', 'allowAccess', '()Z') ]

Elements:

IDENTICAL: 8

SIMILAR: 0

NEW: 1

DELETED: 0

SKIPPED: 0

<androguard.core.analysis.analysis.DVMBasicBlock instance at 0xa0be0cc> 0x0 0x4

**NEW 0x0 const/4 v0, #+1**

**NEW 0x2 return v0**

NEW METHODS

……

LtleVBkT/znz1cznhA; E2FNioQ8s9 ()V 1

LtleVBkT/znz1cznhA; YQFrSSSwr (Landroid/content/Context;)V 4

LtleVBkT/znz1cznhA; <clinit> ()V 20

LtleVBkT/znz1cznhA; l981D5mdiLdUdJXk (Ljava/lang/Object;)V 58

LtleVBkT/znz1cznhA; kG7KVksqM47PJaXWO (Ljava/io/File;)Z 40

LtleVBkT/znz1cznhA; DCbse0wdpGE5Ql7o ()Z 2

DELETED METHODS

从上面输出的信息来看，LicenseChecker类的getVersionCode()方法与APKExpansionPolicy类的allowAccess()方法被动过刀。前者的getPackageInfo()方法调用被替换成了混淆过的方法onz4n4q7IFz()，后者则直接在方法名前面插入了两行smali汇编代码，我们在前面已经分析过了，它的作用就是让allowAccess()方法永远返回真。实际上还有ServerManagedPolicy类的allowAccess()方法也被同样的方法修改过，但androdiff.py脚本似乎并没有发现这点，可能是它的一个bug吧。onz4n4q7IFz()是AntiLVL为MainHook的getPackageInfo()方法生成的随机方法名，具体的代码实现读者可以直接查看MainHook.smali方法的内容。

至此，我们已经完全了解了。AntiLVL破解LVL的方法就是直接修改allowAccess()方法的返回值，让其永远返回为真。同时可以看出，LVL在面对破解时，竟然是如此地不堪一击。