

Android客户端安全测试指南

■ 文档编号

■ 密级

内部使用

■ 版本编号 3.0

■ 日期

2015-1-6



■ 版权声明

本文中出现的任何文字叙述、文档格式、插图、照片、方法、过程等内容，除另有特别说明，版权均属绿盟科技所有，受到有关产权及版权法保护。任何个人、机构未经绿盟科技的书面授权许可，不得以任何方式复制或引用本文的任何片断。

■ 版本变更记录

时间	版本	说明	修改人
2012-8-9	1.0	创建	尚进，刘志伟
2012-11-26	1.1	更新	尚进
2013-04-02	1.2	添加“代理设置”、“证书安装”章节	尚进
2013-07-31	1.3	调整结构，内容添加	尚进
2013-09-09	2.0	完成修订	尚进
2014-08-25	2.2	添加 2.6.5 节，5.8 节 部分测试内容补充、更新	尚进
2014-10-10	2.3	添加第六章内容	尚进
2015-01-06	3.0	添加威胁等级部分 添加手势密码安全性部分 添加 2.7.6 节 修改测试项目风险定级部分内容	黄灿

nsfocus内部使用

目录

ANDROID 客户端安全测试指南	1
一. 测试环境	4
二. 安全测试列表	4
2.1 客户端程序安全	4
2.1.1 安装包签名	4
2.1.2 客户端程序保护	5
2.1.3 应用完整性检测	7
2.1.4 组件安全	8
2.1.5 *webview 组件安全	9
2.2 敏感信息安全	10
2.2.1 数据文件	10
2.2.2 logcat 日志	12
2.3 密码软键盘安全性	13
2.3.1 键盘劫持	13
2.3.2 随机布局软键盘	14
2.3.3 屏幕录像	14
2.3.4 *系统底层击键记录	15
2.4 安全策略设置	15
2.4.1 密码复杂度检测	15
2.4.2 帐号登录限制	16
2.4.3 帐户锁定策略	16
2.4.4 *私密问题验证	16
2.4.5 会话安全设置	16
2.4.6 界面切换保护	17
2.4.7 UI 信息泄漏	17
2.4.8 验证码安全性	17
2.4.9 安全退出	18
2.4.10 密码修改验证	18
2.4.11 Activity 界面劫持	18
2.5 手势密码安全性	19
2.5.1 手势密码复杂度	19
2.5.2 手势密码修改和取消	20
2.5.3 手势密码本地信息保存	20
2.5.4 手势密码锁定策略	21
2.5.5 手势密码抗攻击测试	22
2.6 进程保护	22
2.6.1 内存访问和修改	22
2.6.2 *动态注入	23

2.7	通信安全	24
2.7.1	通信加密	24
2.7.2	证书有效性	24
2.7.3	关键数据加密和校验	27
2.7.4	*访问控制	27
2.7.5	客户端更新安全性	28
2.7.6	短信重放攻击	28
2.8	业务功能测试	28
三.	测试项目风险定级	30
四.	合规性参考	32
五.	ANDROID 应用分析	33
5.1	APK 解包	33
5.2	逆向 CLASSES.DEX	33
5.2.1	反编译为 java 代码	33
5.2.2	反编译为 smali 代码	34
5.3	处理 ODEX 文件	35
5.4	反编译 SO 库	36
5.5	处理 XML	37
5.6	打包 APK	38
5.6.1	使用 apktool 打包 smali 代码	38
5.6.2	签名和优化	38
5.7	修改已安装 APK	40
5.8	内存获取 CLASSES.DEX	42
5.8.1	内存转储	42
5.8.2	ZjDroid 工具	42
5.9	ANDROID HOOK 框架	43
5.9.1	Xposed Framework	44
5.10	集成分析工具	44
5.10.1	APKAnalyser	44
5.10.2	Eclipse	48
5.10.3	Android debug monitor	51
5.10.4	apk 编辑工具	52
5.11	ANT 编译源代码	53
5.12	动态调试	53
5.12.1	使用 eclipse+ADT	53
5.12.2	使用 IDA pro	54
5.12.3	andbug 调试	54
5.13	ADB SHELL 命令	58
5.13.1	网络工具 (root)	58
5.13.2	进程查看和监视 ps/top	60
5.13.3	系统调用记录 Strace	61

5.13.4	事件操作 getevent/sendevent	61
5.13.5	截图工具 Fbtool	63
5.13.6	用户切换 Run-as/su	63
5.13.7	文件列举 lsof	64
5.13.8	数据库文件查看 sqlite3	64
5.13.9	日志查看 logcat	65
5.13.10	测试工具 Monkey	65
5.14	ANDROID 代理配置	66
5.15	手机根证书安装	68
5.16	DROZER 组件测试工具	69
六.	ANDROID 代码分析	72
6.1	ANDROID 组件功能相关代码	72
6.1.1	Content provider	72

一. 测试环境

SDK: Java JDK, Android SDK。

工具: 7zip, dex2jar, jd-gui, apktool, IDA pro (6.1), ApkAnalyser, Eclipse, dexopt-wrapper, 010 editor, SQLite Studio, ApkIDE。

apk 工具: android 组件安全测试工具, activity 劫持测试工具, android 按键记录测试工具, 代理工具 (proxydroid), MemSpector, Host Editor。

二. 安全测试列表

注: 下面的测试项目中标记*的为可选测试项, 在正式测试中可以不进行测试。

2.1 客户端程序安全

2.1.1 安装包签名

检测客户端是否经过恰当签名 (正常情况下应用都应该是签名的, 否则无法安装), 签名是否符合规范。

测试方法:

如图, 当输出结果为“jar 已验证”时, 表示签名正常。(下面的警告是因为签名密钥不在本地密钥库中)

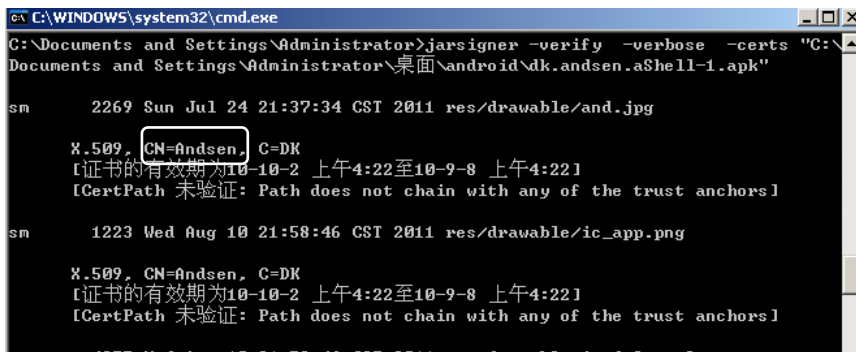


```
C:\Documents and Settings\Administrator>jarsigner -verify "C:\Documents and Settings\Administrator\桌面\android\dk.andsen.aShell-1.apk"
jar 已验证。

警告:
此 jar 包含证书链未验证的条目。

有关详细信息, 请使用 -verbose 和 -certs 选项重新运行。
```

检测签名的 CN 及其他字段是否正确标识客户端程序的来源和发布者身份:



```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Administrator>jarsigner -verify -verbose -certs "C:\Documents and Settings\Administrator\桌面\android\dk.andsen.aShell-1.apk"

sm      2269 Sun Jul 24 21:37:34 CST 2011 res/drawable/and.jpg

X.509, CN=Andsen, C=DK
[证书的有效期为10-10-2 上午4:22至10-9-8 上午4:22]
[CertPath 未验证: Path does not chain with any of the trust anchors]

sm      1223 Wed Aug 10 21:58:46 CST 2011 res/drawable/ic_app.png

X.509, CN=Andsen, C=DK
[证书的有效期为10-10-2 上午4:22至10-9-8 上午4:22]
[CertPath 未验证: Path does not chain with any of the trust anchors]
```

威胁等级:

若客户端安装包签名有异常（例如签名证书为第三方开发商而不是客户端发布方），此时高风险；若无异常则无风险。

2.1.2 客户端程序保护

1. 反编译保护

测试客户端安装程序，判断是否能反编译为源代码，java 代码和 so 文件是否存在代码混淆等保护措施。未作保护的 java 代码，可以轻易分析其运行逻辑，并针对代码中的缺陷对客户端或服务器端进行攻击。

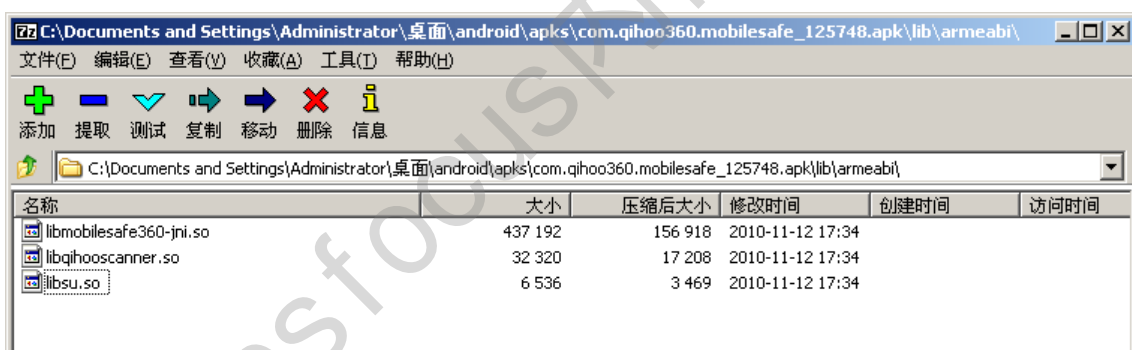
测试方法:

参考 5.1apk 解包，5.2 逆向 classes.dex，将客户端 apk 文件中的程序代码导出为 Java 代码或 smali 代码；或使用 5.10.1APKAnalyser，直接打开 apk 文件。

如下图所示，经过混淆保护的代码，其最明显的特征是大部分类和变量名都被替换为简单的 abcd 字母。



客户端程序可以把关键代码以 JNI 方式放在 so 库里。so 库中是经过编译的 arm 汇编代码，可以对其进行加壳保护，以防止逆向分析。参考 5.1apk 解包，打开 apk 文件。如果客户端程序使用了 JNI 技术，在“lib\armeabi”文件夹下会有相应的 so 库文件，如图所示：



然后在代码中查找是否加载了 so 库。例如 Java 代码：

```
Static{  
    system.loadLibrary("jni_pin");  
    system.load("./libjni_pin2.so") }  
}
```

将加载 libjni_pin.so 和 libjni_pin2.so，so 的导出函数则通过 native 关键字声明，如图所示：

```
public class MakeRldPin
{
    static
    {
        System.loadLibrary("jni_pin");
    }

    public native byte[] MakeRldPin(String paramString);

    public native String unimplementedStringFromJNI();
}
```

对 so 代码的分析，可参考 5.4 反编译 so 库。

威胁等级：

若客户端进行加壳保护，此时认为无风险。

若大部分代码（包括核心代码）经过混淆，此时低风险。

若部分代码混淆，关键代码（加密或通信等）可以获知其关键代码，此时中风险。

2.1.3 应用完整性检测

测试客户端程序是否对自身完整性进行校验。攻击者能够通过反编译的方法在客户端程序中植入自己的木马，客户端程序如果没有自校验机制的话，攻击者可能会通过篡改客户端程序窃取手机用户的隐私信息。

测试方法：

参考 5.10.2 Eclipse 关于 DDMS 的文件操作和 5.7 修改已安装 apk。推荐修改 apk 中 assets 目录下或 res/raw 目录下的文件。将修改后的 apk 文件导入到/data/app 目录下，覆盖原文件，然后重启客户端，观察客户端是否会提示被篡改。

* 或在 Java 代码中查找是否包含校验功能。

威胁等级：

若应用完整性校验不使用 MANIFEST.MF 中的数据，且核心代码通过 JNI 技术写入.so 库，同时于服务端进行相关校验，此时无风险。

若应用完整性于本地进行验证而不存在其他问题或使用 MANIFEST.MF 中的数据作为验证凭证（有新文件时提示应用完整性验证失败），此时低风险；若在本地进行验证的基础上只通过 MANIFEST.MF 对客户端原有文件进行校验而忽略新增文件的检验，此时中风险；若未进行应用完整性校验此时高风险。

2.1.4 组件安全

测试客户端是否包含后台服务、Content Provider、第三方调用和广播等组件，Intent 权限的设置是否安全。应用不同组成部分之间的机密数据传递是否安全。

测试方法：

检查 AndroidManifest.xml 文件中各组件定义标签的安全属性是否设置恰当。如果组件无须跨进程交互，则不应设置 exported 属性为 true。例如，如下图所示，当 MyService 的 exported 属性为 true 时，将可以被其他应用调用（当有设置权限(permissions)时，需要再考察权限属性。如 android:protectionLevel 为 signature 或 signatureOrSystem 时，只有相同签名的 apk 才能获取权限。参考 [SDK](#)）。



```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="1" android:versionName="1.0" package="com.emit.aic"
  xmlns:android="http://schemas.android.com/apk/res/android">
  <uses-sdk android:minSdkVersion="10" />
  <application android:label="@string/app_name" android:icon="@drawable/ic_laur"
    <activity android:label="@string/app_name" android:name=".AidlDemoActivit"
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <service android:name=".MyService" android:exported="true">
      <intent-filter>
        <action android:name="com.emit.aidl.server.COMMON_OPERATION" />
      </intent-filter>
    </service>
  </application>
</manifest>
```

可以使用“组件安全测试工具”来检测组件的 exported 属性，如图所示。凡是列出来的组件都是 exported 属性为 true 的。点击 Save 按钮可以把检测结果保存在 SD 卡上。



或者使用 [Dexter](#) 在线检测环境（或 [sanddroid](#)）来做，如图所示，Exported 为对号的是已经导出的组件，可能存在安全问题。（注意：Dexter 对 Content Provider 判断不一定准确。）

NAME	SYSTEM INSTANTIABLE	EXPORTED
cn.com.cmbc.mbank.twodimensioncode.encoding.CaptureActivity	✓	✓
com.hexin.plat.android.AndroidLogoActivity	✓	✓
com.hexin.plat.android.Hexin	✓	✗
com.hexin.plat.android.LoginAndRegisterActivity	✓	✗
com.hexin.plat.android.WebViewActivity	✓	✗

NAME	SYSTEM INSTANTIABLE	EXPORTED
com.hexin.plat.android.CommunicationService	✓	✗
com.hexin.plat.android.LogoUpgradeService	✓	✗

NAME	SYSTEM INSTANTIABLE	EXPORTED
com.hexin.app.QuitWeiTuoReceiver	✓	✗

当发现有可利用的组件导出时，可参考 5.16 drozer 测试工具进行测试。

注意：不是所有导出的组件都是不安全的，如需确定须看代码，对代码逻辑进行分析。

注：有些应用在代码中动态注册组件，这种组件无法使用“组件安全测试工具”测试，需要通过阅读代码确定是否安全。

关于 [Android SDK](#) 中对 exported 属性的默认设置说明：对 service, activity, receiver，当没有指定 exported 属性时，没有过滤器则该服务只能在应用程序内部使用，相当于 exported 设置为 false。如果至少包含了一个过滤器，则意味着该服务可以给外部的其他应用提供服务，相当于 exported 为 true。对 provider，SDK 小于等于 16 时，默认 exported 为 true，大于 16 时，默认为 false。（某些广播如 android.intent.action.BOOT_COMPLETED 是例外）

威胁等级：

若不存在组件暴露的情况，此时无风险。

如存在组件暴露的情况，但暴露的组件无关客户端逻辑核心或不会泄露用户敏感信息，此时低风险；若暴露的组件会泄露用户敏感信息（例如邮件客户端存在消息组件的暴露，攻击者可以通过编写 APK，通过组件利用的方式读取用户邮件信息）

2.1.5 *webview 组件安全

Android 4.2 版本以下的 webview 组件存在安全漏洞（[CVE-2012-6636](#)）。检测客户端是否采取措施避免漏洞被利用。

测试方法：

检查应用 AndroidManifest.xml 中的 targetSdkVersion 是否大于等于 17。

```
<uses-sdk
    android:minSdkVersion="14"
    android:targetSdkVersion="17" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
```

或者使用测试网页进行测试（腾讯的测试页面链接，在被测应用中打开即可。

http://security.tencent.com/lucky/check_tools.html）。

威胁等级：

当存在 **sdk** 版本太低时存在 **webview** 组件漏洞被利用的可能，此时中风险。

当版本高时无风险。

2.2 敏感信息安全

检测客户端是否保存明文敏感信息，能否防止用户敏感信息的非授权访问。对 **android** 系统的文件导出可参考 5.10.2Eclipse。

对 **android** 的每一个应用，**android** 系统会分配一个私有目录，用于存储应用的私有数据。此私有目录通常位于“ / **data** / **data** / 应用名称 / ”。

在测试时，建议完全退出客户端后，再进行私有文件的测试，以确保测试结果的准确性。（有些客户端在退出时会清理临时文件）

2.2.1 数据文件

1. 检查私有目录下的文件权限。

测试方法：

正常的文件权限最后三位应为空（类似“**rw-rw----**”），即除应用自己以外任何人无法读写；目录则允许多一个执行位（类似“**rwxrwx—x**”）。如下图所示，**script** 文件的权限设置不安全，**script** 可以被任意应用读取。（**lib** 子目录是应用安装时由 **android** 系统自动生成，可以略过）

```

drwxr-xr-x system system 2013-08-01 11:21 lib
drwxrwx--x app_34 app_34 2013-07-23 14:28 cache
drwxrwx--x app_34 app_34 2013-07-23 14:28 files
-rwx----- app_34 app_34 198652 2013-07-23 14:28 iptables
-rwx----- app_34 app_34 130276 2013-07-23 14:28 redsocks
-rwx----- app_34 app_34 1927 2013-07-23 14:28 proxy.sh
-rwx----- app_34 app_34 84916 2013-07-23 14:28 cntlm
-rwx----- app_34 app_34 22235 2013-07-23 14:28 tproxy
-rwx----- app_34 app_34 657016 2013-07-23 14:28 stunnel
drwxrwx--x app_34 app_34 2013-08-02 05:12 shared_prefs
-rwxr-xr-x app_34 app_34 114 2013-08-02 05:12 script
-rw----- app_34 app_34 76 2013-08-02 05:12 default
drwxrwx--x app_34 app_34 2013-08-01 17:37 databases
#

```

注意：当客户端使用 `MODE_WORLD_READABLE` 或 `MODE_WORLD_WRITEABLE` 模式创建文件时，`shared_prefs` 目录下文件的权限也会多出一些，这不一定是安全问题（Google 已不推荐使用这些模式）

```

ls -l
-rw-rw-rw- app_46 app_46 103 2013-08-02 08:17 test3.xml
-rw-rw-rw- app_46 app_46 103 2013-08-02 08:10 test2.xml
-rw-rw-rw- app_46 app_46 101 2013-08-02 07:59 com.example.filetest_preferences.xml
#

```

2. 检查客户端程序存储在手机中的 `SharedPreferences` 配置文件。

威胁等级：

当权限存在问题时低风险，不存在问题时无风险。

测试方法：

对本目录下的文件内容（通常是 `xml`）进行检查，看是否包含敏感信息。

3. 检查客户端程序存储在手机中的 `SQLite` 数据库文件。

测试方法：

在私有目录及其子目录下查找以 `.db` 结尾的数据库文件。对于使用了 `webView` 缓存的应用，会在 `databases` 子目录中保存 `webview.db` 和 `webviewCache.db`，如图所示。其中有可能记录 `cookies` 和提交表单等信息。

```

├── app_database
│   ├── CachedGeoposition.db
│   └── Databases.db
├── localstorage
│   ├── file_0.localstorage
│   └── http_www.baidu.com_0.localstorage
├── cache
│   └── webviewCache
│       ├── 8cf40983
│       └── 9a68c995
├── databases
│   ├── webview.db
│   └── webviewCache.db
├── lib
└── shared_prefs
    └── WebViewSettings.xml

```

File Name	Size	Modified	Permissions
app_database		2012-10-24 14:11	drwxr-xr-x
CachedGeoposition.db	0	2012-10-24 14:11	-rw-rw-rw-
Databases.db	0	2012-10-24 14:11	-rw-rw-rw-
localstorage		2012-10-26 12:46	drwxr-xr-x
file_0.localstorage	4096	2012-10-25 22:27	-rw-rw-rw-
http_www.baidu.com_0.localstorage	131072	2012-10-26 12:46	-rw-rw-rw-
cache		2012-10-26 15:31	drwxr-xr-x
webviewCache		2012-10-26 15:55	drwxr-xr-x
8cf40983	22614	2012-10-26 15:55	-rw-rw-rw-
9a68c995	1563	2012-10-26 15:44	-rw-rw-rw-
databases		2012-10-26 15:58	drwxr-xr-x
webview.db	14336	2012-10-26 15:58	-rw-rw-rw-
webviewCache.db	6144	2012-10-26 15:55	-rw-rw-rw-
lib		2012-10-25 21:01	drwxr-xr-x
shared_prefs		2012-10-24 14:43	drwxr-xr-x
WebViewSettings.xml	118	2012-10-24 14:43	-rw-rw-rw-

参考 5.13.8 数据库文件查看 `sqlite3`，或是使用 `SQLite Studio`，查看、修改数据库文件。

威胁等级：

若私有目录中存在存储了用户登陆密码（明文或只进行过一次单项哈希散列），手势密码（明文或只进行过一次单项哈希散列）或曾经访问过网址的 `Cookie` 等敏感信息的文件，此时为高风险，若不存在则无风险。

4. 检查客户端程序 `apk` 包中是否保存有敏感信息。

测试方法：

参考 5.1 `apk` 解包，5.4 反编译 `so` 库和 5.2 逆向 `classes.dex`，检查 `apk` 包中各类文件是否包含硬编码的敏感信息。对可执行文件可通过逆向方法寻找，也可以直接使用 16 进制编辑器查找。

5. 检查客户端程序的其他文件存储数据，如缓存文件和外部存储。

测试方法：

参考 5.10.2 `Eclipse`，在应用的私有目录以及 `SD` 卡中包含应用名称的子目录中进行遍历，检查是否有包含敏感信息的文件。

参考 5.13.3 系统调用记录 `Strace`，查找应用和文件 `IO` 相关的系统调用（如 `open`，`read`，`write` 等），对客户端读写的文件内容进行检查。

6. 检查手机客户端程序的敏感信息是否进行了加密，加密算法是否安全。

测试方法：

查找保存在应用私有目录下的文件。检查文件中的数据是否包含敏感信息。如果包含非明文信息，在 `Java` 代码中查找相应的加密算法，检查加密算法是否安全。（例如，采用 `base64` 的编码方法是不安全的，使用硬编码密钥的加密也是不安全的。）

2.2.2 logcat 日志

检查客户端程序存储在手机中的日志。

测试方法：

参考 5.10.2 `Eclipse` 或 5.13.9 日志查看 `logcat`，检查日志是否包含敏感信息。

威胁等级：

当 `Logcat` 日志中会显示用户输入的信息（包括用户名、明文密码或单次哈希的密码）、用户访问服务器的 `URL` 和端口等核心敏感信息时为高风险；当 `Logcat` 日志中会显示调用逻辑或一些可供攻击者猜测逻辑的报错时为中风险；当 `Logcat` 日志中会打出除上述外的一些开发者的调试信息时为低风险；如果不存在上述情况则无风险。

2.3 密码软键盘安全性

2.3.1 键盘劫持

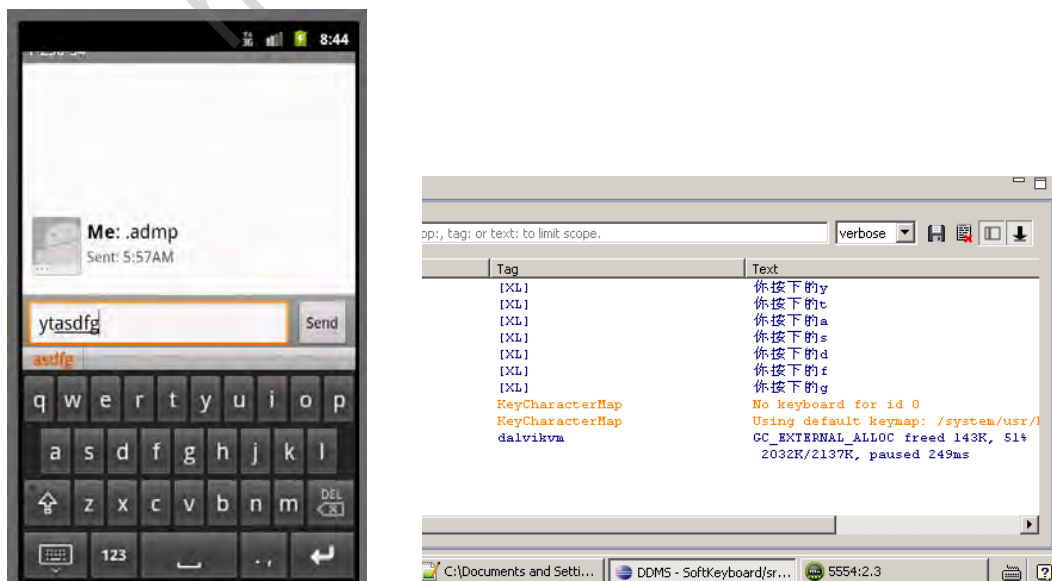
测试客户端程序在密码等输入框是否使用自定义软键盘。安卓应用中的输入框默认使用系统软键盘，手机安装木马后，木马可以通过替换系统软键盘，记录手机银行的密码。

测试方法：

安装 android 击键记录测试工具。然后在“语言和键盘设置”中选择“Sample Soft Keyboard”。然后启动客户端，在输入框长按，弹出提示框后选择“input method”（输入法），选择我们安装的软键盘。



下图是书写短信息时，使用软键盘输入，在 logcat 日志中可以看到所有的击键。



威胁等级:

当客户端不存在自定义而是使用系统默认键盘时为中风险，客户端存在自定义软键盘时无风险。

2.3.2 随机布局软键盘

测试客户端实现的软键盘，是否满足键位随机布放要求。

测试方法:

人工检测。

威胁等级:

当客户端软键盘未进行随机化处理时为低风险；当客户端软键盘只在某一个页面载入时初始化一次而不是在点击输入框时重新进行随机化也为低风险。

2.3.3 屏幕录像

客户端使用的随机布局软键盘是否会对用户点击产生视觉响应。当随机布局软键盘对用户点击产生视觉响应时，安卓木马可以通过连续截屏的方式，对用户击键进行记录，从而获得用户输入。

测试方法:

使用现有的 android 截屏工具，连续截取屏幕内容，测试能否记录客户端软键盘输入。

检测需较高安全性的窗口（如密码输入框），看代码中在窗口加载时是否有类似下图的代码。按照 android SDK 的要求，开启 FLAG_SECURE 选项的窗口不能被截屏。

注意：FLAG_SECURE 可能存在兼容性问题，能否防护截图可能与硬件有关。

```
public class FlagSecureTestActivity extends Activity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        getWindow().setFlags(LayoutParams.FLAG_SECURE,  
                             LayoutParams.FLAG_SECURE);  
  
        setContentView(R.layout.main);  
    }  
}
```

（目前 FLAG_SECURE 测试结果：

N-PASS, 可截图,

ZTE 880E, 可截图

ASUS TF300T, 可阻止工具及 ddms 截图)

威胁等级:

当使用第三程序（或系统截屏）可以对客户端内容进行截屏时，为中风险；当客户端会对截屏操作进行有效抵抗时（无法截屏或截屏结果为黑屏等无意义图片）无风险。

2.3.4 *系统底层击键记录

拥有 root 权限后，安卓木马可以通过读取系统文件/dev/input/eventN 得到键盘码，从而获得用户输入。

注意：目前很多 android 系统不再向 event 文件输出键盘码，如需测试需先确定键盘输入对应的 event 文件是否存在。

测试方法:

运行客户端，在输入密码的同时，参考 5.13.4 事件操作 getevent/sendevent，在 shell 中使用命令行监控输入。

2.4 安全策略设置

2.4.1 密码复杂度检测

测试客户端程序是否检查用户输入的密码强度，禁止用户设置弱口令。

测试方法:

人工测试，尝试将密码修改为弱口令，如：123456，654321，121212，888888 等，查看客户端是否拒绝弱口令。

* 阅读逆向后的客户端 java 代码，寻找对用户输入口令的检查方法。

威胁等级:

当系统允许用户设置弱密钥时为低风险，如果存在系统存在一定安全策略（密码使用数字和字母组成，至少为 8 位）时无风险。

2.4.2 帐号登录限制

测试一个帐号是否可以同时在多个设备上成功登录客户端，进行操作。

测试方法：

人工测试。

威胁等级：

若同一个账号可以同时在多台移动终端设备上登陆时为低风险，若不可以则无风险。

2.4.3 帐户锁定策略

测试客户端是否限制登录尝试次数。防止木马使用穷举法暴力破解用户密码。

测试方法：

人工测试。

威胁等级：

当系统不存在账户锁定策略时为中风险，若存在则无风险。

2.4.4 *私密问题验证

测试对账号某些信息（如单次支付限额）的修改是否有私密问题验证。私密问题验证是否将问题和答案一一对应。私密问题是否足够私密。

测试方法：

人工测试。

威胁等级：

当用户进行忘记密码操作时，在发送邮件给用户邮箱前是否进行私密问题的验证，若验证则无风险；若不验证则低风险。

2.4.5 会话安全设置

测试客户端在超过 20 分钟无操作后，是否会使会话超时并要求重新登录。超时时间设置是否合理。

测试方法：

人工测试。

威胁等级：

当系统不存在会话超时逻辑判断时为低风险，若存在则无风险。

2.4.6 界面切换保护

检查客户端程序在切换到其他应用时，已经填写的账号密码等敏感信息是否会清空，防止用户敏感信息泄露。如果切换前处于已登录状态，切换后一定时间内是否会自动退出当前会话。

测试方法：

人工检测。在登录界面（或者转账界面等涉及密码的功能）填写登录名和密码，然后切出，再进入客户端，看输入的登录名和密码是否清除。登录后切出，5 分钟内自动退出为安全。

威胁等级：

当移动终端设备进行进程切换操作，显示界面不为客户端页面时，若客户端提示用户确认是否为本人操作，则无风险；若无相应提示则为低风险。

2.4.7 UI 信息泄漏

检查客户端的各种功能，看是否存在敏感信息泄露问题。

测试方法：

人工测试。使用错误的登录名或密码登录，看客户端提示是否不同。在显示卡号等敏感信息时是否进行部分遮挡。

威胁等级：

若在用户名输入错误和密码输入错误时提示信息不同则存在 UI 信息泄露问题，此时为低风险，否则无风险。

2.4.8 验证码安全性

测试客户端在登录和交易时是否使用图形验证码。验证码是否符合如下要求：由数字和字母等字符混合组成；采取图片底纹干扰、颜色变换、设置非连续性及旋转图片字体、变异

字体显示样式等有效方式，防范恶意代码自动识别图片上的信息；具有使用时间限制并仅能使用一次；验证码由服务器生成，客户端文件中不包含图形验证码文本内容。

测试方法：

人工测试。

威胁等级：

当图形验证码由本地生成而不是从服务器获取时为中风险；当验证码安全性低或不存在验证码时为中风险；不存在以上两个问题时无风险。

2.4.9 安全退出

测试客户端退出时是否正常终止会话。

测试方法：

检查客户端在退出时，是否向服务端发送终止会话请求。客户端退出后，还能否使用退出前的会话 id 访问登录后才能访问的页面。

威胁等级：

若客户端退出登录时不会和服务器进行Logout的相关通信则为中风险，否则无风险。

2.4.10 密码修改验证

测试客户端在修改密码时是否验证旧密码正确性。

测试方法：

人工测试。

威胁等级：

当进行密码修改时是否要求输入原密码已验证其正确性，若需要输入则无风险；如不需输入原密码则中风险。

2.4.11 Activity 界面劫持

检查是否存在 activity 劫持风险，确认客户端是否能够发现并提示用户存在劫持。

测试方法：

使用 activity 界面劫持工具，在工具中指定要劫持的应用进程名称。进程名的获取可参考 5.13.2 进程查看和监视 ps/top。如图所示，从列表中选择被测试的应用，点击 OK。打开应用，

测试工具会尝试用自己的窗口覆盖被测的应用。当测试工具试图显示自己的窗口时，安全的客户端应该弹出警告提示。



威胁等级：

若客户端无法抵抗 Activity 界面劫持攻击时为中风险；若可以抵抗攻击则无风险。

2.5 手势密码安全性

2.5.1 手势密码复杂度

测试客户端手势密码复杂度，观察是否有点位数量判断逻辑。

测试方法：

1. 进入客户端设置手势密码的页面进行手势密码设置。
2. 进行手势密码设置，观察客户端手势密码设置逻辑是否存在最少点位的判断。
3. 反编译 APK 为 jar 包，通过 jd-gui 观察对应代码逻辑是否有相应的判断和限制条件。（一般设置手势密码若输入点数过少时会有相应的文字提示，通过此文字提示可以快速定位到代码位置）

威胁等级：

当用户设置或修改手势密码时服务器会对手势密码安全性（使用点数）进行判断时无风险，否则低风险。

2.5.2 手势密码修改和取消

检测客户端在取消手势密码时是否会验证之前设置的手势密码，检测是否存在其他导致手势密码取消的逻辑问题。

测试方法：

1. 进入客户端设置手势密码的位置，一般在个人设置或安全中心等地方。
2. 进行手势密码修改或取消操作，观察进行此类操作时是否需要输入之前的手势密码或普通密码。
3. 观察在忘记手势密码等其他客户端业务逻辑中是否存在无需原始手势或普通密码即可修改或取消手势密码的情况。
4. 多次尝试客户端各类业务，观察是否存在客户端逻辑缺陷使得客户端可以跳转回之前业务流程所对应页面。若存在此类逻辑（例如手势密码设置），观察能否修改或取消手势密码。
5. 反编译 APK 为 jar 包，通过 jd-gui 观察对应代码逻辑，寻找客户端对于手势密码的修改和删除是否存在相应的安全策略。

威胁等级：

当取消或修改手势密码时，如果不会验证之前的手势密码则为中风险；若存在验证则无风险。

2.5.3 手势密码本地信息保存

检测在输入手势密码以后客户端是否会在本地记录一些相关信息，例如明文或加密过的手势密码。

测试方法：

1. 首先通过正常的操作流程设置一个手势密码并完整一次完整的登陆过程。

2. 寻找/data/data 的私有目录下是否存在手势密码对应敏感文件，若进行了相关的信息保存，基本在此目录下。（关键词为 gesture，key 等）
3. 若找到对应的文件，观察其存储方式，为明文还是二进制形式存储，若为二进制形式，观察其具体位数是否对应进行 MD5（二进制 128 位，十六进制 32 位或 16 位）、SHA-1（二进制 160 位，十六进制 40 位）等散列后的位数。如果位数对应，即可在反编译的 jar 包中搜索对应的关键字以迅速对应代码。
4. 通过代码定位确认其是否进行了除单项哈希散列之外的加密算法，若客户端未将手势密码进行加密或变形直接进行散列处理可认为其不安全，一是因为现阶段 MD5、SHA-1 等常用的哈希算法已被发现碰撞漏洞，二是网络中存在 www.cmd5.com 等散列值查询网站可以通过大数据查询的方式获取散列前的明文手势密码。

安全建议：

建议不在客户端保存任何与手势密码相关的敏感信息（最好也不要加密存储）。建议在保证通信安全的情况每次由服务器对手势密码正确性进行验证。

威胁等级：

当本地保存了明文存储（数组形式）的手势密码时为高风险；当本地保存了只进行单项哈希散列的手势密码时为中风险。

2.5.4 手势密码锁定策略

测试客户端是否存在手势密码多次输入错误被锁定的安全策略。防止木马使用穷举法暴力破解用户密码。因为手势密码的存储容量非常小，一共只有 $9! = 362880$ 种不同手势，若手势密码不存在锁定策略，木马可以轻易跑出手势密码结果。

手势密码在输入时通常以 `a[2][2]` 这种 3×3 的二维数组方式保存，在进行客户端同服务器的数据交互时通常将此二维数组中数字转化为类似手机数字键盘的 `b[8]` 这种一维形式，之后进行一系列的处理进行发送。

测试方法：

1. 首先通过正常的操作流程设置一个手势密码。

2. 输入不同于步骤 1 中的手势密码，观察客户端的登陆状态及相应提示。若连续输入多次手势密码错误，观察当用户处于登陆状态时是否退出当前的登陆状态并关闭客户端；当客户未处于登录状态时是否关闭客户端并进行一定时间的输入锁定。
3. 反编译 APK 为 jar 包，通过 jd-gui 观察对应代码逻辑，寻找客户端是否针对输入次数及锁定时间有相应的逻辑处理。

威胁等级：

当服务器不会验证手势密码输入错误次数时为中风险，会进行验证时无风险。

2.5.5 手势密码抗攻击测试

验证是否可以通过插件绕过手势密码的验证页面。

测试方法：

1. 下载并安装 Xposed 框架及 SwipeBack 插件。
2. 启动客户端并进入手势密码输入页。
3. 启动 SwipeBack 插件，观察是否可以通过滑动关闭手势密码输入页的方式进入登陆后的页面。

威胁等级：

若客户端采用附着的方式将手势密码放置于登陆后的界面上时，如果无法抵抗 SwipeBack 插件的滑动攻击则高风险，如果可以抵抗则无风险。

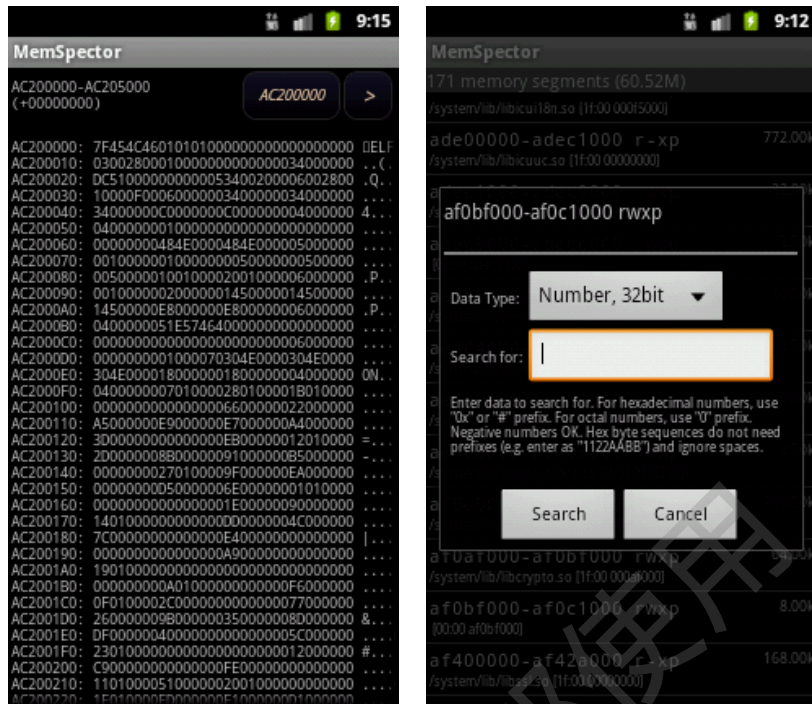
2.6 进程保护

2.6.1 内存访问和修改

通过对客户端内存的访问，木马将有可能得到保存在内存中的敏感信息（如登录密码，帐号等）。测试客户端内存中是否存在的敏感信息（卡号、明文密码等等）。

测试方法：

需要 root 权限，可以使用 MemSpector 查看、搜索和修改客户端内存数据，如图所示。用户名、密码等数据通常会在/dev/ashmem/dalvik-heap 内存段。（目前大多数工具都是通过 ptrace 接口修改客户端内存，可以使用 ptrace 机制本身防护。）



威胁等级:

当进行敏感操作后在内存中可以搜索到用户输入的敏感信息时为高风险，否则无风险。

2.6.2 *动态注入

通过注入动态链接库，hook 客户端某些关键函数，从而获取敏感信息或者改变程序执行。

测试方法:

检测 LD_PRELOAD 环境变量。使用 LD_PRELOAD 环境变量，可以让进程预先加载任意 so，劫持函数。如图是劫持 ls 命令 __libc_init() 函数的效果。（可参考此[链接](#)）

```
# LD_PRELOAD=/data/data/debug/libhookm.so ls
LD_PRELOAD=/data/data/debug/libhookm.so ls
hooked __libc_init!
libhookm.so
```

或者使用工具动态注入应用进程内存。参考 <https://github.com/crmulliner/ddi>。

或者使用 hook 框架来进行测试。对于 Android 上比较完善的 hook 框架可参考 5.9Android Hook 框架。

威胁等级:

当客户端存在动态注入隐患时高风险，否则无风险。

2.7 通信安全

2.7.1 通信加密

客户端和服务端通信是否强制采用 https 加密。

测试方法：

参考 5.13.1.2 嗅探流量 tcpdump，使用 tcpdump 嗅探客户端提交的数据，将其保存为 pcap 文件。使用 Wireshark 打开 pcap 文件，检查交互数据是否是 https。

将客户端链接到的地址改为 http（将所有 URL 开头的 https 改为 http），查看客户端是否会提示连接错误。

威胁等级：

当客户端和服务器的通信不经过 SSL 加密（或没有参考 TLS 协议，RFC4346 等实现加密信道）时为高风险；当自实现通信算法存在漏洞可被解析或绕过时为高风险；使用低版本 SSL 协议（SSLV2，SSLV3 均存在漏洞，至少使用 TLSV1.1 以上算法）时为高风险；以上问题均不存在时无风险。

2.7.2 证书有效性

1. 客户端程序和服务器端 SSL 通信是否严格检查服务器端证书有效性。避免手机银行用户受到 SSL 中间人攻击后，密码等敏感信息被嗅探到。

测试方法：

通过 wifi 将手机和测试 PC 连接到同一子网。参考 5.14 android 代理配置在手机上配置好代理，代理 IP 为测试 PC IP 地址，端口为代理的监听端口，如图所示。此时，客户端通信将会转发给测试 PC 上的 fiddler 代理。



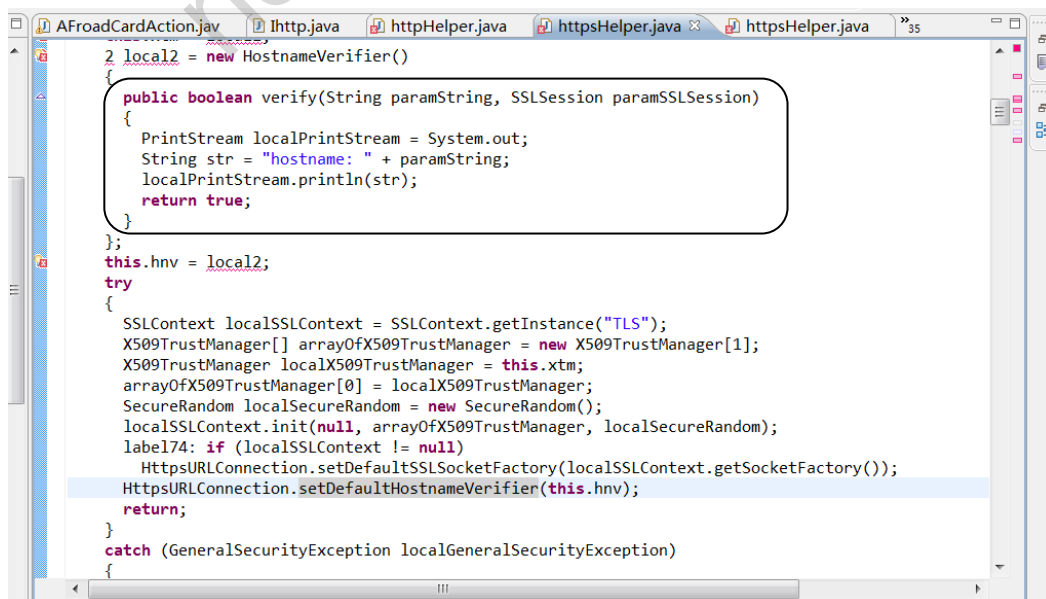
然后使用客户端访问服务端，查看客户端是否会提示证书问题。

2. *测试客户端程序是否严格检查服务器端证书信息，避免手机银行用户访问钓鱼网站后，泄露密码等敏感信息。

测试方法：

通过修改 DNS，将客户端链接到的主页地址改为 <https://mail.qq.com/>，然后使用客户端访问服务端，查看客户端是否会提示连接错误。此项测试主要针对客户端是否对 SSL 证书中的域名进行确认。

* 查阅代码中是否有 SSL 验证。下图是 Java 中进行服务端 SSL 证书验证的一种方式。关键函数为：`java.net.ssl.HttpURLConnection.setDefaultHostnameVerifier()`，通过此函数查找 `HostnameVerifier` 的 `verify` 函数。如 `verify()` 函数总返回 `true`，则客户端对服务端 SSL 证书无验证。（可能还有其他 SSL 实现，需要验证）



详情请参考 [Android SDK](#)。

（在代码中添加证书的代码如下，证书保存在资源 R.raw.mystore 中。

```
KeyStore trusted = KeyStore.getInstance("BKS");
InputStream in = _resources.openRawResource(R.raw.mystore);
try {
    trusted.load(in, "pwd".toCharArray());
} finally {
    in.close();
}
```

可参考此[链接](#)。）

3. SSL 协议安全性。检测客户端使用的 SSL 版本号是否不小于 3.0（或 TLS v1），加密算法是否安全。（安全规范要求）

测试方法：

使用 openssl，指定域名和端口，可以看到 SSL 连接的类型和版本。如下图所示，使用了 TLSv1，加密算法为 AES 256 位密钥。（也可以使用[这个网站](#)检测）（RC4，DES 等算法被认为是不安全的）

```
asky@debian:~$ openssl s_client -host mail.yahoo.com -port 443
CONNECTED(00000003)
depth=2 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert High Assurance EV Root CA
verify error:num=20:unable to get local issuer certificate
verify return:0
---
New, TLSv1/SSLv3, Cipher is AES256-SHA
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: zlib compression
Expansion: zlib compression
SSL-Session:
    Protocol : TLSv1
    Cipher   : AES256-SHA
    Session-ID: A6BE8591056B3953C172DEF70791DD1C268B3893B988005132811468EFC7867
    Session-ID-ctx:
    Master-Key: 65C8D1A61C235CF999ABBBF09CA023893A4898D11C21A04A6880A49A922D7748
    BC36BD5057E702BC1A67C3D94C07E2C4
```

威胁等级：

当客户端和服务端互相不验证证书时高风险，当只有客户端验证服务器证书时为中风险；当服务器不通过白名单的方式验证客户端时为中风险；当客户端和服务端进行双向认证，并且服务器通过白名单方式验证客户端证书时无风险。

2.7.3 关键数据加密和校验

1. 测试客户端程序提交数据给服务端时，密码、收款人信息等关键字段是否进行了加密，防止恶意用户嗅探到用户数据包中的密码等敏感信息。

测试方法：

参考 5.14 android 代理配置在手机上配置好代理，观察客户端和服务端的交互数据。检查关键字段是否加密。

如果客户端对根证书进行了严格检测，导致代理无法使用。则可以参考 5.15 手机根证书安装将代理的根证书安装到设备上，使根证书可信。或是参考 5.7 修改已安装 apk 和 5.15 手机根证书安装，替换客户端 apk 中的根证书文件。

如果上述方法均失效，则只能参考 5.2.1 反编译为 java 代码，将客户端逆向后，通过阅读 java 代码的方式寻找客户端程序向服务端提交数据的代码，检查是否存在加密的代码。

2. 测试客户端程序提交数据给服务端时，是否对提交数据进行签名，防止提交的数据被木马恶意篡改。

测试方法：

参考 5.14 android 代理配置一节配置代理。使用代理观察交互数据，确认是否包含签名字段。尝试在代理中篡改客户端提交的数据，检查服务端是否能检测到篡改。

威胁等级：

当账号，密码，卡号等数据明文传输，未进行二次加密时为高风险；当密码只进行了单项散列而未经过加密时为高风险；当返回数据中包含更新的 URL 且数据不加密时为高风险；当校验字段删除后服务器仍会处理所发送的数据包时为高风险；当校验字段的散列中不包含随机因子时为高风险。以上问题均不存在时无风险。

2.7.4 *访问控制

测试客户端访问的 URL 是否仅能由手机客户端访问。是否可以绕过登录限制直接访问登录后才能访问的页面，对需要二次验证的页面（如私密问题验证），能否绕过验证。

测试方法：

人工测试。在 PC 机的浏览器里输入 URL，尝试访问手机银行页面。

威胁等级：

当 PC 端也可访问手机页面时低风险，当可以绕过登陆限制访问登陆后才能访问的页面时中风险。

2.7.5 客户端更新安全性

测试客户端自动更新机制是否安全。如果客户端更新没有使用官方应用商店的更新方式，就可能导致用户下载并安装恶意应用，从而引入安全风险。

测试方法：

使用代理抓取检测更新的数据包，尝试将服务器返回的更新 url 替换为恶意链接。看客户端是否会直接打开此链接并下载应用。

在应用下载完毕后，测试能否替换下载的 apk 文件，测试客户端是否会安装替换后的应用。

威胁等级：

当客户端返回明文 URL 地址并可以通过篡改的方式控制用户下载恶意 APK 包进行安装，则高风险；若返回数据包经过二次加密则无风险。

2.7.6 短信重放攻击

检测应用中是否存在数据包重放攻击的安全问题。是否会对客户端用户造成短信轰炸的困扰。

测试方法：

尝试重放短信验证码数据包是否可以进行了短信轰炸攻击。

威胁等级：

当存在短信轰炸的情况时为中风险，若短信网关会检测短时间内发送给某一手机号的短信数量则无风险。

2.8 业务功能测试

对于可以通过代理的方式对交互数据进行分析的客户端，可以对涉及到敏感信息操作的具体业务功能进行测试。

测试方法：

根据客户端的业务流程，使用代理截获客户端每个功能进行的通信数据，测试对数据的篡改或重放所导致的问题。

具体测试内容包括但不限于：篡改造成的越权操作（如跨帐户查询），交易篡改（如修改金额，转帐金额为负值），特殊数据提交（如各种注入问题），重放导致的多次交易，以及用户枚举和暴力密码破解，等等。

（有条件的话可以使用扫描器工具来进行测试）

1. 重放
2. 篡改
3. 越权

nsfocus内部使用

三. 测试项目风险定级

此表格列举了第二章中各个测试项在存在安全风险时，对其风险等级定级的参考标准。

测试分类	测试项目	风险等级
客户端程序安全	* 安装包签名	--
	客户端程序保护	低
	应用完整性检测	高
	组件安全	高：可获得密码等敏感信息，可篡改敏感数据； 中或低：仅不恰当导出，不能获取敏感信息
	webview 组件安全	高
敏感信息安全	数据文件	高：保存明文或简单编码的密码、cookies、包含敏感信息的网页缓存、用户其他敏感信息等； 中：保存登录用户名、其他非敏感信息。
	logcat 日志	
密码软键盘安全性	键盘劫持	高：可以劫持（使用系统软键盘）
	随机布局软键盘	高：使用系统软键盘；中：使用自带软键盘，但键盘布局不随机。
	屏幕录像	高：输入时有视觉回显
进程保护	内存访问和修改	高：内存中可以搜索到明文密码
	* 动态注入	高
手势密码安全性	手势密码复杂度	中：没有复杂度检测
	手势密码修改和取消	高
	手势密码本地信息保存	高
	手势密码锁定策略	高
	手势密码抗攻击测试	高
安全策略	密码复杂度检测	中：没有复杂度检测；低：有检测，不全面
	帐号登录限制	高
	帐户锁定策略	高
	* 私密问题验证	中
	会话安全设置	高
	界面切换保护	中

	UI 信息泄露	中
	验证码安全性	中
	安全退出	高
	密码修改验证	高
	activity 界面劫持	中
通信安全	通信加密	高：未使用 HTTPS 且没有加密
	证书有效性检测	高
	关键数据加密和校验	高
	*访问控制	中
	客户端更新安全性	高
	短信重放攻击	中
业务功能测试	越权查询 / 修改账户信息	高
	其他	

四. 合规性参考

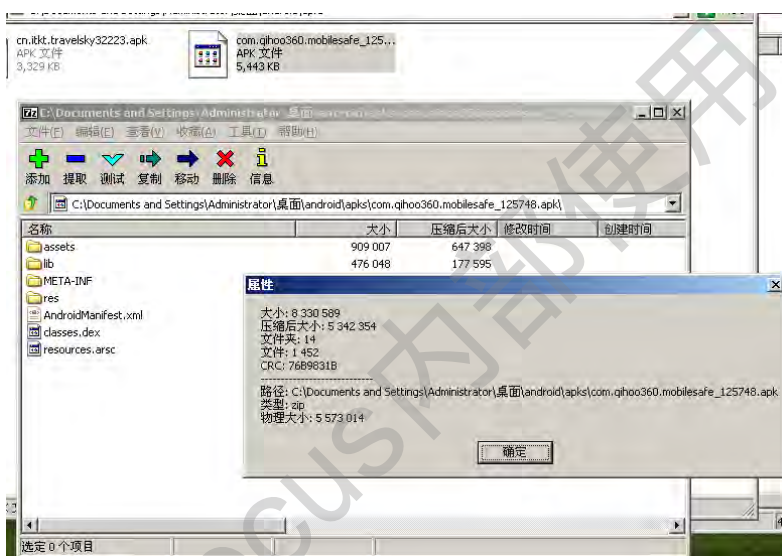
下表是《网上银行系统信息安全通用规范》（2012）中关于客户端安全方面的要求。

分类	要求项	对应编号
客户端程序（基本要求）	a) 金融机构应采取有效技术措施保证客户端处理的敏感信息、客户端与服务器交互的重要信息的机密性和完整性；应保证所提供的客户端程序的真实性和完整性，以及敏感程序逻辑的机密性	2.1.2, 2.1.3, 2.6.3
	b) 客户端程序上线前应进行严格的代码安全测试，如果客户端程序是外包给第三方机构开发的，金融机构应要求开发商进行代码安全测试。金融机构应建立定期对客户端程序进行安全检测的机制	--
	c) 客户端程序应通过指定的第三方中立测试机构的安全检测，每年至少开展一次。	--
	d) 应对客户端程序进行签名，标识客户端程序的来源和发布者，保证客户所下载的客户端程序来源于所信任的机构。	2.1.1
	e) 客户端程序在启动和更新时应进行真实性和完整性校验，防范客户端程序被篡改或替换。	2.1.3, 2.6.5
	f) 客户端程序的临时文件中不应出现敏感信息，临时文件包括但不限于 Cookies。客户端程序应禁止在身份认证结束后存储敏感信息，防止敏感信息的泄露。	2.2
	g) 客户端程序应提供客户输入敏感信息的即时加密功能，例如采用密码保护控件。	2.3
	h) 客户端程序应具有抗逆向分析、抗反汇编等安全性防护措施，防范攻击者对客户端程序的调试、分析和篡改。	2.1.2
	i) 客户端程序应防范恶意程序获取或篡改敏感信息，例如使用浏览器接口保护控件进行防范。	（针对 PC 客户端）
	j) 客户端程序应防范键盘窃听敏感信息，例如防范采用挂钩 Windows 键盘消息等方式进行键盘窃听，并应具有对通过挂钩窃听键盘信息进行预警的功能。	（针对 PC 客户端）
	k) 客户端程序应提供敏感信息机密性、完整性保护功能，例如采取随机布放按键位置、防范键盘窃听技术、计算 MAC 校验码等措施。	2.3
客户端程序（增强要求）	a) 客户端程序应保护在客户端启动的用于访问网上银行的进程，防止非法程序获取该进程的访问权限。	2.5.1
	b) 客户端程序应采用反屏幕录像技术，防范非法程序获取敏感信息。	2.3.3
	c) 客户端程序应采取代码混淆等技术手段，防范攻击者对客户端程序的调试、分析和篡改。	2.1.2
	d) 客户端程序开发设计过程中应注意规避各终端平台存在的安全漏洞，例如，按键输入记录、自动拷屏机制、文档显示缓存等。	2.2, 2.3

五. android 应用分析

5.1 apk 解包

android 的 apk 文件是使用 zip 算法的压缩包, 可以使用任何支持 zip 格式的工具(winRAR, 7zip 等等)解压缩。



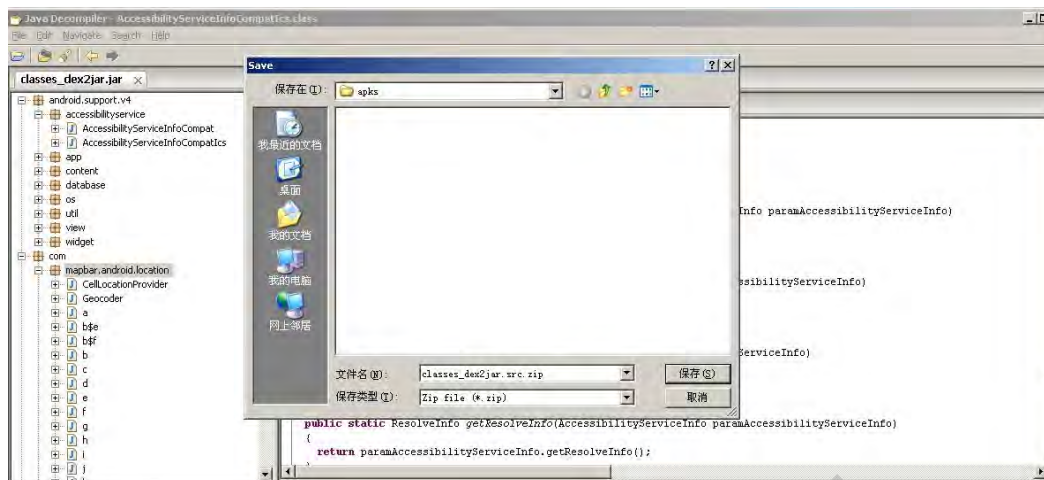
5.2 逆向 classes.dex

5.2.1 反编译为 java 代码

1. 使用 dex2jar 工具, 以 classes.dex 为参数运行 dex2jar.bat。成功运行后, 在当前文件夹会生成 classes_dex2jar.jar 文件。

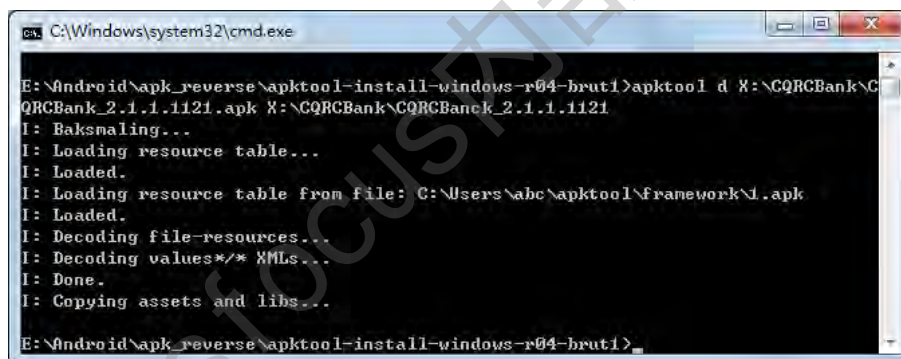


2. 使用 jd-gui 工具查看、搜索并保存 jar 中的 java 代码。



5.2.2 反编译为 smali 代码

使用 apktool 工具可以对 apk 进行解包。具体的解包命令格式为: `apktool d[ecode] [OPTS] <file.apk> [<dir>]`。例如, 对 CQRCBank_2.1.1.1121.apk 进行解包的命令如下:



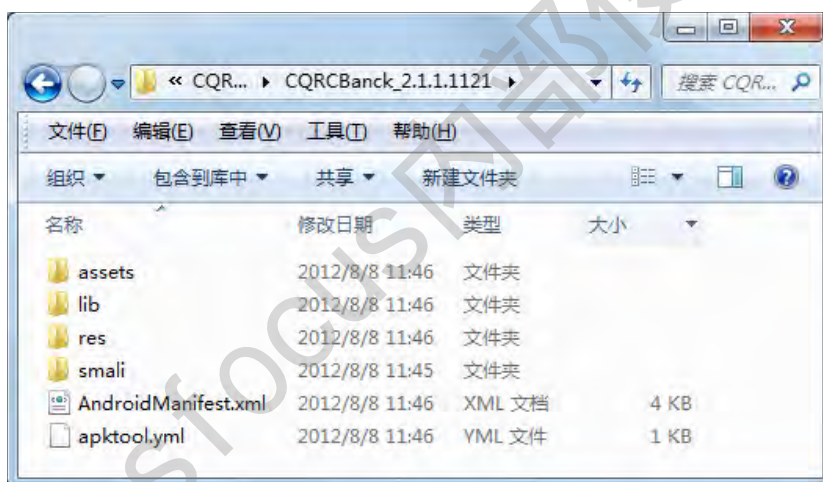
1. 如果只需要修改 smali 代码, 不涉及资源文件的修改, 可以在解包时加入 -r 选项 (也可以直接使用 baksmali 将 dex 反编译为 smali 代码, 见 5.3), 不解码 apk 中的资源。在打包时可以避免资源方面的问题 (如 aapt 报的各种错误)。
2. 如果只需要反编译资源文件, 可以在解包时加入 -s 选项, 不对 classes.dex 进行反编译。
3. 如果在 5.6.1 使用 apktool 打包 smali 代码中出现资源相关的错误, 可能是需要较新的 framework 文件。可参考[此处](#), 添加 framework 文件。例如, 添加 Android 4.4.2 SDK 中的 framework 文件, 命令如下:

```
C:\Documents and Settings\Administrator>"C:\Program Files\apktool1.5.2\apktool.bat" if "C:\Program Files\Android\android-sdk\platforms\android-19\android.jar" tag0
I: Framework installed to: C:\Documents and Settings\Administrator\apktool\framework\1-tag0.apk
```

4. 解包时指定相应的 framework（上面命令中的 tag0 是对添加的 framework 的标记，用于标识不同的 framework），如图所示：

```
C:\Documents and Settings\Administrator>"C:\Program Files\apktool1.5.2\apktool.bat" d -t tag0 C:\root\android\CQRCB_MobileBank_android_1.1.4.apk
I: Baksmaling...
I: Loading resource table...
I: Loaded.
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: C:\Documents and Settings\Administrator\apktool\framework\1-tag0.apk
I: Loaded.
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Done.
I: Copying assets and libs...
```

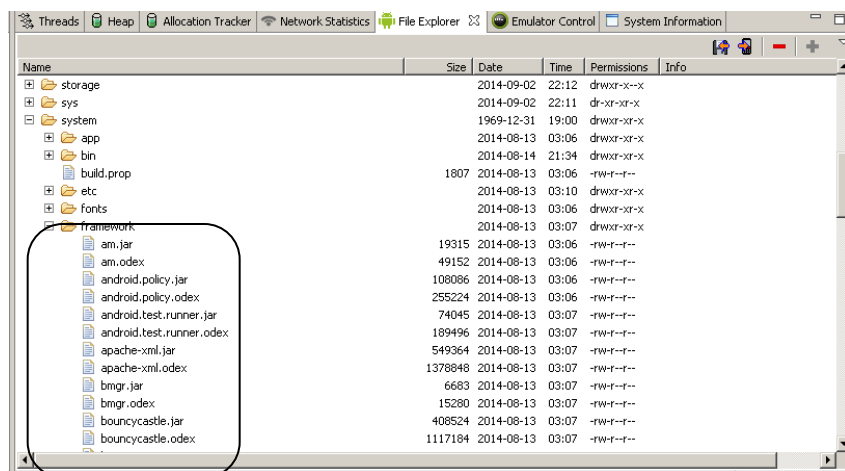
解包完成后，会将结果生成在指定的输出路径中，其中，smali 文件夹下就是最终生成的 Dalvik VM 汇编代码，AndroidManifest.xml 文件以及 res 目录下的资源文件也已被解码。如图：



5.3 处理 odex 文件

odex 是 android 系统中对 dex 文件优化后生成的文件。关于 odex 的生成可以参考 5.7 修改已安装 apk 第 5. 步。如果要使用上述反编译方法，需要先将 odex 转换成 dex。

1. 下载 smali 工具（<https://code.google.com/p/smali/>）。
2. 将虚拟机中/system/framework/中的 jar 文件复制出来，放到一个文件夹中。所需的虚拟机版本可参考 odex 生成的环境（如 odex 是在 android 4.4 中生成的，就复制 4.4 虚拟机，如果在真机中生成，则可以复制真机的）。



3. 运行 baksmali.jar, 将 odex 解析为 smali 代码。-x 选项表示输入是 odex 文件, -d 选项指定上个步骤中复制出来的 jar 文件路径, 如下图所示。当命令成功执行后, 在当前目录会创建一个 out 文件夹, 里面就是 smali 代码。

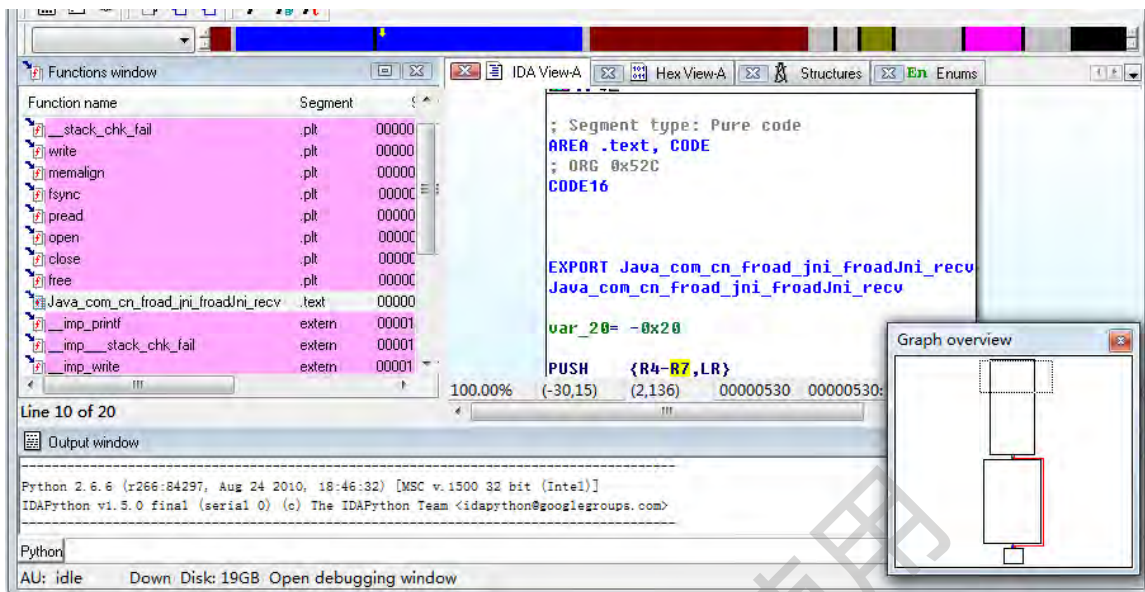
```
C:\Documents and Settings\Administrator>java -jar Z:\ShareFolder\bin\smali-bins\
baksmali-2.0.3.jar -x -d Z:\ShareFolder\Code\AndroidFramework "C:\Documents an
d Settings\Administrator\桌面\dexdump.odex"
```

4. 运行 smali.jar, 可生成 dex。如下图所示:

```
C:\Documents and Settings\Administrator>java -jar Z:\ShareFolder\bin\smali-bins\
smali-2.0.3.jar -x "C:\Documents and Settings\Administrator\out" -o dd.dex
```

5.4 反编译 so 库

apk 解压缩后, 将 lib\armeabi\目录下的 so 文件直接拖入 IDA 中, 可以对 so 文件进行静态分析。可以看到 so 文件中包含的函数, ARM 汇编代码, 导入导出函数等信息。



5.5 处理 xml

apk 中的 xml 大部分是经过编译的，无法直接查看和修改。

1. 如果需要查看 xml 文件，可以参考 5.2.2 反编译为 smali 代码部分，使用 apktool 将整个 apk 解包。或者是使用 [AXMLPrinter](#) 或 [APKParser](#) 工具对要查看的 xml 进行解码。如图：

```
C:\Documents and Settings\Administrator>java -jar Z:\ShareFolder\bin\AXMLPrinter
2.jar
Usage: AXMLPrinter <binary xml file>

C:\Documents and Settings\Administrator>java -jar Z:\ShareFolder\bin\AXMLPrinter
2.jar C:\root\android\t2\AndroidManifest.xml > C:\root\android\t2\AndroidManife
st-txt.xml
```

```
C:\Documents and Settings\Administrator>java -jar Z:\ShareFolder\bin\APKParser.j
ar
Usage: AXMLPrinter <APK FILE PATH>

C:\Documents and Settings\Administrator>java -jar Z:\ShareFolder\bin\APKParser.j
ar C:\root\android\t2\cn.cj.pe-1.apk > C:\root\android\t2\AndroidManifest-txt1
.xml
```

2. 如果需要将修改后的 xml 重新打包到 apk 中，则可以参考 5.6.1 节，使用 apktool 打包。目前还没有发现可以单独编译一个 xml 文件的方法。对于已经解包的 apk，也可以直接使用 android SDK 中的 aapt 直接编译资源文件（包括 xml）。命令格式如下，apk-src 是 apk 的解包目录，output.zip 是输出的 zip 文件（编译好的资源文件都会打包到里面），-l 选项指定相应版本的 android.jar。

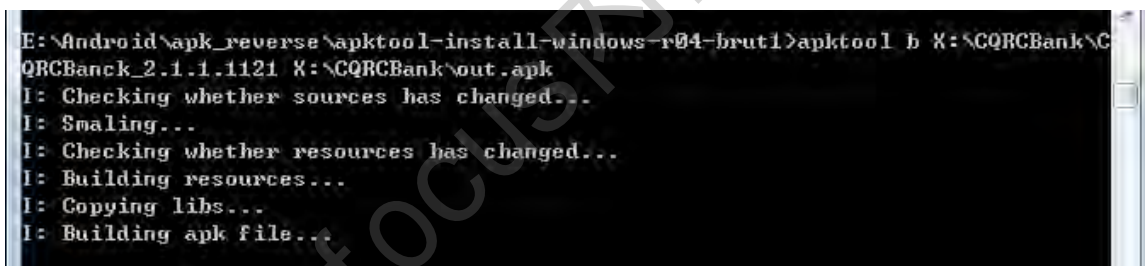

```
"ANDROID-SDK\build-tools\20.0.0\aapt.exe" package -f -M [apk-src\AndroidManifest.xml]  
-I "ANDROID-SDK\platforms\android-19\android.jar" -S [apk-src\res] -F [output.zip]
```

注：上述 aapt 和 android.jar 的路径为安装 android SDK build-tools rev.20, android 4.4.2 SDK platform 后才存在。如果没有安装上述版本的组件，可将路径改为其他版本相应的路径。（通常较新版本的 SDK 出错的可能性会小一些。）

5.6 打包 apk

5.6.1 使用 apktool 打包 smali 代码

使用 apktool 打包 apk 文件的命令格式为：b[uild] [OPTS] [<app_path>] [<out_file>]。其中<app_path>是apk 已经被解包后的目录，打包成功后会将生成的 apk 文件输出到<out_file>。其过程如图：



```
E:\Android\apk_reverse\apktool-install-windows-r04-brut1>apktool b X:\CQRCBank\CQRCBank_2.1.1.1121 X:\CQRCBank\out.apk  
I: Checking whether sources has changed...  
I: Smaling...  
I: Checking whether resources has changed...  
I: Building resources...  
I: Copying libs...  
I: Building apk file...
```

如果在打包过程中出错，可以参考命令给出的错误提示进行解决。如一个不是 png 格式的图形文件扩展名是.png 时，apktool 打包时就会报错（NOT A PNG file）。可以根据图片格式修改图片的扩展名，或者将图片转换为 png 格式。

如果有其他资源类错误发生，可参考 5.2.2 的第 3 步处理。

5.6.2 签名和优化

Android 系统要求每一个 apk 安装包都必须经过数字证书签名。与 Windows 程序的数字证书签名不同，apk 安装包的数字证书不需要权威的数字证书签名机构认证。数字证书签名，一可以判断 apk 安装包在签名后是否被篡改过，二可以识别 apk 安装包的作者。程序升级时，只有当新版程序和旧版程序的数字证书相同时，Android 系统才会认为这两个程序是同一个程

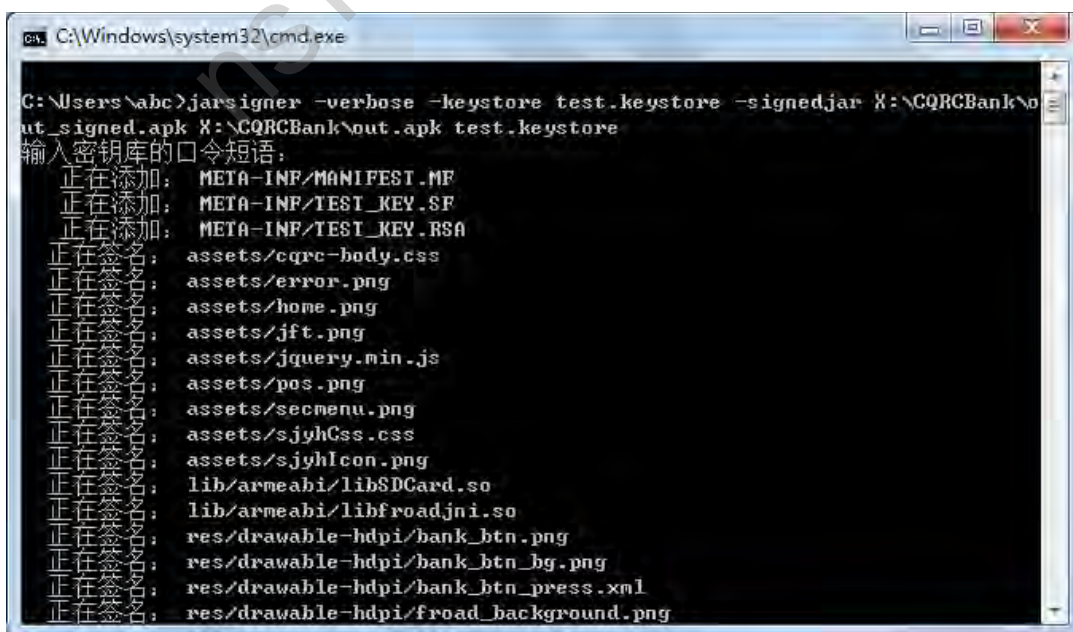
序的不同版本，并使用新版覆盖旧版程序；否则认为二者是不同的程序，会要求新程序更改包名。使用 apktool 重新打包的 apk 文件并不能直接安装，因为在 apk 中缺少签名。

1. 首先用 JDK 中的 keytool 生成一个签名文件。其中，-alias 是指定签名的别名，-keyalg 选项指定了签名加密的算法（对于 Android APK 文件必须指定为 RSA 算法），-validity 选项指定签名的有效天数。



```
C:\Users\abc>keytool -genkey -alias test.keystore -keyalg RSA -validity 40000 -k test.keystore test.keystore
输入 keystore 密码:
再次输入新密码:
您的名字与姓氏是什么?
[Unknown]: test
您的组织单位名称是什么?
[Unknown]: test
您的组织名称是什么?
[Unknown]: test
您所在的城市或区域名称是什么?
[Unknown]: test
您所在的州或省份名称是什么?
[Unknown]: test
该单位的两字母国家代码是什么?
[Unknown]: test
CN=test, OU=test, O=test, L=test, ST=test, C=test 正确吗?
[否]: y
输入<test.keystore>的主密码
(如果和 keystore 密码相同, 按回车):
再次输入新密码:
```

2. 然后使用 JDK 中的 jarsigner 工具将生成的签名签署在 apk 文件中。其中，-keystore 选项指定密钥库的位置，-signedjar 选项指定签名后产生的文件，另外两个参数为要签名的文件 out.apk 和密钥库 test.keystore。对于 java 版本大于等于 1.7.0 的，还需要添加选项“-digestalg SHA1 -sigalg MD5withRSA”。



```
C:\Windows\system32\cmd.exe
C:\Users\abc>jarsigner -verbose -keystore test.keystore -signedjar X:\CQRCBank\out_signed.apk X:\CQRCBank\out.apk test.keystore
输入密钥库的口令短语:
正在添加: META-INF/MANIFEST.MF
正在添加: META-INF/TEST_KEY.SF
正在添加: META-INF/TEST_KEY.RSA
正在签名: assets/cqrc-body.css
正在签名: assets/error.png
正在签名: assets/home.png
正在签名: assets/jft.png
正在签名: assets/jquery.min.js
正在签名: assets/pos.png
正在签名: assets/secmenu.png
正在签名: assets/sjyhCss.css
正在签名: assets/sjyhIcon.png
正在签名: lib/armabi/libSDCard.so
正在签名: lib/armabi/libfroadjni.so
正在签名: res/drawable-hdpi/bank_btn.png
正在签名: res/drawable-hdpi/bank_btn_bg.png
正在签名: res/drawable-hdpi/bank_btn_press.xml
正在签名: res/drawable-hdpi/froad_background.png
```

生成的 out_signed.apk 文件是已被签名可以直接安装的 apk 文件。

3. 签名后的 apk 文件可以使用 Android SDK 附带的 zipalign 工具进一步做对齐优化。详细命令为: `zipalign -v 4 <out_signed.apk> <final.apk>`

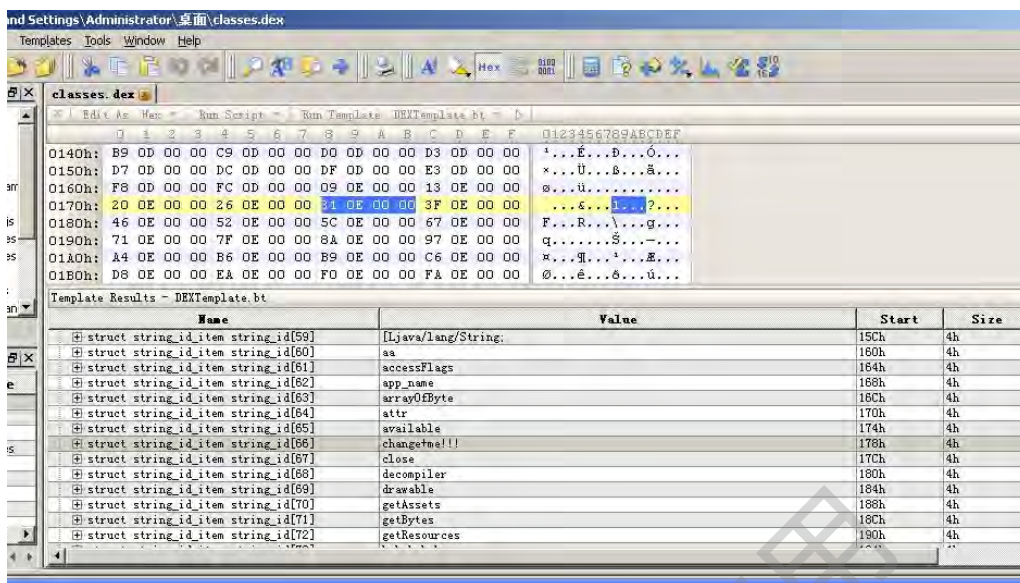
```
C:\Windows\system32\cmd.exe

C:\Users\abc>zipalign -v 4 X:\CQRCBank\out_signed.apk X:\CQRCBank\final.apk
Verifying alignment of X:\CQRCBank\final.apk (4)...
 50 META-INF/MANIFEST.MF (OK - compressed)
2981 META-INF/TEST_KEY.SF (OK - compressed)
6377 META-INF/TEST_KEY.RSA (OK - compressed)
7042 assets/cqrc-body.css (OK - compressed)
79536 assets/error.png (OK)
98636 assets/home.png (OK)
264812 assets/jft.png (OK)
335083 assets/jquery.min.js (OK - compressed)
360396 assets/pos.png (OK)
415920 assets/secmenu.png (OK)
594200 assets/sjyhCss.css (OK - compressed)
662596 assets/sjyhIcon.png (OK)
1500980 lib/armeabi/libSDCard.so (OK - compressed)
1503792 lib/armeabi/libfroadjni.so (OK - compressed)
1505240 res/drawable-hdpi/bank_btn.png (OK)
1511844 res/drawable-hdpi/bank_btn_bg.png (OK)
1520028 res/drawable-hdpi/bank_btn_press.xml (OK - compressed)
1520460 res/drawable-hdpi/froad_background.png (OK)
1524980 res/drawable-hdpi/froadshop_btn.png (OK)
1531812 res/drawable-hdpi/froadshop_btn_bg.png (OK)
1539162 res/drawable-hdpi/froadshop_btn_press.xml (OK - compressed)
1539584 res/drawable-hdpi/icon.png (OK)
```

5.7 修改已安装 apk

apk 安装到手机后, 会在如下目录安装文件: /data/dalvik-cache: 优化后的 dex 文件; /data/app: apk 文件。修改文件后注意被修改文件的文件权限要和修改前保持一致。

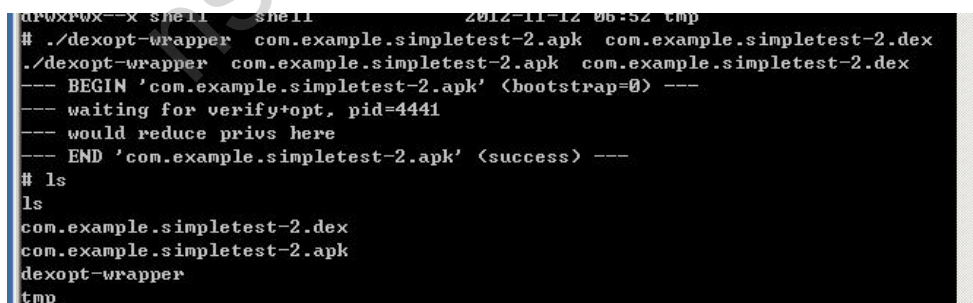
1. 修改 apk 中 assets 目录中的文件, 使用 zip 压缩工具打开 apk, 然后直接修改并覆盖 /data/app 中的原有 apk 即可。
2. 修改 apk 中的 dex 文件, 需要同时更新/data/dalvik-cache 和/data/app 目录中的文件。
3. 首先导出/data/app 目录中的 apk 文件, 将 classes.dex 解压出来, 进行修改。如图是使用 010 Editor 修改 dex 中的字符串资源 (注意: dex 里字符串的顺序很重要, 如果修改后字符串顺序有变化, 第 5 步优化的时候会报错。如原字符串是“change”, 下一个字符串是“close”, 则原字符串可以改为“cllgg”, 而不能改成“cyyggg”。猜测 dex 里的字符串可能是按照 ASCII 排序的, “cyyggg”在“close”的后面。))。



4. dex 修改完毕，使用工具更新 dex 的 checksum 和 SHA-1 签名，如图所示。然后覆盖 apk 里的原有 classes.dex 文件。



5. 将修改后的 apk 导入到手机中，使用 dexopt-wrapper 工具生成优化的 odex 文件，如图所示。（当/data/dalvik-cache 中的 dex 文件所有人可写时，此步可跳过，android 系统会自动更新 dex 文件）



6. 使用修改的 apk 覆盖/data/app 目录中的 apk，使用生成的 dex 文件覆盖/data/dalvik-cache 中的 dex 文件。

5.8 内存获取 classes.dex

针对某些加固过的应用，通过上述方法无法获得真正的 `classes.dex`，从而无法对 `apk` 应用进行静态分析。此时，可以通过应用运行时的内存数据还原 `classes.dex`。具体方法有如下几种。

5.8.1 内存转储

5.8.2 ZjDroid 工具

ZjDroid 是一个基于 Xpose framework 的逆向分析工具。可以通过此工具获得加固应用的真实 dex 文件。

1. 首先，安装 Xpose framework，确定 Xpose framework 正常工作（可参考 5.9 Android Hook 框架）。
2. 然后从官网（<https://github.com/BaiduSecurityLabs/ZjDroid>）下载代码，使用 Eclipse 编译（参考 5.10.2 Eclipse 导入代码步骤），安装。
3. 安装完重启 android 系统后，进入 adb shell。同时打开 Android debug monitor（5.10.3 Android debug monitor）或 Eclipse，以便随时可以查看 logcat 日志。
4. 在 adb shell 中输入命令获取 dex 的信息：`am broadcast -a com.zjdroid.invoke --ei target pid --es cmd '{"action":"dump_dexinfo"}'`，其中 `pid` 为目标应用的 PID。logcat 输出如图：

ation	Tag	Text
example.blanktest	zjdroid-shell-com.example.blanktest	the cmd = dump_dexinfo
example.blanktest	zjdroid-shell-com.example.blanktest	The DexFile Infomation ->
example.blanktest	zjdroid-shell-com.example.blanktest	filepath:/data/app/com.example.blanktest-1.apk mCookie:19427 0
example.blanktest	zjdroid-shell-com.example.blanktest	56816
example.blanktest	zjdroid-shell-com.example.blanktest	filepath:/data/data/com.example.blanktest/app_bangleplugin/ 0
example.blanktest	zjdroid-shell-com.example.blanktest	collector.dex mCookie:1933617568
example.blanktest	zjdroid-shell-com.example.blanktest	filepath:/data/data/com.example.blanktest/.cache/classes.jar 0
example.blanktest	zjdroid-shell-com.example.blanktest	mCookie:2033072416
example.blanktest	zjdroid-shell-com.example.blanktest	filepath:/data/data/com.example.blanktest/app_bangleplugin/ 0
example.blanktest	zjdroid-shell-com.example.blanktest	container.dex mCookie:2036652720
example.blanktest	zjdroid-shell-com.example.blanktest	End DexFile Infomation

5. 上面的 dex 文件列表中有多个文件（filepath），可以先查看对应文件的类信息，使用命令 `am broadcast -a com.zjdroid.invoke --ei target pid --es cmd '{"action":"dump_class","dexpath":"path"}'`，此处输入命令如图：

```

root@x86:/ # am broadcast -a com.zjdroid.invoke --ei target 3107 --es cmd '{"action":"dump_class","dexpath":"/data/data/com.example.blanktest/.cache/classes.jar"}'
Broadcasting: Intent { act=com.zjdroid.invoke (has extras) }
Broadcast completed: result=0
root@x86:/ #

```

6. 在 logcat 中看到输出，其中包含了我们想要的类。

LOG	TEXT
example.blanktest	zjdroid-shell-com.example.blanktest the cmd = dump class
example.blanktest	Start Loadable ClassName ->
example.blanktest	ClassName = com.example.blanktest.MainActivity
example.blanktest	ClassName = com.example.blanktest.BuildConfig
example.blanktest	ClassName = com.example.blanktest.R\$attr
example.blanktest	ClassName = com.example.blanktest.R\$drawable
example.blanktest	ClassName = com.example.blanktest.R\$layout
example.blanktest	ClassName = com.example.blanktest.R\$string
example.blanktest	ClassName = com.example.blanktest.R\$style
example.blanktest	ClassName = com.example.blanktest.R
example.blanktest	ClassName = com.example.blanktest.Testrec
example.blanktest	ClassName = neo.proxy.ContainerFactory
example.blanktest	ClassName = neo.proxy.DistributeReceiver
example.blanktest	ClassName = neo.proxy.FastService
example.blanktest	ClassName = neo.skeleton.base.Actions
example.blanktest	ClassName = neo.skeleton.base.Coder\$CoderException
example.blanktest	ClassName = neo.skeleton.base.Coder
example.blanktest	ClassName = neo.skeleton.base.Configurable
example.blanktest	ClassName = neo.skeleton.base.Configs
example.blanktest	ClassName = neo.skeleton.base.Plugable
example.blanktest	ClassName = neo.skeleton.base.Containable
example.blanktest	ClassName = neo.skeleton.base.NetStream\$NetStatus
example.blanktest	ClassName = neo.skeleton.base.NetStream
example.blanktest	ClassName = neo.skeleton.base.Plugin
example.blanktest	ClassName = neo.skeleton.base.SafeConfigs

7. 转储上述 dex 的数据，命令格式为: am broadcast -a com.zjdroid.invoke --ei target pid --es cmd '{"action":"dump_dexfile","dexpath":"path"}'，实际输入如图：

```

root@x86:/ # am broadcast -a com.zjdroid.invoke --ei target 3107 --es cmd '{"action":"dump_dexfile","dexpath":"/data/data/com.example.blanktest/.cache/classes.jar"}'
Broadcasting: Intent { act=com.zjdroid.invoke (has extras) }
Broadcast completed: result=0
root@x86:/ #

```

8. logcat 中日志如图，转储文件已保存在应用的私有目录的 files 子目录下。

LOG	TEXT
example.blanktest	zjdroid-shell-com.example.blanktest End Loadable ClassName
example.blanktest	zjdroid-shell-com.example.blanktest the cmd = dump dexfile
example.blanktest	zjdroid-shell-com.example.blanktest the dexfile data save to =/data/data/com.example.blanktest/files/dexdump.odex

9. 转储下来的文件是 odex 格式的，可参考 5.3 处理 odex 文件将其转换为 dex 格式，以便于分析。
10. am broadcast -a com.zjdroid.invoke --ei target pid --es cmd '{"action":"backsmali","dexpath":"*****"}'

5.9 Android Hook 框架

Android 下的 hook 框架通过修改 android 系统，挂钩底层函数，可实现很多个性化功能，类似于 iOS 下的 Mobile Substrate。目前 android 下常用的 hook 框架是 Xposed Framework

5.9.1 Xposed Framework

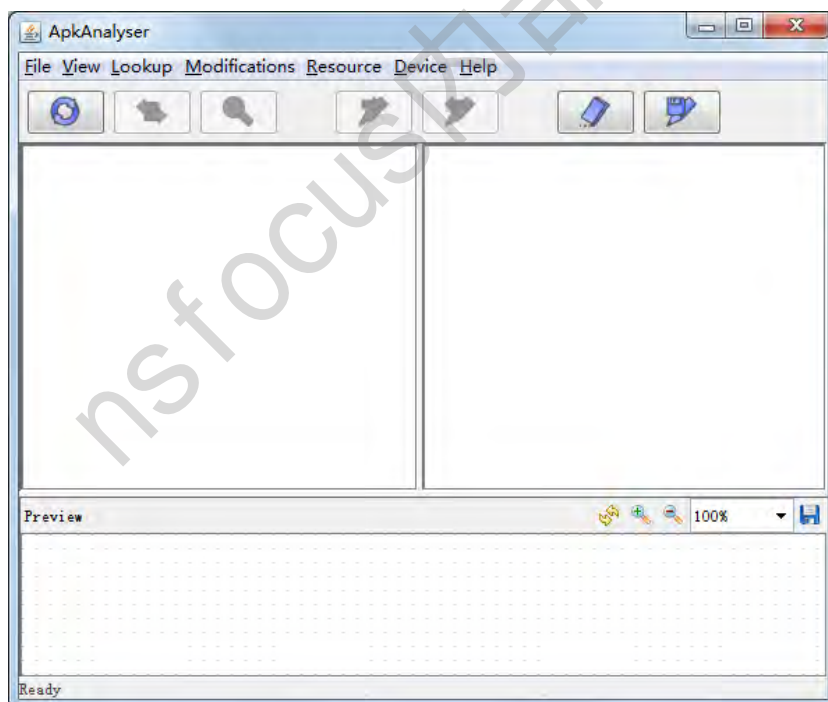
安装 [Xposed 框架](#)。框架支持 4.0.3 或更高版本的 android 系统。

5.10 集成分析工具

5.10.1 APKAnalyser

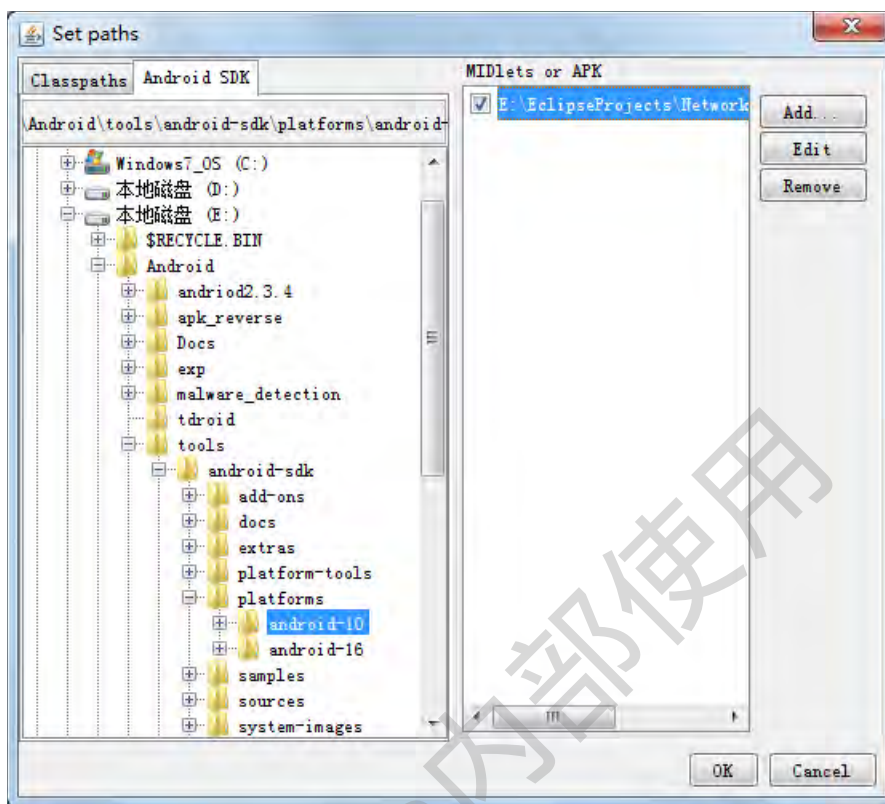
APKAnalyser 可用于直接查看 apk 文件的 smali 代码，类结构，可以给 apk 里各个类函数添加日志记录等。

1. 使用 `java -Xmx1024m -jar ApkAnalyser.jar` 来启动 APKAnalyser。因为分析 APK 时会使用大量内存，可能会导致内存耗尽，所以需要用 `-Xmx1024m` 指定 APKAnalyser 最多可以使用的内存数量。程序的主界面如下：

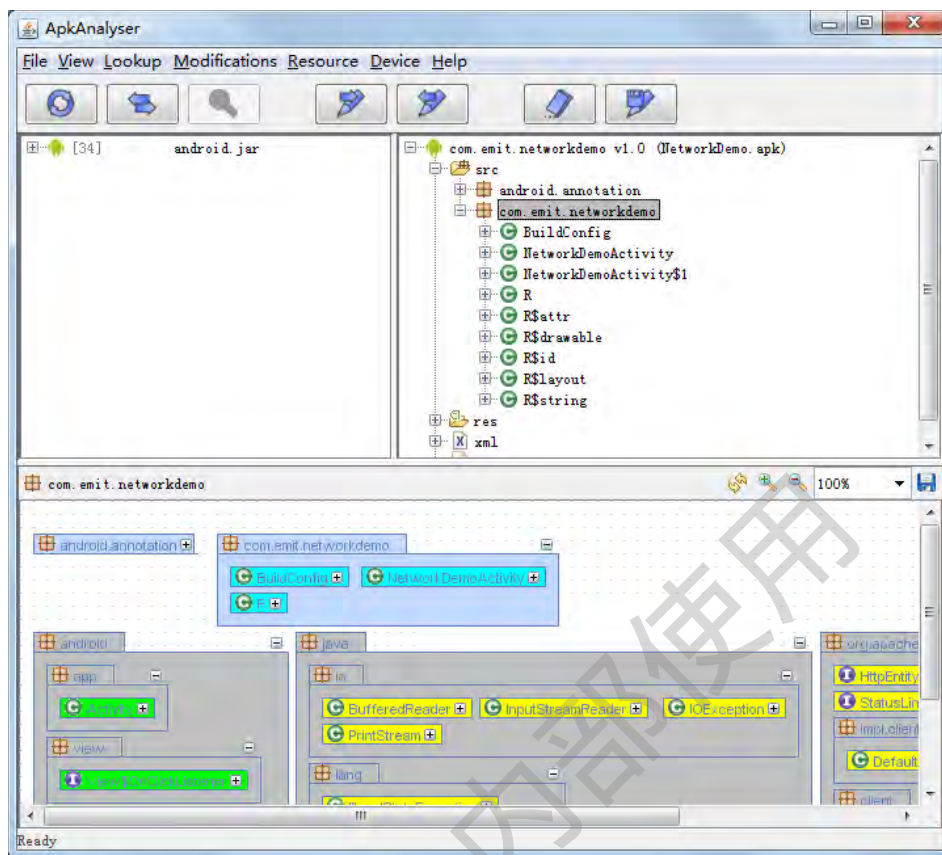


2. 在使用前需要先配置 APKAnalyser 的环境。
 - a) 在 File->Settings 中选择 adb 命令的路径。
 - b) 在 File->Set paths 中，首先设置 Classpaths，点击“Add...”添加被分析程序的类库文件(android.jar 一般是必须有的)，然后设置 Android SDK，选中 SDK 中 platforms 下的与手机执行环境匹配的版本目录。（此步骤非必须）

- c) 在右侧 “MIDlets or APK” 中添加要被分析的 APK 文件。



3. 点击 File->Analyze 开始分析过程。分析结束后会生成所有类的结构视图。



4. 点击右上区域树形结构的 xml 文件可以查看其文件内容（已被解码）。
5. 点击树形结构某个类中某个方法，可以查看其 smali 汇编代码。



The screenshot shows the Android Studio Console window with the following logcat output:

```

08-08 15:47:15.523 V/APKANALYSER ( 6125): < com.emit.networkdemo.NetworkDemoActivity:access$31(com.emit.networkdemo.NetworkDemoAc
08-08 15:47:15.523 V/APKANALYSER ( 6125): < com.emit.networkdemo.NetworkDemoActivity:access$32(com.emit.networkdemo.NetworkDemoAc
08-08 15:47:15.523 V/APKANALYSER ( 6125): @ ReadField com.emit.networkdemo.NetworkDemoActivity:access$32(com.emit.networkdemo.N
08-08 15:47:15.531 V/APKANALYSER ( 6125): < com.emit.networkdemo.NetworkDemoActivity:access$32(com.emit.networkdemo.NetworkDemoAc
08-08 15:47:15.531 V/APKANALYSER ( 6125): < com.emit.networkdemo.NetworkDemoActivity:inputStream2String(java.io.InputStream is)
08-08 15:47:15.539 V/APKANALYSER ( 6125): < com.emit.networkdemo.NetworkDemoActivity:inputStream2String(java.io.InputStream is)
08-08 15:48:00.039 V/APKANALYSER ( 6297): < com.emit.networkdemo.NetworkDemoActivity:<init>Ovoid(0, 23)
08-08 15:48:00.039 V/APKANALYSER ( 6297): < com.emit.networkdemo.NetworkDemoActivity:<init>Ovoid(3, 23)
08-08 15:48:00.039 V/APKANALYSER ( 6297): < com.emit.networkdemo.NetworkDemoActivity:onCreate(android.os.Bundle savedInstanceState)
08-08 15:48:00.507 V/APKANALYSER ( 6297): < com.emit.networkdemo.NetworkDemoActivity:onCreate(android.os.Bundle savedInstanceState)
08-08 15:48:17.484 V/APKANALYSER ( 6297): < com.emit.networkdemo.NetworkDemoActivity:access$30(com.emit.networkdemo.NetworkDemoAc
08-08 15:48:17.484 V/APKANALYSER ( 6297): < com.emit.networkdemo.NetworkDemoActivity:downloadFile(O.java.io.InputStream(0, 70)
08-08 15:48:17.484 V/APKANALYSER ( 6297): @ ReadField com.emit.networkdemo.NetworkDemoActivity:downloadFile(O.java.io.InputStre
08-08 15:48:22.789 V/APKANALYSER ( 6297): @ ReadLocal com.emit.networkdemo.NetworkDemoActivity:downloadFile(O.java.io.InputStre
08-08 15:48:22.789 V/APKANALYSER ( 6297): @ ReadLocal com.emit.networkdemo.NetworkDemoActivity:downloadFile(O.java.io.InputStre
08-08 15:48:22.789 V/APKANALYSER ( 6297): < com.emit.networkdemo.NetworkDemoActivity:downloadFile(O.java.io.InputStream(36, 87)
08-08 15:48:22.789 V/APKANALYSER ( 6297): < com.emit.networkdemo.NetworkDemoActivity:access$30(com.emit.networkdemo.NetworkDemoAc
08-08 15:48:22.789 V/APKANALYSER ( 6297): < com.emit.networkdemo.NetworkDemoActivity:access$31(com.emit.networkdemo.NetworkDemoAc
08-08 15:48:22.789 V/APKANALYSER ( 6297): < com.emit.networkdemo.NetworkDemoActivity:access$32(com.emit.networkdemo.NetworkDemoAc
08-08 15:48:22.789 V/APKANALYSER ( 6297): @ ReadField com.emit.networkdemo.NetworkDemoActivity:access$32(com.emit.networkdemo.NetworkDemoAc
08-08 15:48:22.789 V/APKANALYSER ( 6297): < com.emit.networkdemo.NetworkDemoActivity:access$32(com.emit.networkdemo.NetworkDemoAc
08-08 15:48:22.789 V/APKANALYSER ( 6297): < com.emit.networkdemo.NetworkDemoActivity:inputStream2String(java.io.InputStream is)
08-08 15:48:22.796 V/APKANALYSER ( 6297): < com.emit.networkdemo.NetworkDemoActivity:inputStream2String(java.io.InputStream is)

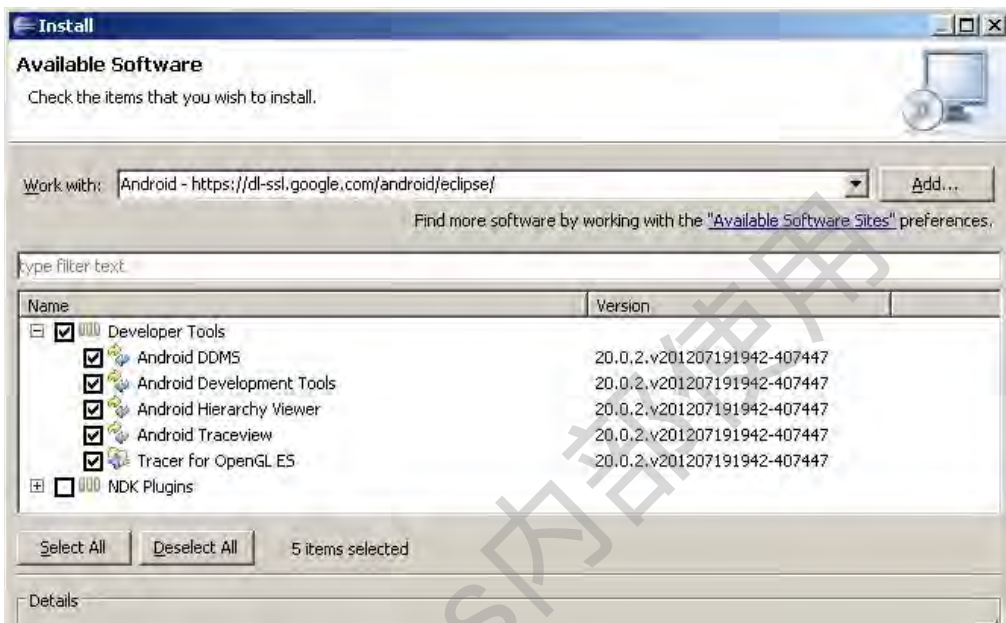
```

The bottom toolbar shows buttons for V, D, I, W, E, F, S (highlighted), Clear, Relaunch, and Close.

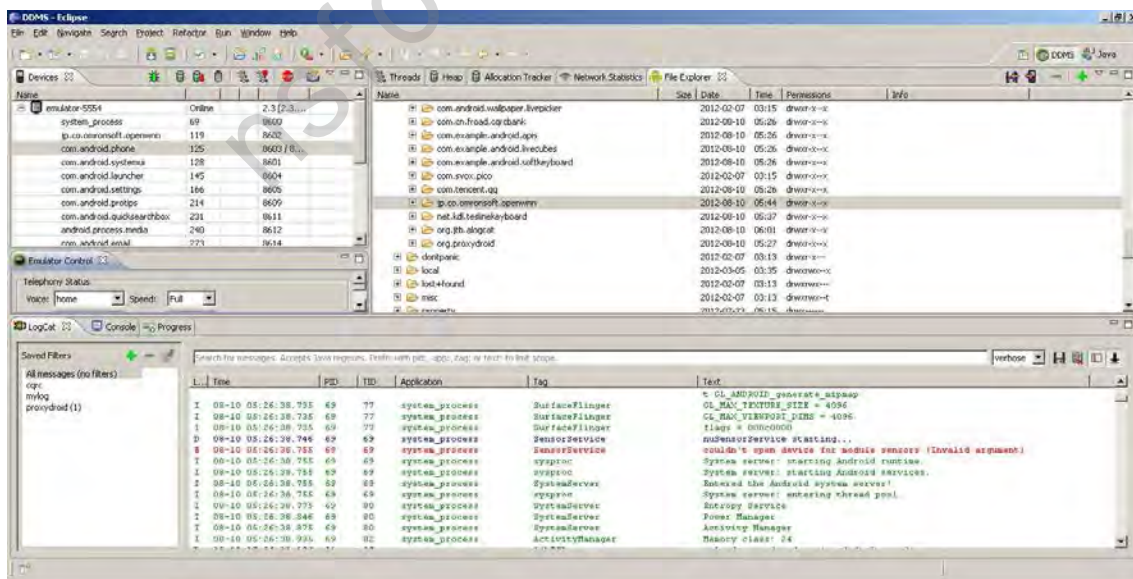
5.10.2 Eclipse

Eclipse 是用于 Java 开发的集成环境,在安装 ADT 插件后可以用于 android 应用的开发、调试和监控。

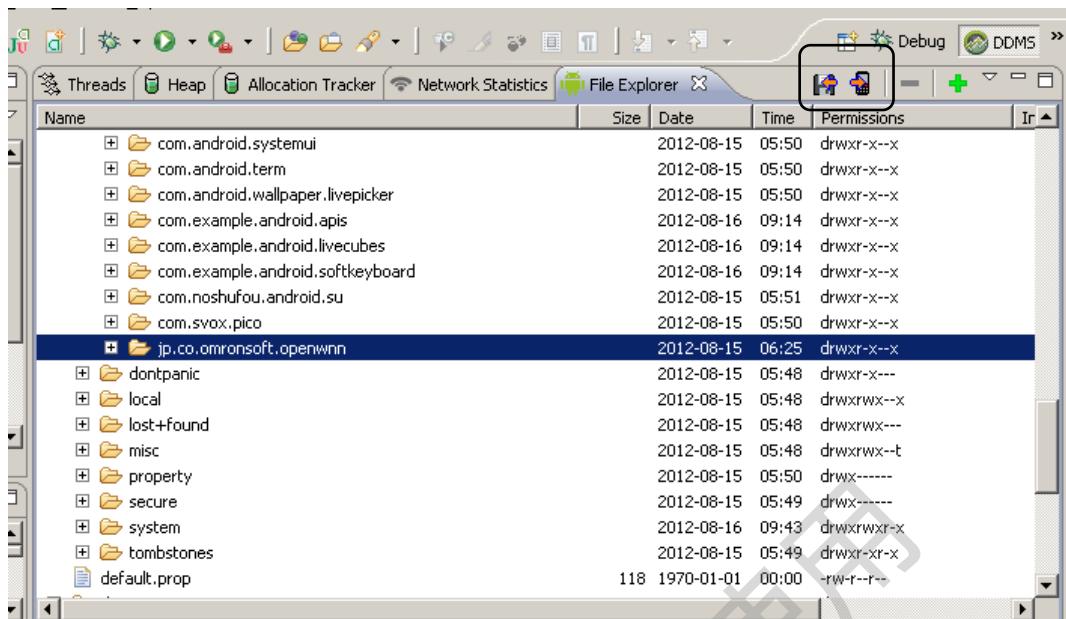
1. 在 eclipse 里安装 ADT 插件。



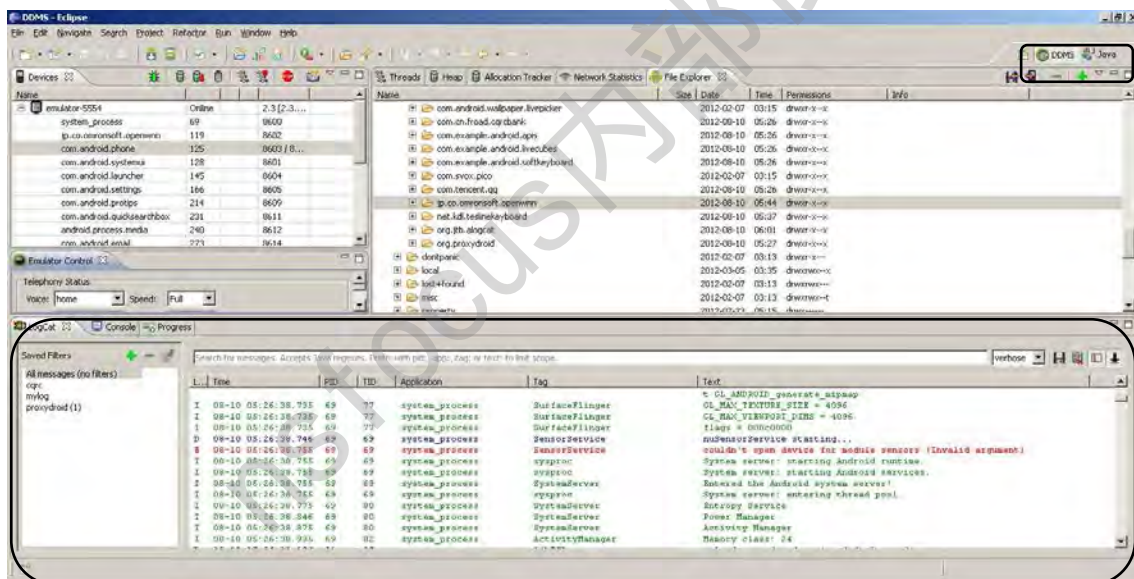
2. 切换到 eclipse 的 DDMS 视图,如下图所示。可以进行文件浏览,进程查看 / 管理,logcat 日志查看等等。



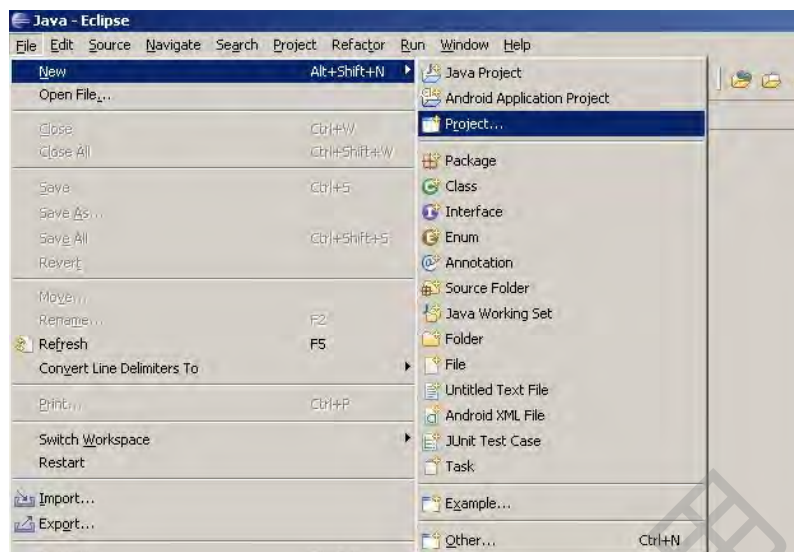
3. 在文件浏览窗口,可以实现 android 中文件的上传和下载。在选择要保存的文件或文件夹后,点击右上角的保存图标即可,如下图所示。



4. 在下方的 logcat 窗口，可以查看日志，还可以自定义过滤器，对日志进行过滤。



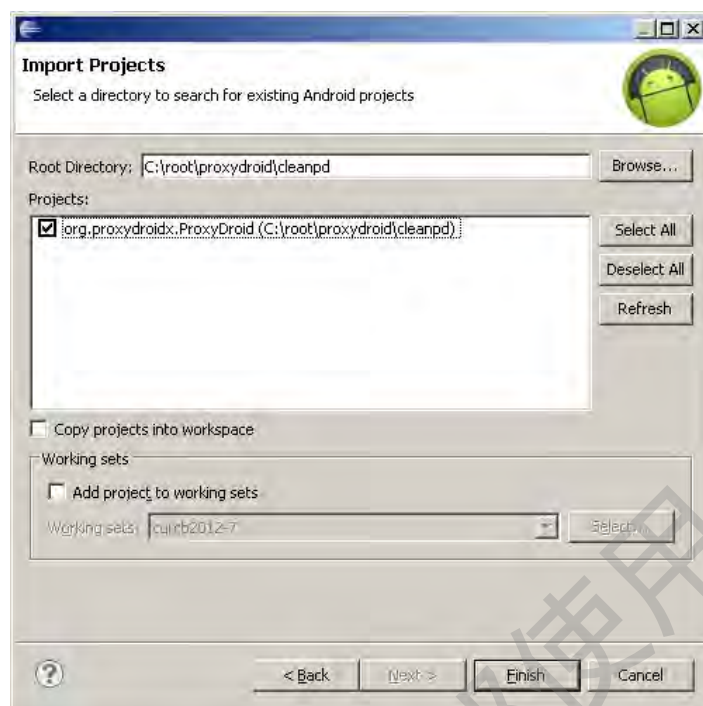
5. 导入已有的 android 项目。新建 project，如图所示



6. 选择从已有代码新建。

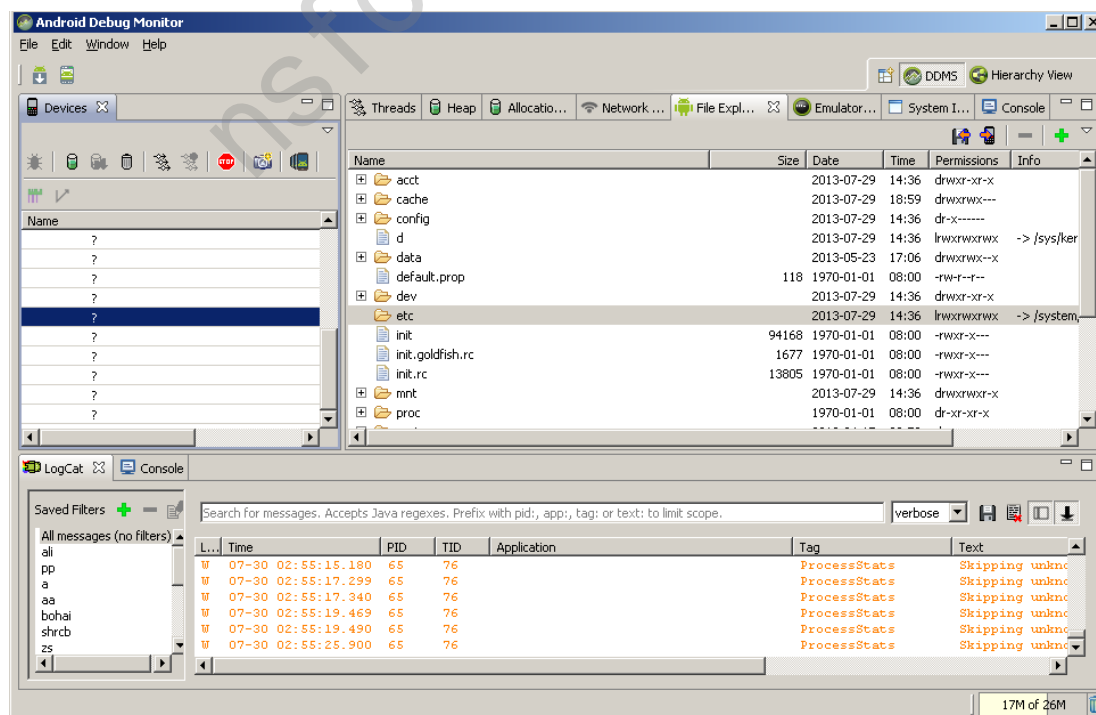


7. 接下来将项目根目录指向代码的根目录。点击“finish”完成代码的导入。



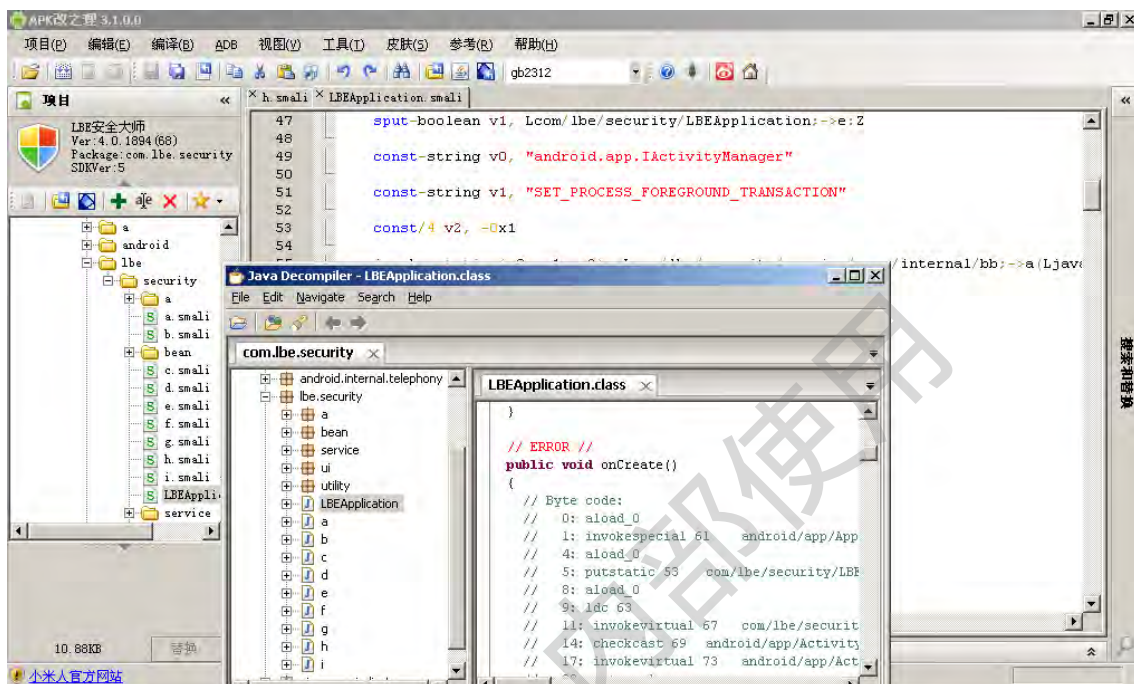
5.10.3 Android debug monitor

Android debug monitor 是 android 较新版 SDK 中自带的 GUI 工具，启动 android-sdk\tools\monitor.bat 即可运行。其功能与 Eclipse DDMS 界面基本相同。

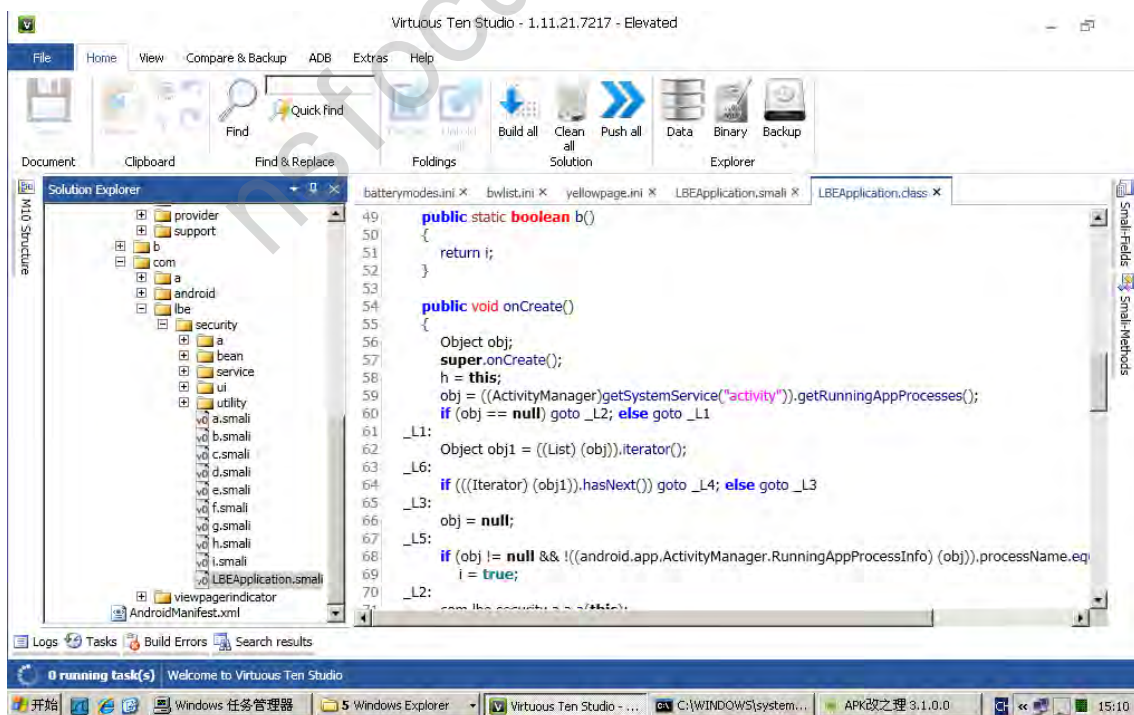


5.10.4 apk 编辑工具

1. ApkIDE, 中文名 APK 改之理。



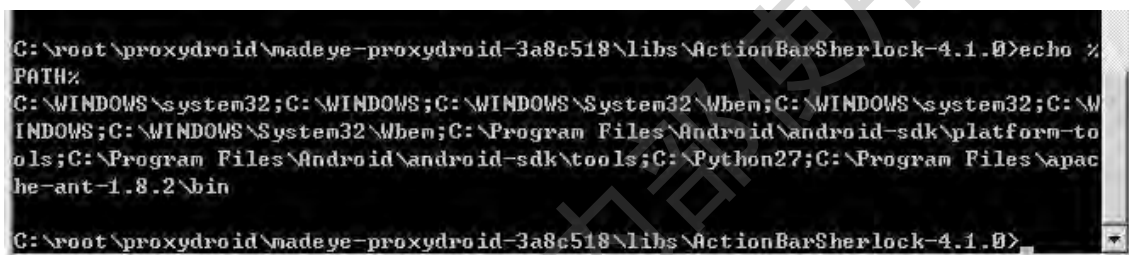
2. Virtuous Ten Studio, 国外的一款类似于改之理的工具。



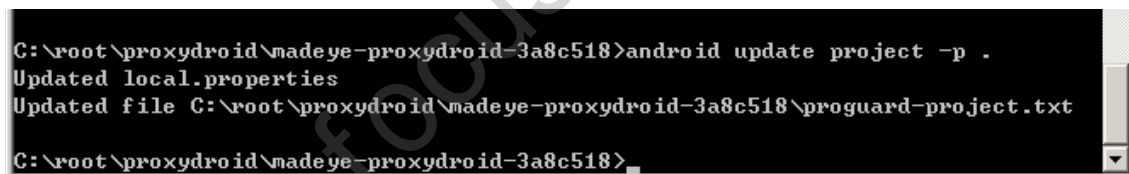
5.11 ant 编译源代码

对于指定使用 ant 编译的源代码，可以参考如下步骤：

1. 设置环境变量。JAVA_HOME 指向 JDK 的安装路径，PATH 中添加 android SDK 和 ant 可执行文件目录。



2. 检测编译环境。在代码的根目录输入“android update project -p .”。



3. 编译，安装。在代码的根目录，命令行输入“ant debug install”。将会以 debug 模式编译代码并将编译好的 apk 安装到默认的设备或虚拟机上。

5.12 动态调试

5.12.1 使用 eclipse+ADT

当得到的 java 代码可以编译通过时，可以使用本方法。

1. 参考 5.2.1 反编译为 java 代码得到 java 代码。
2. 参考 5.2.2 反编译为 smali 代码得到解码的资源文件。

3. 参考 5.10.2Eclipse 将上两步得到的应用代码导入到 eclipse 项目中，编译通过，生成 apk 包，就可以开始调试。

5.12.2 使用 IDA pro

介绍使用 IDA pro 调试在 Android SDK 虚拟机中运行的应用。

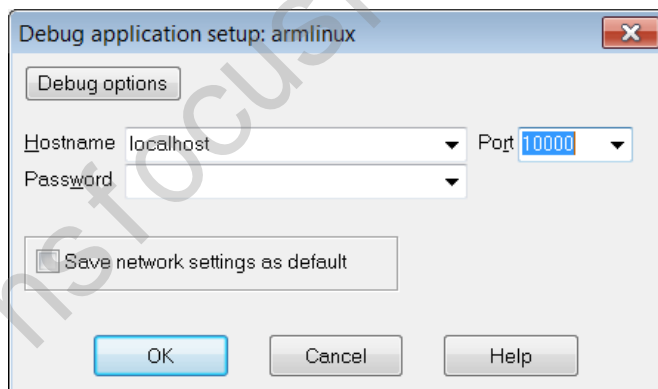
1. 将 IDA pro 自带的 android_server 拷贝到 android 系统中，运行

```
# ./android_server
./android_server
IDA Android 32-bit remote debug server(ST) v1.14. Hex-Rays (c) 2004-2011
Listening on port #23946...
```

2. 连接本机到虚拟机，telnet localhost 5554，然后如下图输入

```
redir add tcp:10000:23946
OK
```

3. 在 IDA 中选择“remote android server”，设置连接参数：



4. 连接成功后出现 android 系统的进程列表，可以附加到要调试的 apk 应用进程上调试了。

5.12.3 andbug 调试

此处介绍在 linux 系统中使用 andbug 调试 java 应用的方法。

1. 下载并编译 AndBug 源代码。

```
git clone https://github.com/swdunlop/AndBug.git
make
```

需要注意的是编译和使用 AndBug 需要先安装 python。

2. 配置环境变量 PYTHONPATH。

```
export PYTHONPATH=<AndBug-Dir>/lib
```

3. 在 Android 系统中启动要调试的程序，在命令行下输入 `adb shell ps`，找到相应的进程 pid。

4. 进入 AndBug 目录，执行 `./andbug` 可以查看详细的用法，说明环境配置成功。

```
emit@emit-VirtualBox:~/Downloads/swdunlop-AndBug-80d602f$ ./andbug

## AndBug (C) 2011 Scott W. Dunlop <swdunlop@gmail.com>
AndBug is a reverse-engineering debugger for the Android Dalvik virtual
machine employing the Java Debug Wire Protocol (JDWP) to interact with
Android applications without the need for source code. The majority of
AndBug's commands require the context of a connected Android device and a
specific Android process to target, which should be specified using the -d
and -p options.

The debugger offers two modes -- interactive and noninteractive, and a
comprehensive Python API for writing debugging scripts. The interactive mode
is accessed using:

$ andbug shell [-d <device>] -p <process>.

The device specification, if omitted, defaults in an identical fashion to the
ADB debugging bridge command, which AndBug uses heavily. The process
specification is either the PID of the process to debug, or the name of the
process, as found in "adb shell ps."

AndBug is NOT intended for a piracy tool, or other illegal purposes, but as
a tool for researchers and developers to gain insight into the
implementation of Android applications. Use of AndBug is at your own risk,
like most open source tools, and no guarantee of fitness or safety is made or
implied.

## Options:
-- -p, --pid <opt>
    the process to be debugged, by pid or name
-- -d, --dev <opt>
    the device or emulator to be debugged (see adb)
-- -s, --src <opt>
    adds a directory where .java or .smali files could be found
```

5. 可以通过 pid 来 attach 正在运行的进程。首先通过 `adb shell ps` 得到要调试的程序 pid。

例如这里我们得到 networkdemo2 的 pid 3581。

```
root      3560    2      0      0      c013ae5c 00000000 S flush-31:10
root      3561    2      0      0      c013ae5c 00000000 S flush-31:14
root      3562    2      0      0      c013ae5c 00000000 S flush-31:13
root      3563    2      0      0      c013ae5c 00000000 S flush-179:0
app_89    3581  1224  99900  24972  ffffffff afd0c51c S com.emit.networkdemo2
root      3590   1256   828    320  c00bc/b4 ar0c3ac S /system/bin/sh
root      3591   3590   984    324  00000000 afd0b45c R ps
```

6. 在 AndBug 目录下执行命令 `./andbug shell -p 3581` 即可进入 andbug 控制台。

```
emit@emit-VirtualBox:~/AndBug$ ./andbug shell -p 3581  
## AndBug (C) 2011 Scott W. Dunlop <swdunlop@gmail.com>  
>>
```

常用的指令有：

classes: 查看加载的 class

break: 下断点

suspend: 暂停进程

resume: 恢复进程运行

7. 输入 classes java.net 可以查看 java.net 包下所有已被加载的类。

```
>> classes java.net  
## Loaded Classes  
-- java.net.AddressCache  
-- java.net.InetAddress$WaitReachable  
-- java.net.SocketAddress  
-- java.net.Inet4Address  
-- java.net.InterfaceAddress  
-- java.net.InetAddress$1  
-- java.net.HttpURLConnection  
-- java.net.URI  
-- java.net.URL  
-- java.net.ContentHandler  
-- java.net.AddressCache$1  
-- java.net.InetSocketAddress  
-- java.net.Proxy  
-- java.net.URLConnection  
-- java.net.NetworkInterface  
-- java.net.Proxy$Type  
-- java.net.DatagramPacket  
-- java.net.Socket$ConnectLock  
-- java.net.SocketTimeoutException  
-- java.net.URLConnection$DefaultContentHandler  
-- java.net.SocketImplFactory  
-- java.net.UnknownHostException  
-- java.net.InetAddress  
-- java.net.JarURLConnection  
-- java.net.AddressCache$AddressCacheEntry  
-- java.net.SocketImpl  
-- java.net.SocketOptions  
-- java.net.MulticastGroupRequest  
-- java.net.NetPermission
```

8. 在 networkdemo2 这个例子中，我们可以对 java.net.URL 这个类下断点来获取程序要访问的网络资源。输入 break java.net.URL：


```
>> break java.net.URL
## Setting Hooks
-- Hooked <536870912> java.net.URL <class 'andbug.vm.Class'>
>>
```

这条命令会对 URL 类中的所有方法下断点，如果只想断其中某一个方法，可以在参数后面加上方法名。

9. 回到 Android 程序的界面，执行相应的操作，就会在 AndBug 中触发断点：

```
>> ## Breakpoint hit in thread <1> main (running suspended), process suspended.
-- java.net.URL.<init>(Ljava/lang/String;)V:0
-- com.emit.networkdemo2.NetworkDemo2$1.onClick(Landroid/view/View;)V:11
-- android.view.View.performClick()Z:14
-- android.view.View$PerformClick.run()V:2
-- android.os.Handler.handleCallback(Landroid/os/Message;)V:2
-- android.os.Handler.dispatchMessage(Landroid/os/Message;)V:4
-- android.os.Looper.loop()V:82
-- android.app.ActivityThread.main([Ljava/lang/String;)V:31
-- java.lang.reflect.Method.invokeNative(Ljava/lang/Object;[Ljava/lang/Object;
;Ljava/lang/Class;[Ljava/lang/Class;Ljava/lang/Class;IZ)Ljava/lang/Object;
<native>
-- java.lang.reflect.Method.invoke(Ljava/lang/Object;[Ljava/lang/Object;)Ljava
a/lang/Object;:18
-- com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run()V:11
-- com.android.internal.os.ZygoteInit.main([Ljava/lang/String;)V:84
-- dalvik.system.NativeStart.main([Ljava/lang/String;)V <native>
```

可以看到程序断在了 URL 的 init 方法处。

10. 启动 navi server，输入 navi 命令：

```
navi
## navigating process state at http://localhost:8080
-- Process suspended for navigation.
>>
```

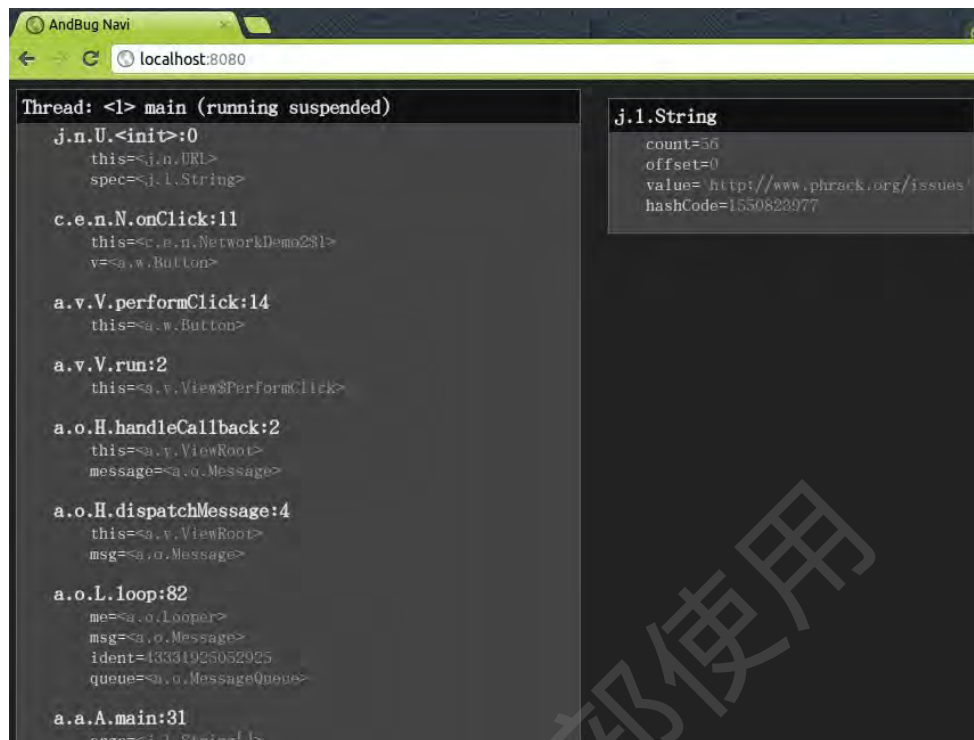
可以看到 server 的地址 <http://localhost:8080>。

如果输入 navi 命令提示：

```
>> navi
!! command not supported: "navi."
>>
```

则说明系统没有安装 python-bottle 包，需要先执行 `sudo apt-get install python-bottle` 下载安装 python-bottle。

11. 启动 navi server 后，通过浏览器打开 <http://localhost:8080>。在这里可以观察到当前被加载的线程及其状态，以及线程内的方法和变量。在本例中，从 init 方法中的 spec 成员变量可以读出程序要访问的资源地址。



注：代码中调用 SetDebugMode(false)，即可防护 andbug。

5.13 adb shell 命令

注意：很多手机厂商会将用不到的 shell 命令移除。下面的命令大部分可以在虚拟机中执行。

5.13.1 网络工具（root）

5.13.1.1 网络接口设备配置 netcfg

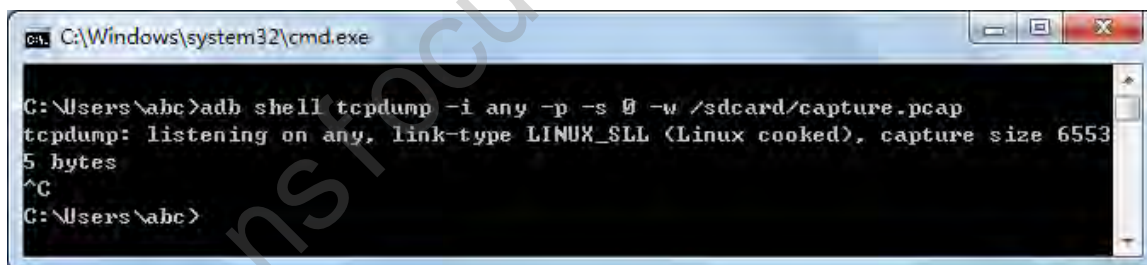
没有参数时，netcfg 可以列举当前设备的网络接口属性和状态，如图：

```
tcpdump: not found
# netcfg
netcfg
lo      UP      127.0.0.1      255.0.0.0      0x00000049
rmnet0  DOWN   0.0.0.0        0.0.0.0        0x00000000
rmnet1  DOWN   0.0.0.0        0.0.0.0        0x00000000
rmnet2  DOWN   0.0.0.0        0.0.0.0        0x00000000
rmnet3  DOWN   0.0.0.0        0.0.0.0        0x00001002
rmnet4  DOWN   0.0.0.0        0.0.0.0        0x00001002
rmnet5  DOWN   0.0.0.0        0.0.0.0        0x00001002
rmnet6  DOWN   0.0.0.0        0.0.0.0        0x00001002
rmnet7  DOWN   0.0.0.0        0.0.0.0        0x00001002
usb0    DOWN   192.168.42.129 255.255.255.0  0x00001002
tunl0   DOWN   0.0.0.0        0.0.0.0        0x00000080
sit0    DOWN   0.0.0.0        0.0.0.0        0x00000080
ip6tnl0 DOWN   0.0.0.0        0.0.0.0        0x00000080
# netcfg usb0
netcfg usb0
usage: netcfg [<interface> {dhcp|up|down}]
```

通过“netcfg rmnet0 up”可以打开 rmnet0 设备，与 ifconfig 类似。

5.13.1.2 嗅探流量 tcpdump

tcpdump 程序可以嗅探指定网络接口的所有网络流量。常用命令行如下图所示。选项-i 指定监听所有网络接口，-p 指定禁用混杂模式，-s 0 指定捕获完整数据包，-w 指定输出文件。执行命令后就会开始抓包，用 Ctrl+C 结束 tcpdump 的执行。得到 SD 卡上的 capture.pcap 文件后，可以在 pc 上结合 Wireshark 进一步分析。



5.13.1.3 监控流量 iftop

Android 系统中的 iftop 命令可以用来监控网卡的实时流量。

name	MTU	Rx bytes	Rx packets	Rx errs	Rx drpd	Tx bytes	Tx packets	Tx errs	Tx drpd
lo	16436	0	0	0	0	0	0	0	0
wlan0	1500	1285	1	0	0	635	2	0	0
lo	16436	0	0	0	0	0	0	0	0
wlan0	1500	831	6	0	0	1611	7	0	0
lo	16436	0	0	0	0	0	0	0	0
wlan0	1500	25548	23	0	0	2070	24	0	0
lo	16436	0	0	0	0	0	0	0	0
wlan0	1500	0	0	0	0	0	0	0	0
lo	16436	0	0	0	0	0	0	0	0
wlan0	1500	0	0	0	0	0	0	0	0
lo	16436	0	0	0	0	0	0	0	0
wlan0	1500	0	0	0	0	0	0	0	0
lo	16436	0	0	0	0	0	0	0	0
wlan0	1500	0	0	0	0	66	1	0	0
lo	16436	0	0	0	0	0	0	0	0
wlan0	1500	52	1	0	0	66	1	0	0
lo	16436	0	0	0	0	0	0	0	0
wlan0	1500	52	1	0	0	132	2	0	0
lo	16436	0	0	0	0	0	0	0	0
wlan0	1500	0	0	0	0	0	0	0	0

5.13.2 进程查看和监视 ps/top

ps 列举进程，如下图所示，最后一列为进程名，第二列是 PID。使用“ps -t”可以查看进程的线程信息。

USER	PID	PPID	UID	GID	CPU	MEM	COMMAND
root	38	1	828	344	c00b8fec	afd0c51c	/system/bin/qemu
shell	40	1	732	312	c0158eb0	afd0b45c	/system/bin/sh
root	41	1	3364	168	ffffffff	00008294	/sbin/adbd
system	61	33	143656	40656	ffffffff	afd0b6fc	system_server
root	84	41	732	328	c003da38	afd0c3ac	/system/bin/sh
app_30	115	33	82936	20164	ffffffff	afd0c51c	com.example.android.softk
keyboard							
radio	123	33	99176	24448	ffffffff	afd0c51c	com.android.phone
system	126	33	88744	25892	ffffffff	afd0c51c	com.android.systemui
app_13	143	33	95112	30476	ffffffff	afd0c51c	com.android.launcher
app_6	170	33	93664	26260	ffffffff	afd0c51c	android.process.acore
app_19	179	33	84312	21344	ffffffff	afd0c51c	com.android.deskclock
app_24	200	33	82976	19972	ffffffff	afd0c51c	com.android.protips
app_5	217	33	83520	20416	ffffffff	afd0c51c	com.android.music
app_2	225	33	84016	21200	ffffffff	afd0c51c	com.android.quicksearchbo
x							
app_0	236	33	86492	22440	ffffffff	afd0c51c	android.process.media
app_15	247	33	96632	21892	ffffffff	afd0c51c	com.android.mms
app_28	267	33	85972	22892	ffffffff	afd0c51c	com.android.email
root	289	84	888	324	00000000	afd0b45c	R ps
#							

top 用于监控系统中所有进程的状态。如下图所示，第一行是系统的 CPU 占用情况。

```

C:\WINDOWS\system32\cmd.exe

User 2%, System 4%, IOW 0%, IRQ 0%
User 6 + Nice 0 + Sys 10 + Idle 229 + IOW 0 + IRQ 0 + SIRQ 0 = 245

  PID CPU% S  #THR   USS   RSS PCY UID      Name
  290  5% R    1   912K   376K fg root    top
  126  0% S   11 88744K 25904K fg system  com.android.systemui
    3  0% S    1    0K    0K fg root    ksoftirqd/0
    4  0% S    1    0K    0K fg root    events/0
    5  0% S    1    0K    0K fg root    khelper
    6  0% S    1    0K    0K fg root    suspend
    7  0% S    1    0K    0K fg root    kblockd/0
    8  0% S    1    0K    0K fg root    cqueue
    9  0% S    1    0K    0K fg root    kseriod
   10  0% S    1    0K    0K fg root    kmmcd
   11  0% S    1    0K    0K fg root    pdfflush
   12  0% S    1    0K    0K fg root    pdfflush
   13  0% S    1    0K    0K fg root    kswapd0
   14  0% S    1    0K    0K fg root    aio/0
   22  0% S    1    0K    0K fg root    mtdblockd
   23  0% S    1    0K    0K fg root    kstripped
   24  0% S    1    0K    0K fg root    hid_compat
   25  0% S    1    0K    0K fg root    rpciod/0
   26  0% S    1    0K    0K fg root    mmcqd
   27  0% S    1   248K   152K fg root    /sbin/ueventd
   28  0% S    1   804K   276K fg system  /system/bin/servicemanager
   29  0% S    3  3864K   592K fg root    /system/bin/vold
   30  0% S    3  3836K   560K fg root    /system/bin/netd
   31  0% S    1   664K   264K fg root    /system/bin/debuggerd
  
```

5.13.3 系统调用记录 Strace

strace 命令可以截获并记录进程执行的系统调用以及进程接收的信号。每个系统调用的名称、参数以及返回值都将被输出。strace 命令有很多参数，常用命令行“strace -f -ff -x -v -F -s 512 -o logfile -p pid”，即监控进程号 pid 的进程中所有线程的系统调用，输出到以 logfile 为起始的多个文件中（每个进程一个文件）。

strace 的 -e 选项可以过滤记录。如“strace -e trace=file”将只跟踪以文件名为参数的函数调用，“strace -e trace=open,close,read,write”将只跟踪 open,close,read,write 四个系统调用。详细用法可参考[此博客](#)或[strace 手册](#)。

5.13.4 事件操作 getevent/sendevent

此命令需要 root 权限。可监控和模拟鼠标事件，按键事件，拖动滑动等等。使用 -p 选项可得到设备属性，如图所示。


```

C:\Documents and Settings\Administrator>adb shell getevent -p
add device 1: /dev/input/event8
  name:      "proximity"
  events:
    SYN <0000>: 0000 0003
    ABS <0003>: 0019 value 0, min 0, max 1, fuzz 0 flat 0
add device 2: /dev/input/event7
  name:      "lightsensor-level"
  events:
    SYN <0000>: 0000 0003
    ABS <0003>: 0028 value 3, min 0, max 9, fuzz 0 flat 0
add device 3: /dev/input/event6
  name:      "compass"
  events:
    SYN <0000>: 0000 0003
    ABS <0003>: 0000 value -11, min -1872, max 1872, fuzz 0 flat 0
               0001 value -734, min -1872, max 1872, fuzz 0 flat 0
               0002 value -8, min -1872, max 1872, fuzz 0 flat 0
               0003 value 0, min 0, max 360, fuzz 0 flat 0
               0004 value 0, min -180, max 180, fuzz 0 flat 0
               0005 value 0, min -90, max 90, fuzz 0 flat 0
               0006 value 0, min -30, max 85, fuzz 0 flat 0
               0007 value -1, min -32768, max 3, fuzz 0 flat 0
               0008 value 0, min -32768, max 3, fuzz 0 flat 0
               0009 value 0, min 0, max 65535, fuzz 0 flat 0

```

指定设备文件，则监听相应设备的事件。

```

C:\Documents and Settings\Administrator>adb shell getevent /dev/input/event3
0003 0030 00000000
0000 0000 00000000
0003 0030 00000000
0000 0000 00000000
0003 0030 00000000
0000 0000 00000000
0003 0030 00000023
0003 0032 00000004
0003 0035 0000020e
0003 0036 00000380
0000 0002 00000000
0000 0000 00000000
0003 0030 00000023
0003 0032 00000004
0003 0035 00000209
0003 0036 0000036b
0000 0002 00000000
0000 0000 00000000
0003 0030 00000023
0003 0032 00000005
0003 0035 00000200
0003 0036 00000305
0000 0002 00000000
0000 0000 00000000

```

sendevent 可向指定设备发送事件。如图是向 event0 发送 keycode 2，也就是数字 1。

```

C:\Documents and Settings\Administrator>adb shell
# sendevent /dev/input/event0 1 2 0
sendevent /dev/input/event0 1 2 0
# sendevent /dev/input/event0 1 2 1
sendevent /dev/input/event0 1 2 1
#

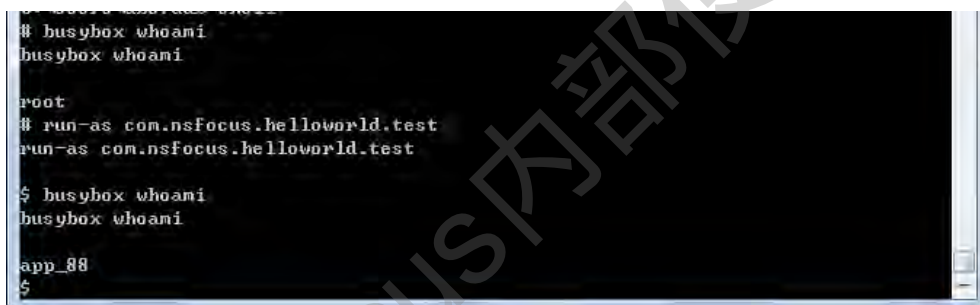
```

5.13.5 截图工具 Fbtool

可以使用 `adb shell /system/bin/fbtool -d /sdcard/screen.bmp` 命令将当前屏幕截图并保存为 `screen.bmp`。此工具需要 `root` 权限，而且不是所有手机上都有此程序。

5.13.6 用户切换 Run-as/su

此命令需要 `root` 或 `shell` 用户权限，且 `apk` 为 `debuggable`。命令行为：`run-as <应用名>`。运行后 `shell` 会切换到应用使用的用户，当前目录也会切换到应用的私有目录。需要注意的是，`run-as` 命令在包名小于 3 层的情况下会出现 `Package is unknown` 的现象，这可能算是一个 `bug`。



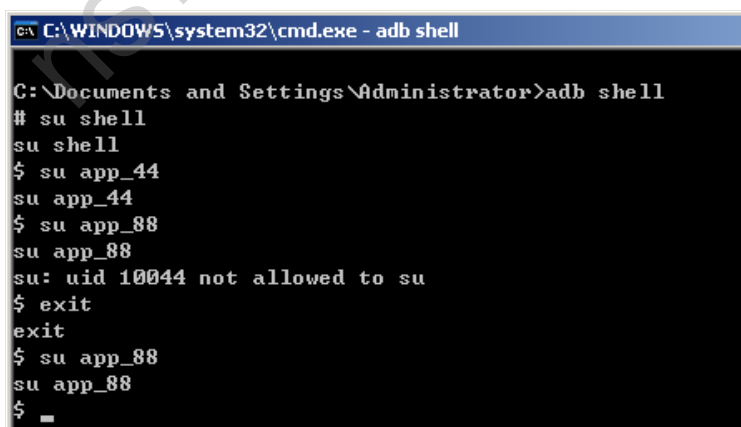
```
# busybox whoami
busybox whoami

root
# run-as com.nsfocus.helloworld.test
run-as com.nsfocus.helloworld.test

$ busybox whoami
busybox whoami

app_88
$
```

`su` 是 Linux 的标准命令行工具，用于“运行替换用户和组标识的 `shell`”。`su` 仅执行简单的用户切换，没有其他多余操作。命令行为：`su <用户名>`。如图所示：

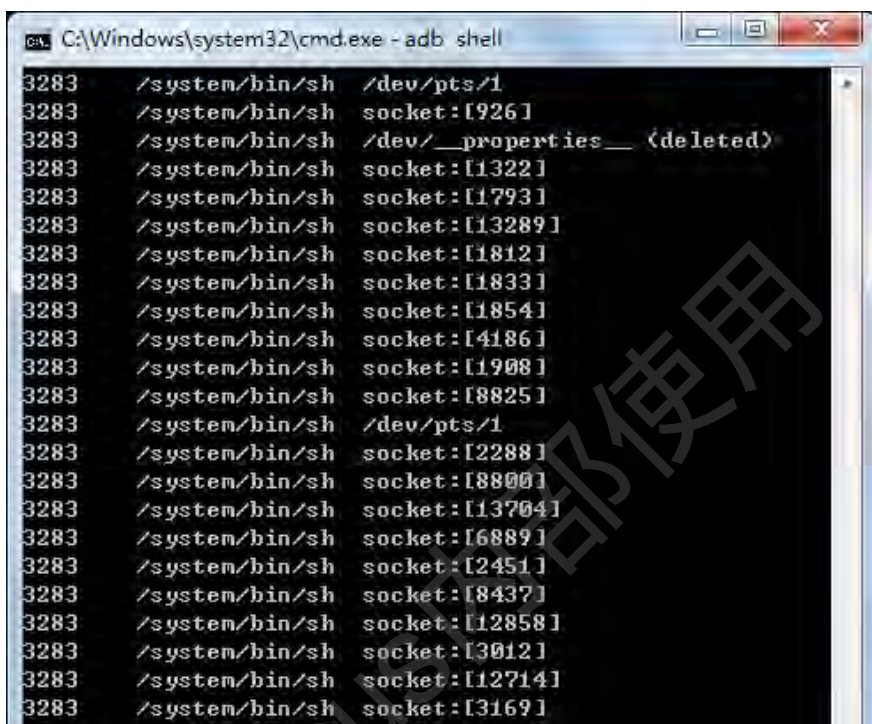


```
C:\WINDOWS\system32\cmd.exe - adb shell

C:\Documents and Settings\Administrator>adb shell
# su shell
su shell
$ su app_44
su app_44
$ su app_88
su app_88
su: uid 10044 not allowed to su
$ exit
exit
$ su app_88
su app_88
$ _
```

5.13.7 文件列举 lsof

Android 中的 lsof 命令可以显示所有进程打开的文件。其中第一列为打开文件的进程 pid，第二列是进程名，第三列是打开的文件名。



```
C:\Windows\system32\cmd.exe - adb shell
3283 /system/bin/sh /dev/pts/1
3283 /system/bin/sh socket:[926]
3283 /system/bin/sh /dev/__properties__ (deleted)
3283 /system/bin/sh socket:[1322]
3283 /system/bin/sh socket:[1793]
3283 /system/bin/sh socket:[13289]
3283 /system/bin/sh socket:[1812]
3283 /system/bin/sh socket:[1833]
3283 /system/bin/sh socket:[1854]
3283 /system/bin/sh socket:[4186]
3283 /system/bin/sh socket:[1908]
3283 /system/bin/sh socket:[8825]
3283 /system/bin/sh /dev/pts/1
3283 /system/bin/sh socket:[2288]
3283 /system/bin/sh socket:[8800]
3283 /system/bin/sh socket:[13704]
3283 /system/bin/sh socket:[6889]
3283 /system/bin/sh socket:[2451]
3283 /system/bin/sh socket:[8437]
3283 /system/bin/sh socket:[12858]
3283 /system/bin/sh socket:[3012]
3283 /system/bin/sh socket:[12714]
3283 /system/bin/sh socket:[3169]
```

5.13.8 数据库文件查看 sqlite3

sqlite3 的命令可以让用户手工输入并执行面向 SQLite 数据库的 SQL 命令。系统命令以. 开头，可以通过 .help 命令详细查看。SQL 命令须以 ; 结尾。首先以 db 文件为参数进入 sqlite3 命令行，然后就可以对挂载的数据库进行操作。

```
# sqlite3 mmsms.db
sqlite3 mmsms.db
SQLite version 3.6.22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .tables
.tables
addr                pdu                 threads
android_metadata    pending_msgs        words
attachments          rate                words_content
canonical_addresses  raw                 words_segdir
drm                  sms                  words_segments
part                 sr_pending
sqlite> select * from sms;
select * from sms;
111110086111344423662960111-112111Test 1101010
```

5.13.9 日志查看 logcat

命令行输入 `adb logcat` 可以查看 android 输出的日志记录。Android SDK 中的 DDMS 可以查看日志记录。

5.13.10 测试工具 Monkey

Monkey 是一个命令行工具，可以运行在模拟器里或实际设备中。它向系统发送伪随机的用户事件流，实现对正在开发的应用程序进行压力测试。`-p` 选项指定了测试的包名，参数 100 是随机模拟事件的次数。在命令执行过程中，设备会接受 monkey 产生的随机时间，画面也会随机切换。

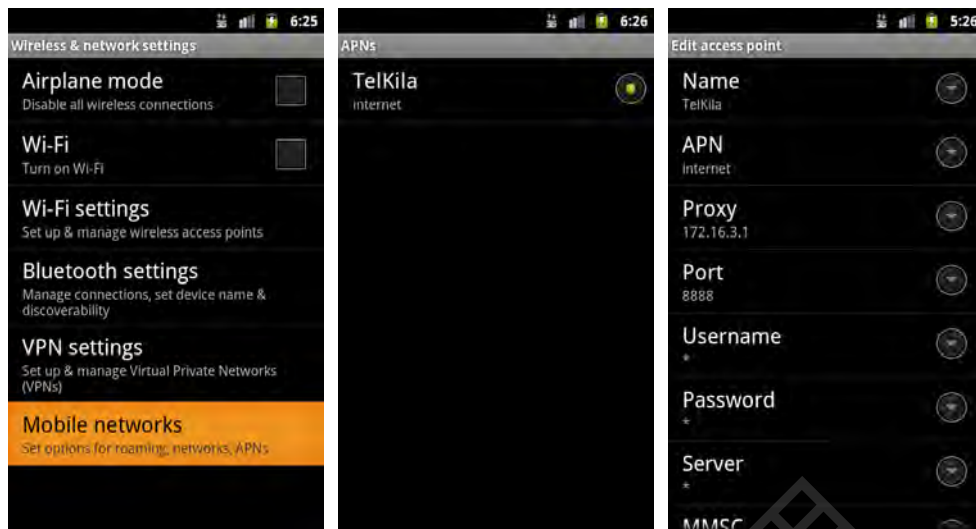
```
ex C:\Windows\system32\cmd.exe -adb -d shell
# monkey -p com.example.android.apis -v 100
monkey -p com.example.android.apis -v 100

:Monkey: seed=0 count=100
:AllowPackage: com.example.android.apis
:IncludeCategory: android.intent.category.LAUNCHER
:IncludeCategory: android.intent.category.MONKEY
// Event percentages:
// 0: 15.0%
// 1: 10.0%
// 2: 15.0%
// 3: 25.0%
// 4: 15.0%
// 5: 2.0%
// 6: 2.0%
// 7: 1.0%
// 8: 15.0%
:Switch: #Intent;action=android.intent.action.MAIN;category=android.intent.category.LAUNCHER;launchFlags=0x10000000;component=com.example.android.apis/.ApiDemos;end
// Allowing start of Intent < act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] cmp=com.example.android.apis/.ApiDemos > in package com.example.android.apis
:Sending Pointer ACTION_DOWN x=234.0 y=197.0
:Sending Pointer ACTION_UP x=238.0 y=190.0
// Rejecting start of Intent < act=android.intent.action.CALL_BUTTON cmp=com.android.contacts/.DialtactsActivity > in package com.android.contacts
:Sending Pointer ACTION_DOWN x=56.0 y=280.0
:Sending Pointer ACTION_UP x=69.0 y=271.0
:Sending Pointer ACTION_MOVE x=-1.0 y=1.0
:Sending Pointer ACTION_DOWN x=28.0 y=490.0
:Sending Pointer ACTION_UP x=28.0 y=490.0
:Sending Pointer ACTION_DOWN x=403.0 y=645.0
:Sending Pointer ACTION_UP x=397.0 y=661.0
:Sending Pointer ACTION_MOVE x=-1.0 y=-1.0
:Sending Pointer ACTION_DOWN x=98.0 y=752.0
:Sending Pointer ACTION_UP x=98.0 y=752.0
:Sending Pointer ACTION_MOVE x=0.0 y=-5.0
:Sending Pointer ACTION_DOWN x=264.0 y=458.0
:Sending Pointer ACTION_UP x=264.0 y=458.0
:Sending Pointer ACTION_DOWN x=113.0 y=170.0
:Sending Pointer ACTION_UP x=98.0 y=156.0
Events injected: 100
:Dropped: keys=0 pointers=0 trackballs=0 flips=0
## Network stats: elapsed time=1644ms (0ms mobile, 1644ms wifi, 0ms not connected)
```

5.14 android 代理配置

android 代理有两种选择，一是 android 自带，另一种是 apk 工具。

1. android 虚拟机，在设置->移动网络->access point names(APN)中，有代理 ip 和端口设置。
这种方式可以截获到所有 http(s)通信，与 http 通信采用的端口无关。



2. 在手机上安装 ProxyDroid 工具，运行后如图所示。在 Host 里指定代理 ip，Port 指定代理监听端口。此代理工具仅对 80，443，5228 等标准 http(s)端口有效。



3. 对于采用非标准端口的 http 通信，因为 ProxyDroid 本身就是通过 iptables 实现的透明代理，所以可手动输入 iptables 命令实现 ProxyDroid 代理。在打开 proxydroid 的代理功能后，对 http 通信，在手机中运行如下命令：

```
iptables -t nat -A OUTPUT -p tcp --dport SERVER_PORT -m tcp -j REDIRECT --to-ports 8123 或
```

```
iptables -t nat -A OUTPUT -p tcp --dport SERVER_PORT -j DNAT --to-destination  
127.0.0.1:8123
```

对 https 通信，在手机中运行如下命令：

```
iptables -t nat -A OUTPUT -p tcp --dport SERVER_PORT -m tcp -j REDIRECT --to-ports  
8124 或
```

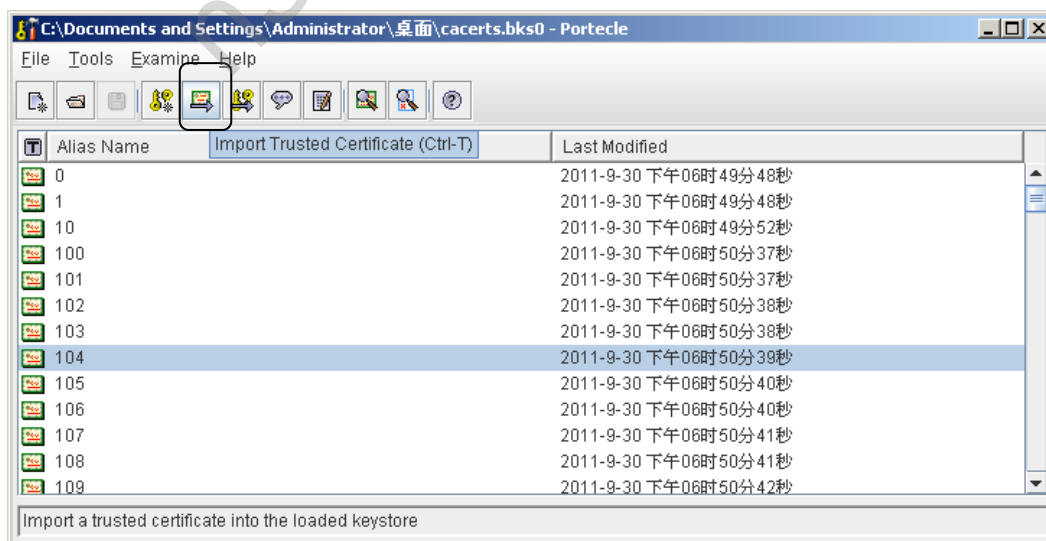
```
iptables -t nat -A OUTPUT -p tcp --dport SERVER_PORT -j DNAT --to-destination  
127.0.0.1:8124
```

其中 `SERVER_PORT` 是 http 服务端的端口。（因为有些手机的内核不支持“-j REDIRECT”，所以提供两种命令。）如果有多个端口，可以按照上述格式，为每个端口运行一次 iptables 命令。（android 虚拟机默认不支持 iptables，需要重新编译内核才能使用 proxydroid）

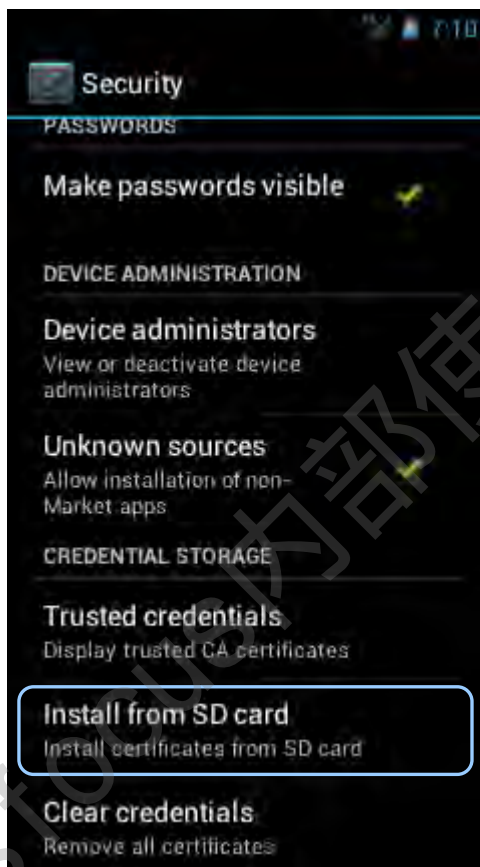
5.15 手机根证书安装

Android 将可信根证书存储在 `/system/etc/security/cacerts.bks` 文件（高版本 android，如 android 4.0 则在 `/system/etc/security/cacerts` 文件夹中）。android 系统使用

1. 将 burpsuit 的根证书导出，过程参考 http://portswigger.net/burp/help/proxy_options_installingcacert.html#firefox。
2. 在 adb shell 中运行 `adb pull /system/etc/security/cacerts.bks`，得到手机中的根证书包（android 4.0 可以跳过 2/3/4 步）。
3. 使用 Portecle 打开 `cacerts.bks`，当提示“Enter Password”时，直接点击 ok。打开文件后选择“Import Trusted Certificate”，将 burpsuit 的根证书导入到 bks 文件中。



4. adb shell 中运行 `su; mount -o remount,rw /system`。将 bks 文件上传到手机上，覆盖原文件：`adb push cacerts.bks /system/etc/security/`。然后重启手机。
5. android 4.0 及以上版本本身已包含根证书管理工具。将证书放置到 SD 卡根目录中，然后选择“设置->安全->从 SD 卡安装(Install from SD card)”即可。安装/禁用的证书都放置在 `/data/misc/keystore/`（或 `keychain`）目录下的相应子目录下。



5.16 drozer 组件测试工具

[drozer](#) 是一个测试 android 应用组件的安全工具，可以对应用导出的各类组件进行测试。drozer 具体使用方法可参考官网上的文档。

1. 安装 drozer 后，首先运行 `drozer.bat`，连接目标 android 系统。然后运行“`run app.package.attacksurface pkgname`”，获取应用的导出组件，如图：

```
dz> run app.package.attacksurface com.android.email
Attack Surface:
  11 activities exported
   4 broadcast receivers exported
   5 content providers exported
   8 services exported
```


2. 测试 content provider。使用的命令为（用于 cp 的命令有很多，测试时请参考文档）

run app.provider.info -a *pkgname* ， 获取导出信息

run scanner.provider.finduris -a *pkgname* ， 枚举 URI

run app.provider.query *URI* 请求数据

下图为测试 android 邮件应用的 content provider:

```
dz> run app.provider.info -a com.android.email
Package: com.android.email
Authority: com.android.email2.conversation.provider
Read Permission: null
Write Permission: null
Content Provider: com.android.mail.browse.EmailConversationProvider
Multiprocess Allowed: False
Grant Uri Permissions: True
Uri Permission Patterns:
Path: .*
Type: PATTERN_SIMPLE_GLOB
Content: com.android.email2.conversation.provider

dz> run scanner.provider.finduris -a com.android.email
Scanning com.android.email...
Unable to Query content://com.android.email2.conversation.provider
Unable to Query content://com.android.mail.mockprovider/account/
Unable to Query content://com.android.email.notifier
Unable to Query content://ui.email.android.com/settings/
Unable to Query content://
Unable to Query content://moveconversations/
Unable to Query content://

Accessible content URIs:
content://com.android.email2.accountcache/
content://com.android.email2.accountcache
dz> run app.provider.query content://com.android.email2.accountcache/
! _id ! name ! senderName ! accountManagerName ! type ! providerVersion ! accountUri ! folderListUri ! fullFolderListUri ! allFolderListUri ! searchUri ! accountFromAddresses ! expungeMessageUri ! undoUri ! accountSettingsIntentUri ! syncStatus ! helpIntentUri ! sendFeedbackIntentUri ! reauthenticationUri ! composeUri ! mimeType ! recentFolderListUri ! color ! defaultRecentFolderListUri ! manualSyncUri ! viewProxyUri ! accountCookieUri ! signature ! auto_advance ! message_text

dz> run app.provider.query content://com.android.email2.accountcache/ --vertical
1
_id 0
name phonesectest2013@gmail.com
senderName tester Aa
accountManagerName phonesectest2013@gmail.com
type com.android.email
providerVersion 1
accountUri content://com.android.email.provider/uiaccount/1
FolderListUri content://com.android.email.provider/uifolders/1
fullFolderListUri content://com.android.email.provider/uifullfolders/1
allFolderListUri content://com.android.email.provider/uiallfolders/1
searchUri content://com.android.email.provider/uisearch/1
accountFromAddresses null
expungeMessageUri
undoUri content://com.android.email.provider/uiundo
accountSettingsIntentUri content://ui.email.android.com/settings?account=1
```

当需要加入查询条件时，命令格式为：`run app.provider.query uri --selection-args rows --projection columns`

3. 测试 activity。使用的命令为

`run app.activity.info -a pkgname`，导出的 activity

`run app.activity.start --component pkgname activity`，打开 activity

下图是测试微信的 activity:

```
dz> run app.activity.info -a com.tencent.mm
Package: com.tencent.mm
com.tencent.mm.ui.LauncherUI
com.tencent.mm.ui.tools.ShareImgUI
com.tencent.mm.ui.tools.ShareToTimeLineUI
com.tencent.mm.plugin.base.stub.WXEntryActivity
com.tencent.mm.plugin.base.stub.WXPayEntryActivity
com.tencent.mm.plugin.accountsync.ui.ContactsSyncUI
```

```
dz> run app.activity.start --component com.tencent.mm com.tencent.mm.ui.tools.ShareImgUI
dz> run app.activity.start --component com.tencent.mm com.tencent.mm.ui.tools.ShareImgUI
dz> run app.activity.start --component com.tencent.mm com.tencent.mm.plugin.base.stub.WXEntryActivity
dz> run app.activity.start --component com.tencent.mm com.tencent.mm.plugin.base.stub.WXEntryActivity
dz> run app.activity.start --component com.tencent.mm com.tencent.mm.plugin.base.stub.WXPayEntryActivity
dz> run app.activity.start --component com.tencent.mm com.tencent.mm.plugin.accountsync.ui.ContactsSyncUI
dz>
```

4. 测试 service。使用的命令为

`run app.service.info -a pkgname`，导出的 service

`run app.service.send pkgname service`，访问 service

下图是测试微信的 service:

```
dz> run app.service.info -a com.tencent.mm
Package: com.tencent.mm
com.tencent.mm.plugin.accountsync.model.AccountAuthenticatorService
Permission: null
com.tencent.mm.plugin.accountsync.model.ContactsSyncService
Permission: null
dz>
```

```
dz>
dz> run app.service.send com.tencent.mm com.tencent.mm.plugin.accountsync.model.AccountAuthenticatorService --msg 1 2 3
Did not receive a reply from com.tencent.mm/com.tencent.mm.plugin.accountsync.model.AccountAuthenticatorService.
dz>
dz> run app.service.send com.tencent.mm com.tencent.mm.plugin.accountsync.model.ContactsSyncService --msg 1 2 3
Did not receive a reply from com.tencent.mm/com.tencent.mm.plugin.accountsync.model.ContactsSyncService.
dz>
```

5. 测试 broadcast。使用的命令为

`run app.broadcast.info -a pkgname`

```
run app.broadcast.send
```

六. Android 代码分析

6.1 Android 组件功能相关代码

6.1.1 Content provider

在 AndroidManifest.xml 中对 Content provider 的声明如下图所示。name 属性指示 java 代码类，authorities 指示访问的 uri。

```
<provider
    android:name="com.tencent.mm.plugin.base.stub.MMPluginProvider"
    android:authorities="com.tencent.mm.sdk.plugin.provider"
    android:exported="true"
    android:writePermission="com.tencent.mm.plugin.permission.WRITE" />
```

代码中的声明（斜体为搜索的关键词，下同）。下面 addURI 方法的参数说明访问这个 cp 使用的 URI 为 content://com.test.provider/table1/。

```
import android.content.ContentProvider;
import android.content.ContentValues;
import android.content.UriMatcher;
import android.database.Cursor;
import android.net.Uri;

public class MMPluginProvider
    extends ContentProvider
{
    private static final UriMatcher ehF;

    static
    {
        UriMatcher localUriMatcher = new UriMatcher(-1);
        ehF = localUriMatcher;
        localUriMatcher.addURI("com.test.provider", "table1", 2);
    }

    public Cursor query(Uri paramUri, String[] projection, String selection, String[]
selectionArgs, String sortOrder)
    {
        .....
    }
    .....
}
```

}

nsfocus内部使用