Exploratory Android™ Surgery

Digging into droids.

Jesse Burns

Black Hat USA 2009

Agenda

- Android Security Model
  - Android's new toys
  - Isolation basics
  - Device information sources
- Exploring Droids
  - Tracking down a Secret Code with Manifest Explorer
  - Exploring what's available with Package Play
  - Exploring what's going on with Intent sniffing
  - Quick look at Intent Fuzzing
- Conclusion
  - Hidden Packages, Root & proprietary bits
  - Common Problems

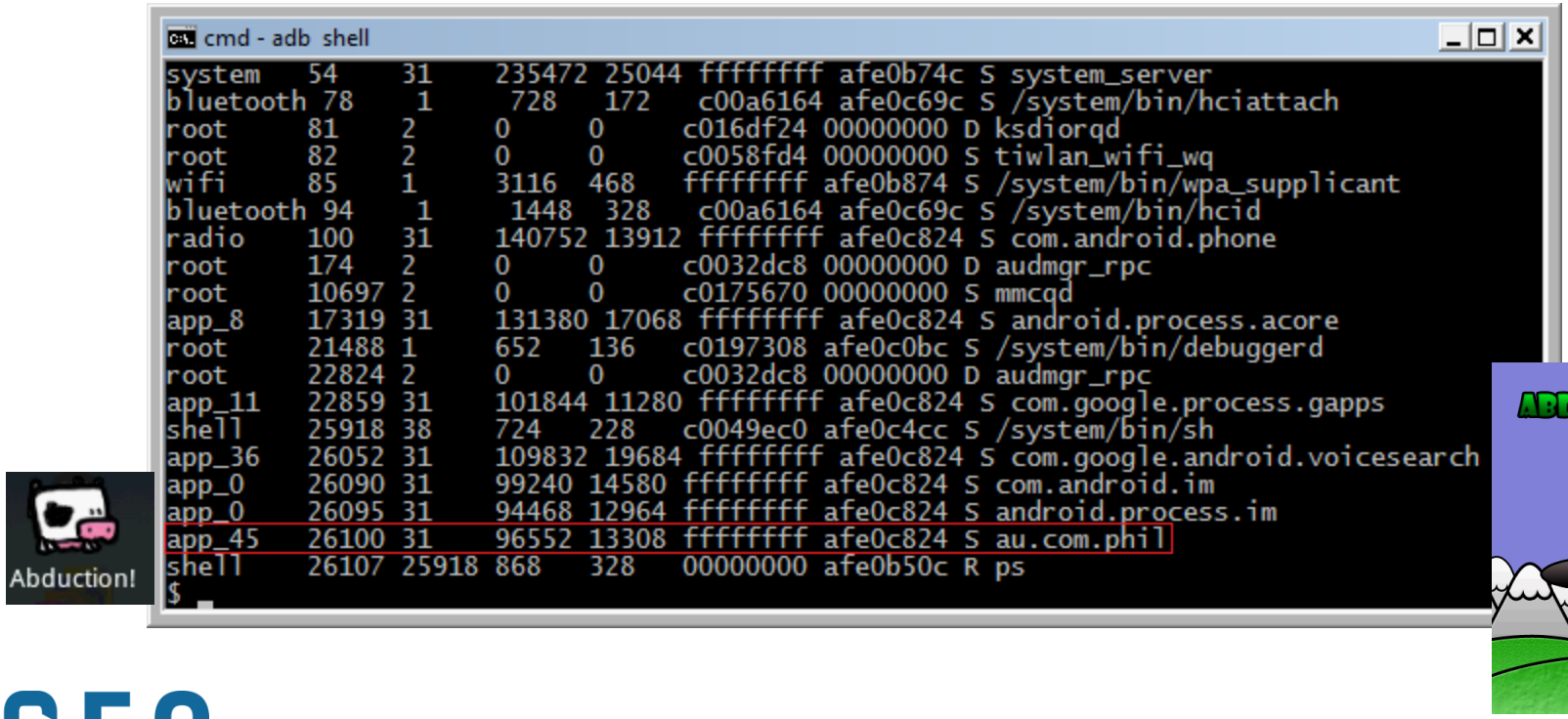
iSEC PARTNERS

Android Security Model

- Android's new toys
- Isolation Basics
- Device Information Sources

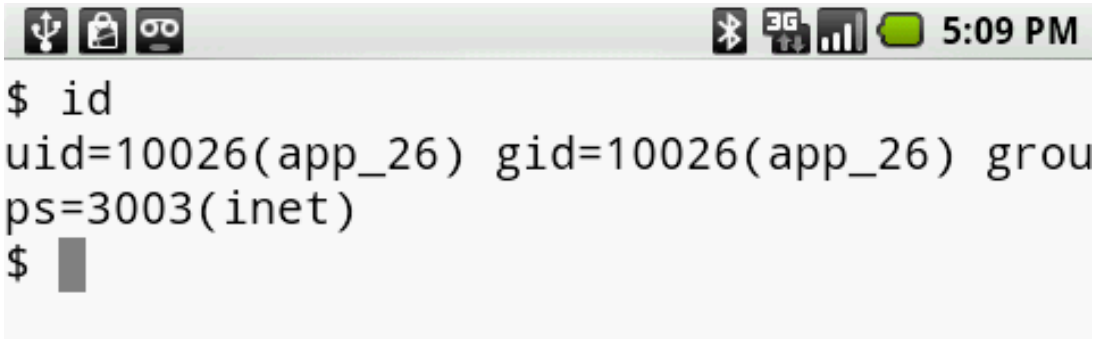
Android Security Model

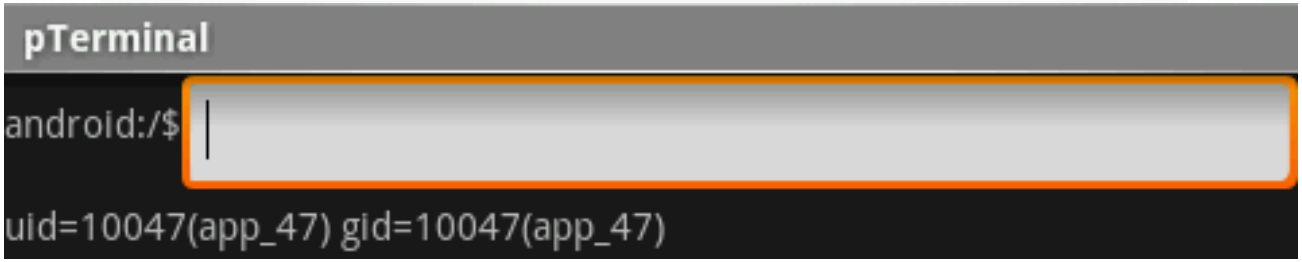
- Linux + Android's *Permissions*
- Application isolation – note editor can't read email
- Distinct UIDs and GIDs assigned on install



Android Security Model

- Rights expressed as *Permissions* & Linux groups!





iSEC PARTNERS

Android's New User Mode Toys

- **Activities** – Screens that do something, like the dialer
- **Services** – background features, like the IM service
- **Broadcast Receivers** – actionable notifications (startup!)
- **Content Providers** – shared relational data
- **Instrumentations** – rare, useful for testing

All secured with Android Permissions like:

"android.permission.READ_CONTACTS" or

"android.permission.BRICK"

See Manifest.permissions and AndroidManifests near you

iSEC PARTNERS

Android's New Toys: Intents

- Like hash tables, but with a little type / routing data
- Routes via an Action String and a Data URI
- Makes platform component replacement easy
- Either implicitly or explicitly routed / targeted

Intent { action=android.intent.action.MAIN
categories={android.intent.category.LAUNCHER}
flags=0x10200000
comp={au.com.phil/au.com.phil.Intro} }

iSEC PARTNERS

Android's Attack Surfaces

- Isolated applications is like having multi-user system
- Single UI / Device → Secure sharing of UI & IO
- Principal maps to code, not user (like browsers)
- Appeals to user for all security decisions i.e. Dialer
- Phishing style attack risks.

- Linux, not Java, sandbox. Native code not a barrier.
- Any java app can exec a shell, load JNI libraries, write and exec programs – without finding a bug.

iSEC PARTNERS

Android's Attack Surfaces

- System Services – Not a subclass of Service
  - Privileged: some native "servicemanager"
    - Some written in Java, run in the system_server
  - SystemManager.listServices() and getService()
  - Exposed to all, secured at the Binder interfaces

44 on a Annalee's Cupcake1.5r3 T-Mobile G1: activity, activity.broadcasts, activity.providers, activity.senders, activity.services, alarm, appwidget, audio, battery, batteryinfo, bluetooth, bluetooth_a2dp, checkin, clipboard, connectivity, content, cpuinfo, devicestoragemonitor, hardware, input_method, iphonesubinfo, isms, location, media.audio_flinger, media.camera, media.player, meminfo, mount, netstat, notification, package, permission, phone, power, search, sensor, simphonebook, statusbar, SurfaceFlinger, telephony.registry, usagestats, wallpaper, wifi, window

System Service Attack Surface

- Some are trivial IClipboard.aidl – ClipboardService

Or "clipboard" to getService()

- CharSequence getClipboardText();
- setClipboardText(CharSequence text);
- boolean hasClipboardText();

```
public CharSequence getClipboardText() {
    synchronized (this) {
        return mClipboard;
    }
}
```

iSEC PARTNERS

System Service Attack Surface

Some system services are complex, even with source:

SurfaceFlinger Native Code (C++)

no AIDL defining it or simple Stubs to call it with.

WindowManagerService. performEnableScreen ()

```
IBinder surfaceFlinger = ServiceManager.getService("SurfaceFlinger");
if (surfaceFlinger != null) {
    //Log.i(TAG, "******* TELLING SURFACE FLINGER WE ARE BOOTED! ");
    Parcel data = Parcel.obtain();
    data.writeInterfaceToken("android.ui.ISurfaceComposer");
    surfaceFlinger.transact(IBinder.FIRST_CALL_TRANSACTION,
                            data, null, 0);
```

iSEC PARTNERS

Android's New Kernel Mode Toys

- Binder - /dev/binder
  - AIDL: Object Oriented, Fast IPC, C / C++ / Java
  - Atomic IPC – ids parties, moves Data, FDs & Binders
  - Similar to UNIX domain sockets

- Ashmem – Anonymous shared memory
  - Shared memory that can be reclaimed (purged) by the system under low memory conditions.
  - Java support: android.os.MemoryFile

New Android Toys

18 Android devices by 8 or 9 manufacturers in 2009?








Images from High End Mobile Graphix blog.
http://highendmobilegrafix.blogspot.com/
Bottom right image from Gizmodo
http://www.gizmodo.com

iSEC PARTNERS

Understanding New Devices

- What software is installed on my new phone?
- Anything new, cool, or dangerous added by the manufacturer or new features for my apps to use?
- How will updates work? Do they have something for deleting that copy of 1984(*) from my library.
- Is the boot loader friendly?
- Will I have root? What about someone else?
- Which apps are system and which are data.

* Even if Amazon or Ahmadinejad intend to update you, it shouldn't be a surprise

Exploratory Tools

- Logcat or DDMS or the "READ_LOGS" permission!
- Android SystemProperties - property_service
- Linux
  - /proc
  - /sys (global device tree)
    - /sys/class/leds/lcd-backlight/brightness
  - dmesg i.e. calls to syslog / klogctl
  - syscall interface
  - File system o+r or groups we can join
    - APKs in /system/app

iSEC PARTNERS

Exploratory Tools

- /data/system/packages.xml
  - Details of everything installed, who shares signatures, definitions of UIDs, and the location of the install APKs for you to pull off and examine.
- /proc/binder – the binder transaction log, state, and stats
- /proc/binder/proc/
  - File for each process using binder, and details of every binder in use – read binder.c
- /dev/socket – like zygote and property_service
- /system/etc/permissions/platform.xml

Exploratory Tools

- DUMP permission – adb shell or granted

```
public void dump(FileDescriptor fd, String[] args) throws RemoteException;
```

  - dumpsys – dumps every system service ServiceManager.listServices()

Example from "activity.provider" dump:

Provider android.server.checkin…

package=android process=system…uid=1000

clients=[ProcessRecord{4344fad0
1281:com.android.vending/10025}, ProcessRecord{433fd800
30419:com.google.process.gapps/10011},
ProcessRecord{43176210 100:com.android.phone/1001},
ProcessRecord{43474c68 31952:com.android.calendar/10006},
ProcessRecord{433e2398 30430:android.process.acore/10008}]

iSEC PARTNERS

Exploratory Tools

- Android Manifest aka AndroidManifest.xml
  - Not only does the system have one, but every app
  - Defines exported attack surface including:
    - Activities, Services, Content Providers, Broadcast Receivers, and Instrumentations
- SystemServices / those privileged System APIs
  - Primarily what my tools use
    - Package Manager - "package" service
    - Activity Manager – "activity"
    - Some non-services like Settings

iSEC PARTNERS

Looking at "Secret Codes"

android.provider.Telephony (private @hide code)

caught my eye with this:

```
/**
 * Broadcast Action: A "secret code" has been entered in the dialer. Secret codes are
 * of the form *#*#<code>#*#*. The intent will have the data URI:</p>
 *
 * <p><code>android_secret_code://&lt;code&gt;</code></p>
 */
    public static final String SECRET_CODE_ACTION =
            "android.provider.Telephony.SECRET_CODE";
```

Grep also noticed SECRET_CODE_ACTION in:

/packages/apps/Contacts - SpecialCharSequenceMgr.java

/packages/app/VoiceDialer - VoiceDialerReceiver.java

iSEC PARTNERS

Looking at "Secret Codes"

SpecialCharSequenceMgr.java (From contacts)

```
/**
 * Handles secret codes to launch arbitrary activities in the form of *#*#<code>#*#*.
 * If a secret code is encountered an Intent is started with the android_secret_code://<code>
 * URI.
 *
 * @param context the context to use
 * @param input the text to check for a secret code in
 * @return true if a secret code was encountered
 */
  static boolean handleSecretCode(Context context, String input) {
      // Secret codes are in the form *#*#<code>#*#*
      int len = input.length();
      if (len > 8 && input.startsWith("*#*#") && input.endsWith("#*#*")) {
          Intent intent = new Intent(Intents.SECRET_CODE_ACTION,
                  Uri.parse("android_secret_code://" + input.substring(4, len - 4)));
          context.sendBroadcast(intent);
          return true;
      }

      return false;
  }
```

iSEC PARTNERS

Looking at "Secret Codes"

VoiceDialer's use of Secret Code – start at the Manifest:

```
<receiver android:name="VoiceDialerReceiver">
...
  <!--  Voice Dialer Logging Enabled, *#*#VDL1#*#* -->
  <intent-filter>
    <action android:name="android.provider.Telephony.SECRET_CODE" />
    <data android:scheme="android_secret_code" android:host="8351" />
  </intent-filter>
  <!--  Voice Dialer Logging Disabled, *#*#VDL0#*#* -->
  <intent-filter>
    <action android:name="android.provider.Telephony.SECRET_CODE" />
    <data android:scheme="android_secret_code" android:host="8350" />
  </intent-filter>
</receiver>
```

iSEC PARTNERS

Exploring Droids

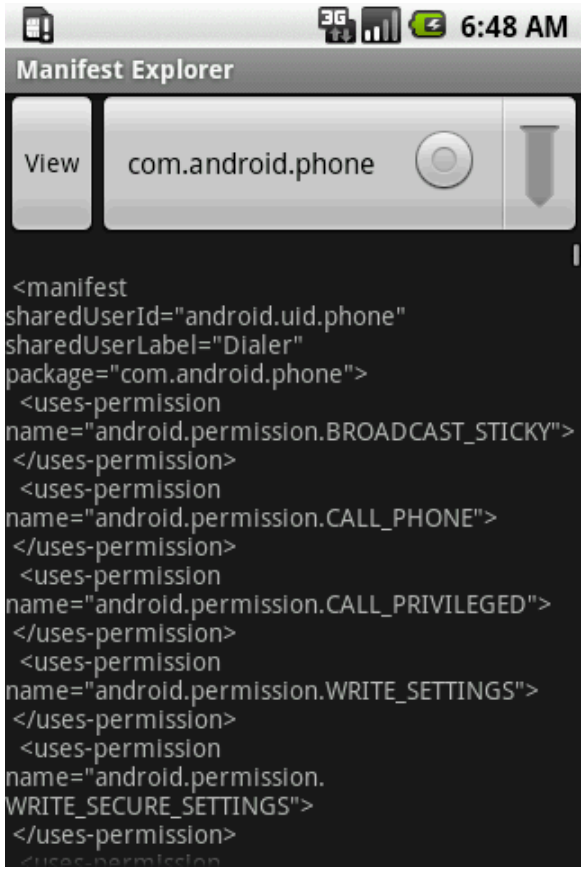
Tracking down a Secret Code with Manifest Explorer
Exploring what's available with Package Play
Exploring with Intent Sniffing
Quick look at Intent Fuzzing

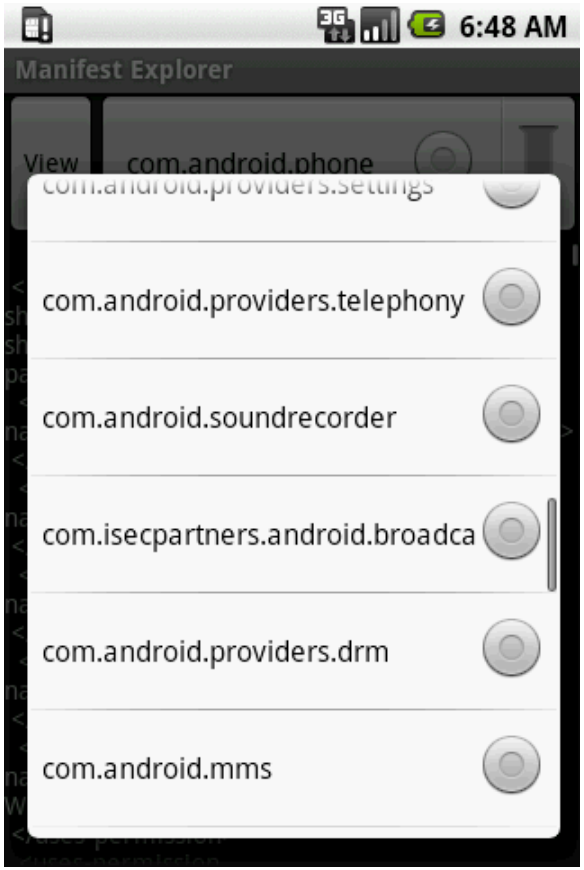
Manifests and Manifest Explorer

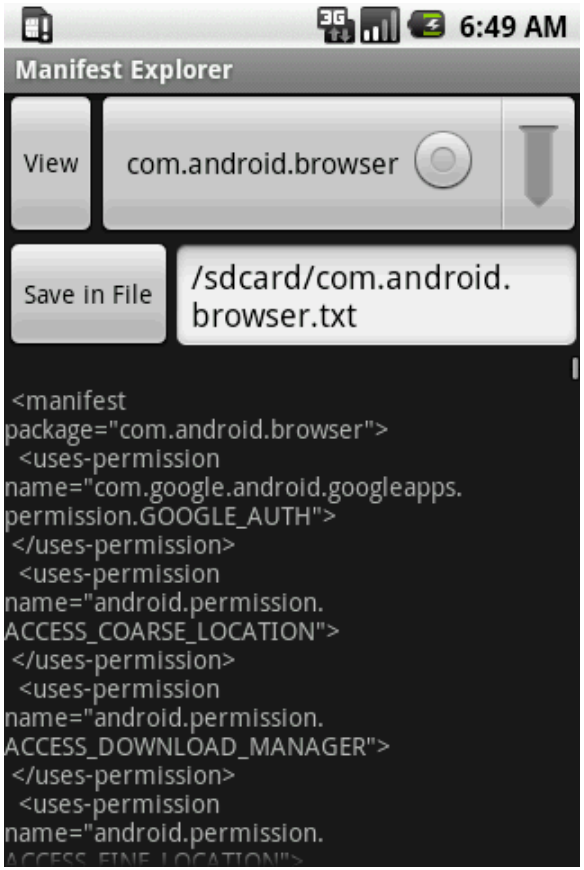
- Applications and System code has AndroidManifest
- Defines permissions, and their use for the system
- Defines attack surface
- Critical starting point for understanding security

- Stored in compressed XML (mobile → small) in .apk

Manifests and Manifest Explorer







iSEC PARTNERS

Manifests and Manifest Explorer

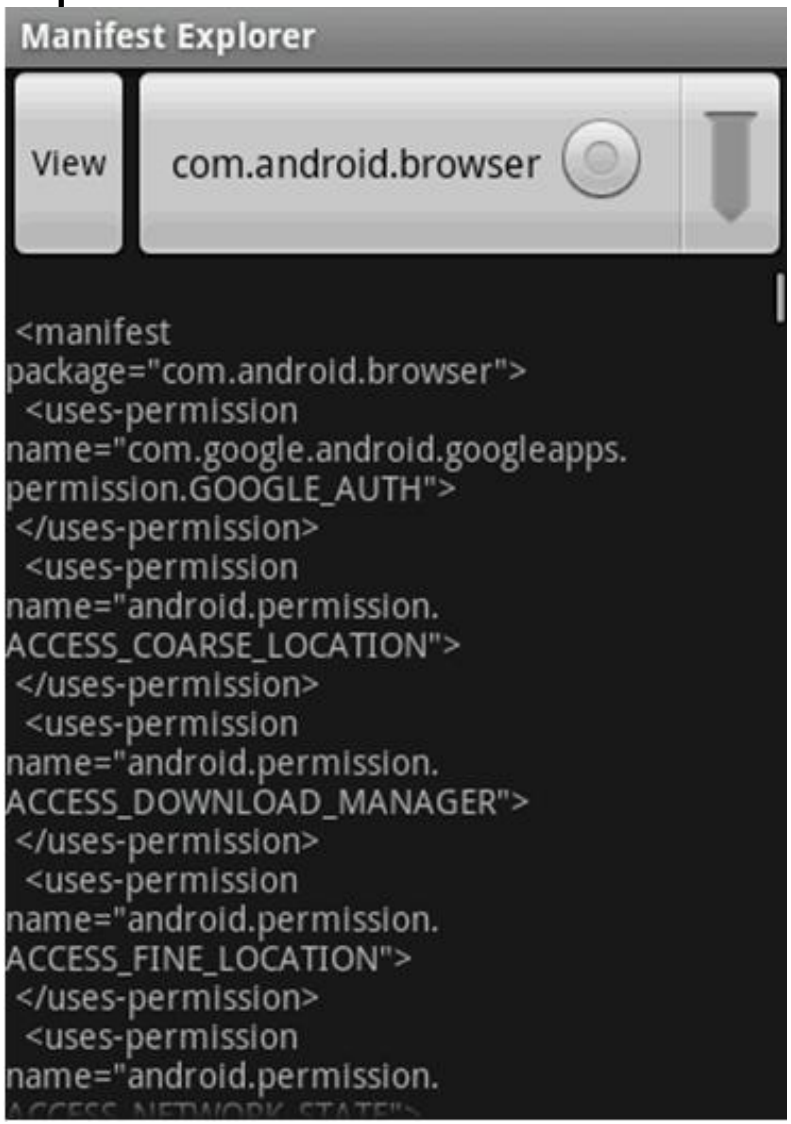
Start of Browser's Manifest (com.android.browser)

```
<!--
/* //device/apps/Browser/AndroidManifest.xml
**
** Copyright 2006, The Android Open Source Project
**
** Licensed under the Apache License, Version 2.0 (the "License");
** you may not use this file except in compliance with the License.
** You may obtain a copy of the License at
**
**     http://www.apache.org/licenses/LICENSE-2.0
**
** Unless required by applicable law or agreed to in writing, software
** distributed under the License is distributed on an "AS IS" BASIS,
** WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
** See the License for the specific language governing permissions and
** limitations under the License.
*/
-->

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.android.browser">

    <uses-permission
    android:name="com.google.android.googleapps.permission.GOOGLE_AUTH" />
    <uses-permission
    android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

iSEC PARTNERS

Manifests and Manifest Explorer

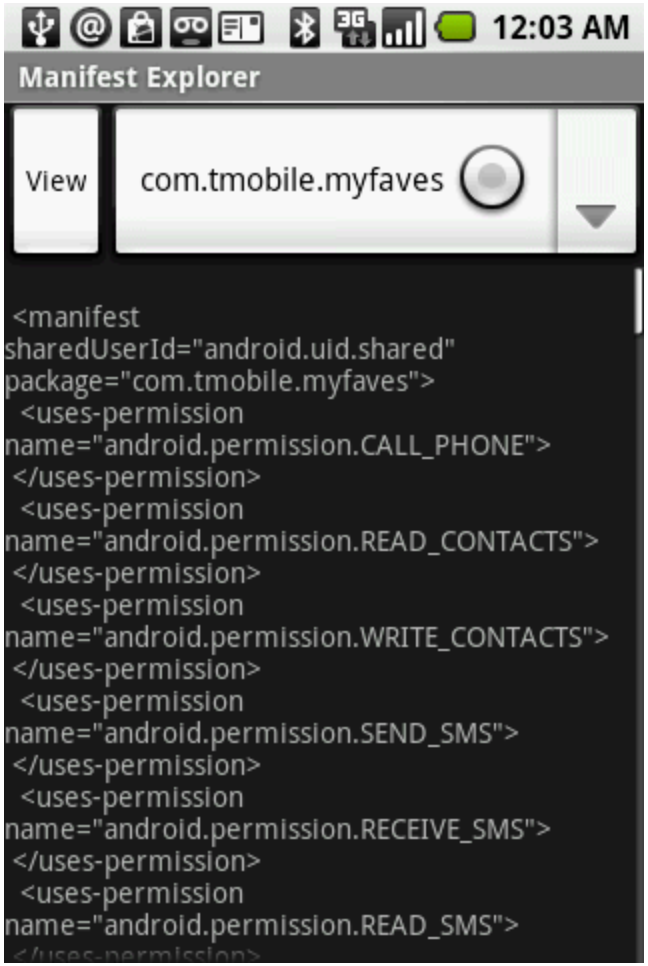
Manifest Explorer on Browser com.android.browser

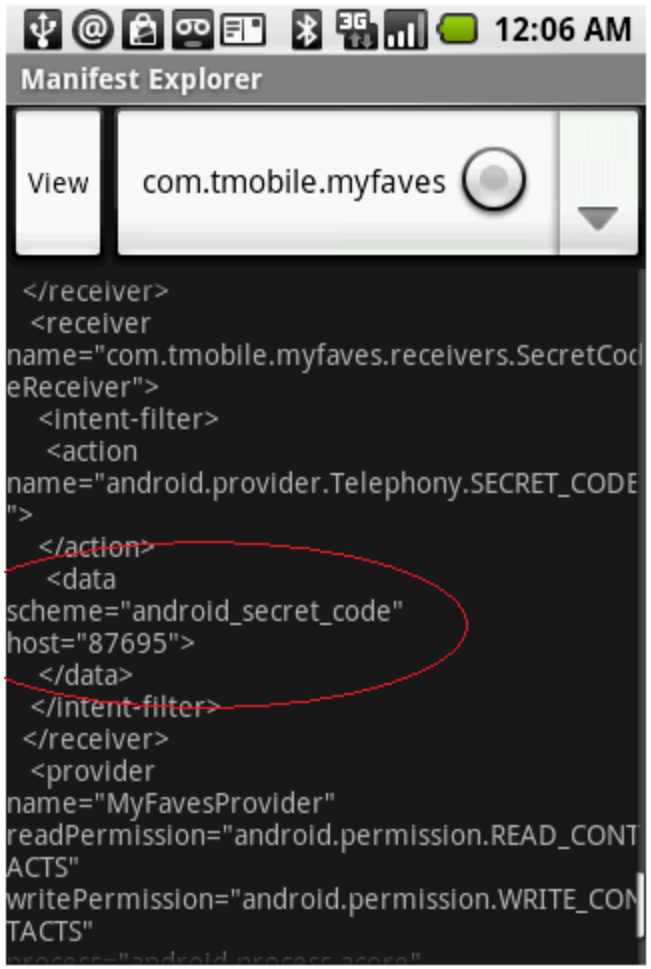


iSEC PARTNERS

Manifests and Manifest Explorer

"Contacts and myFaves storage" com.tmobile.myfaves

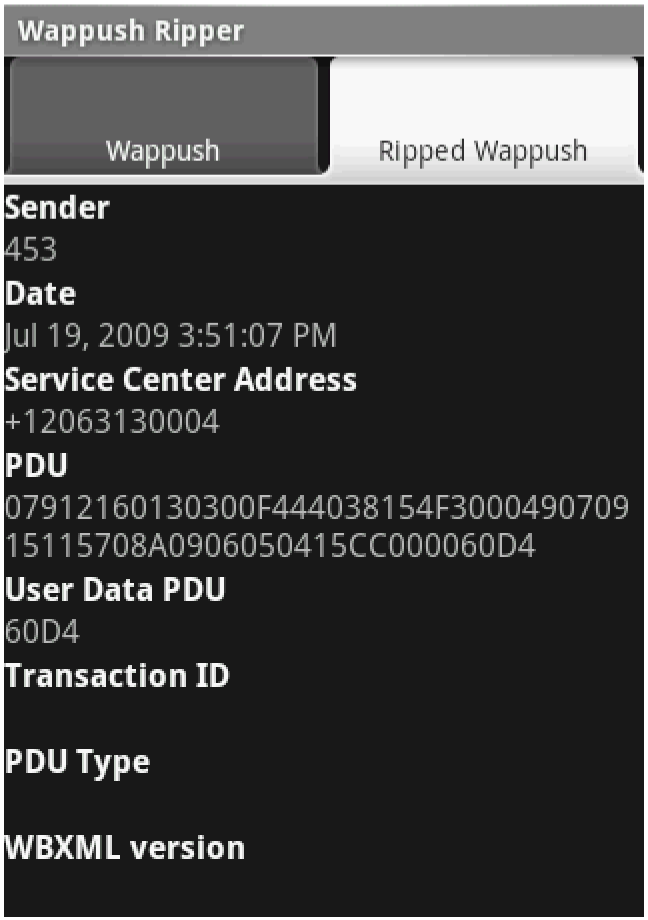




iSEC PARTNERS

What does this "secret code" do?

Got some weird WAPPUSH SMS / PDU



Selective logcat for ~ six seconds around entering the code:
03.792: INFO/MyFaves(26963): starting service with intent: Intent { comp={com.tmobile.myfaves/com.tmobile.myfaves.MyFavesService} (has extras) }
03.802: INFO/MyFaves(26963): handleMessage(4)
04.372: INFO/MyFaves(26963): sending msg: 163582790150134200010000000000000000000000000000000000000 0000000000000000000000000 to 453
06.732: INFO/MyFaves(26963): SMSStatusReceiver.onReceive(extras: Bundle[{id=100}]; resultCode: -1); action: sent
06.762: INFO/MyFaves(26963): starting service with intent: Intent { comp={com.tmobile.myfaves/com.tmobile.myfaves.MyFavesService} (has extras) }
06.762: INFO/MyFaves(26963): handleMessage(0)
06.832: INFO/ActivityManager(54): Stopping service: com.tmobile.myfaves/.MyFavesService
09.122: INFO/MyFaves(26963): queueInboundSMSMesssage: 05
09.152: INFO/MyFaves(26963): starting service with intent: Intent { comp={com.tmobile.myfaves/com.tmobile.myfaves.MyFavesService} (has extras) }
09.162: INFO/MyFaves(26963): handleMessage(6)

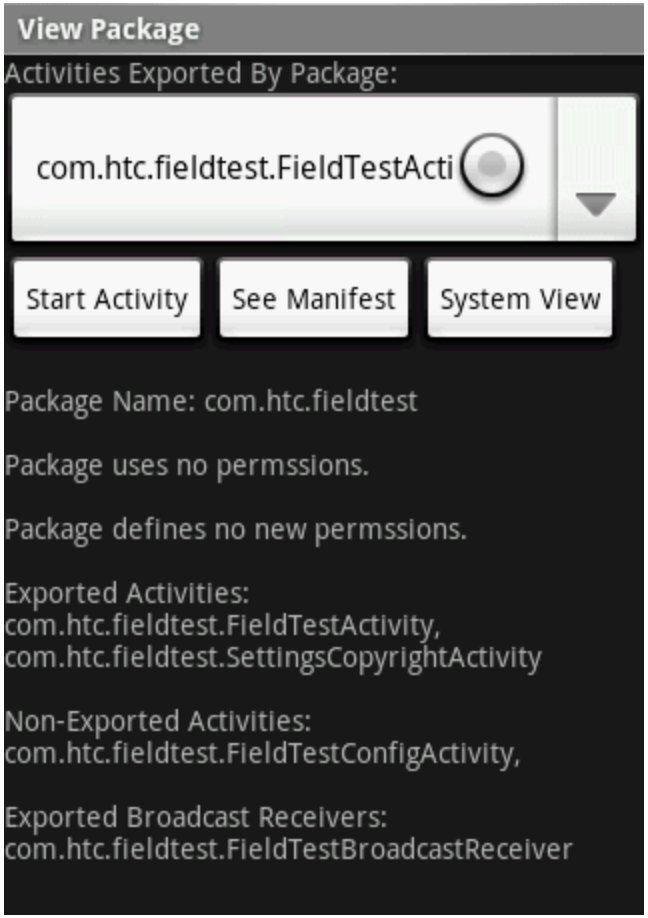
iSEC PARTNERS

Package Play

- Shows you installed packages:
  - Easy way to start exported Activities
  - Shows defined and used permissions
  - Shows activities, services, receivers, providers and instrumentation, their export and permission status
  - Switches to Manifest Explorer or the Setting's applications view of the application.

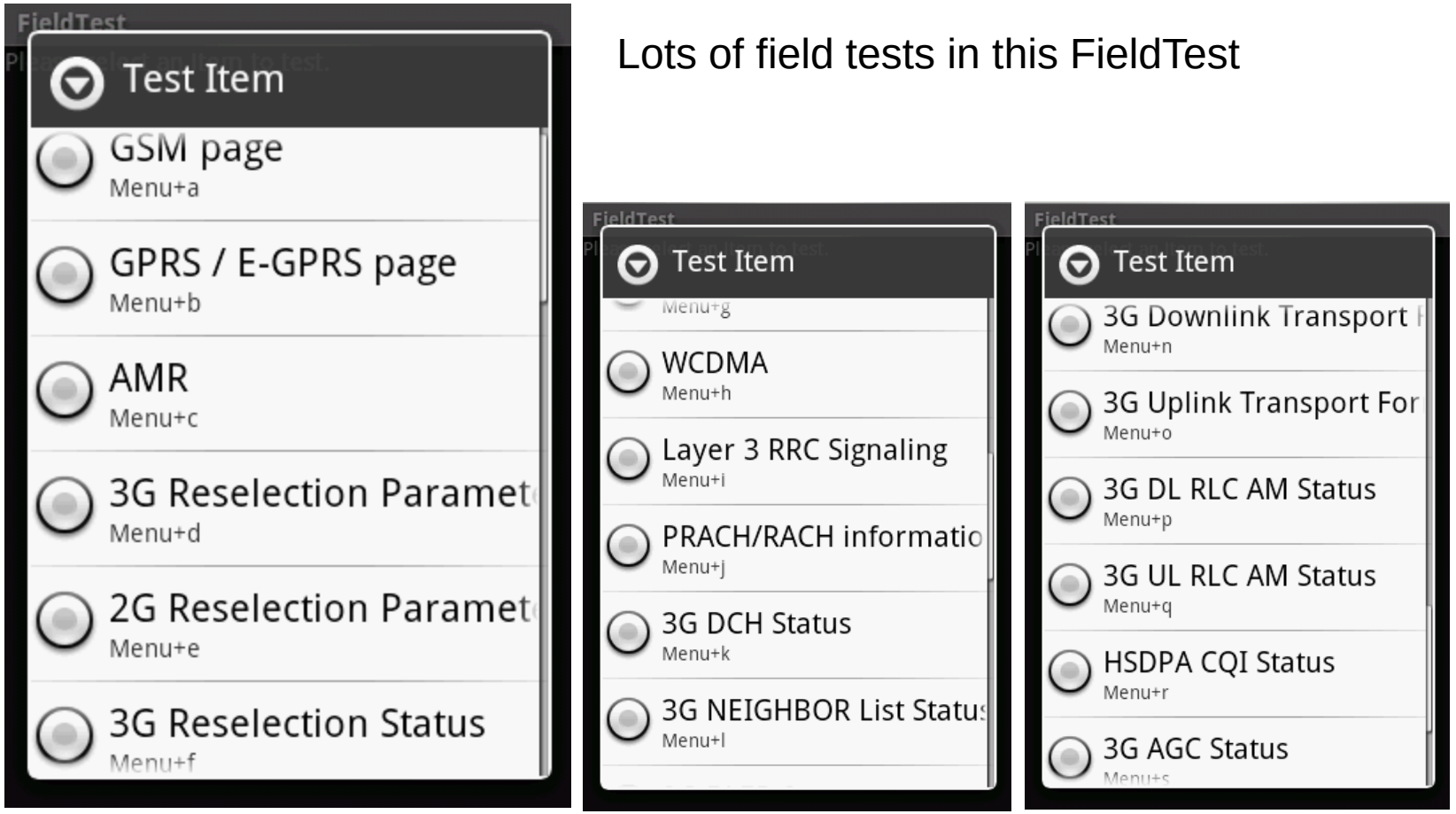
iSEC PARTNERS

Package Play



iSEC PARTNERS

Playing with "FieldTest"



Lots of field tests in this FieldTest

Playing with "FieldTest"

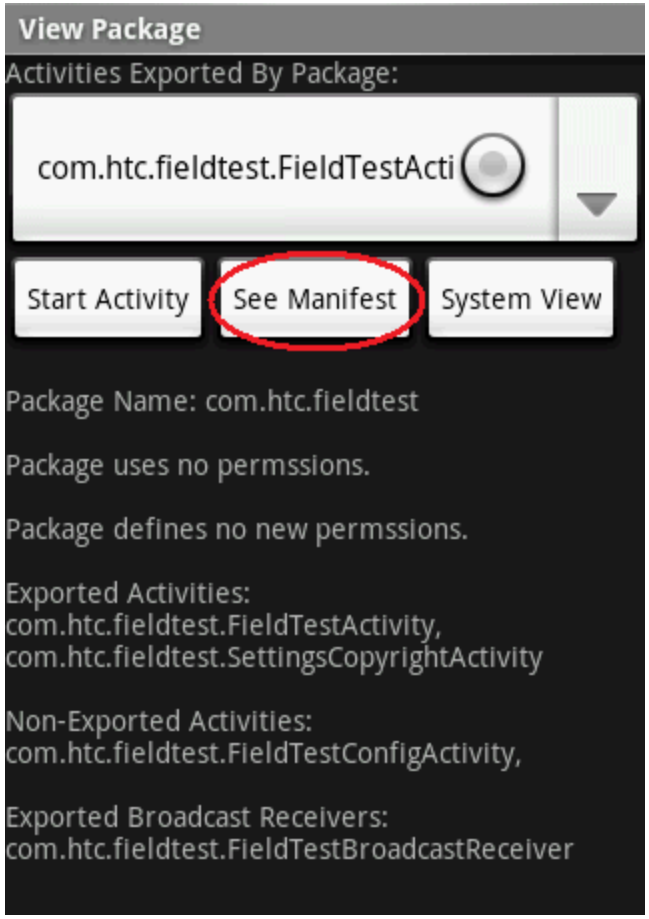
GSM page	
ARFCN	000
LAC	9e31
RAC	01
MNC/MCC	31260
RSSI	16
Ncell Info1	0 -99 dBm
Ncell Info2	0 -99 dBm
Ncell Info3	0 -99 dBm
Ncell Info4	0 -99 dBm
Ncell Info5	0 -99 dBm
Ncell Info6	0 -99 dBm
RX Quality	16
Frequent Hopping	Not active
Last registered network	31260
TMSI	549ea85d
Periodic Location Update Value	1530 (min)
BAND	N/A
Channel In Use	N/A
RSSI 1	0 dBm
Last cell release cause	255

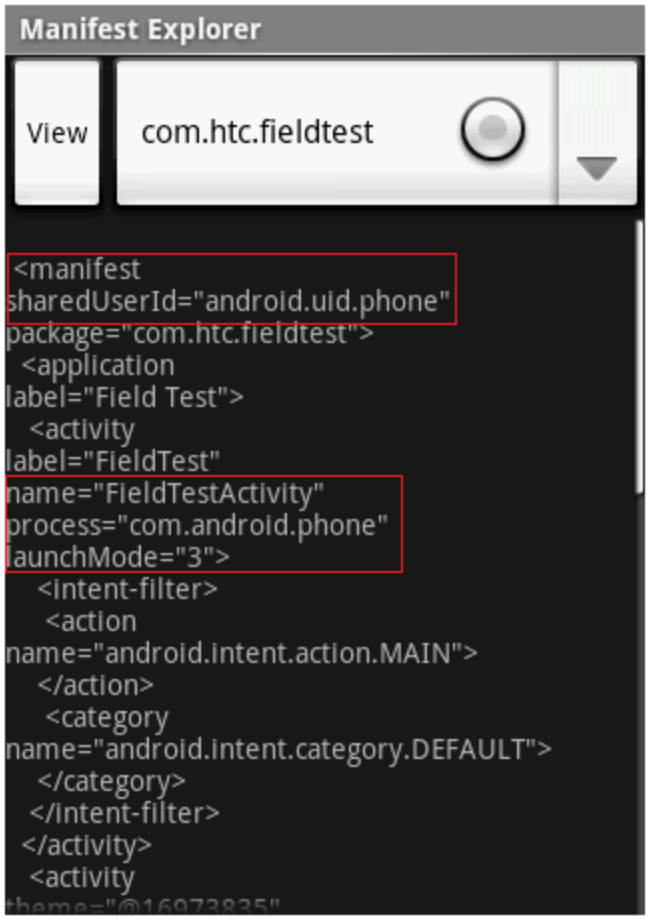
3G Reselection Status	
ServingPSC	
ServingUARFCN	
ServingAGC	-64 dBm
ServingECNO_M_Value	000
ServingECNO_N_Value	00
ServingECNO	
RealECNO	<N/A
Num3GCell	
RankPSC_1	39
RankUARFCN_1	208
RacnkRSCP_1	-84 dBm
RankCalRankRSCP_1	-82
RankECNO_1	-12 dB
RankCalRankECNO_1	-20
RankPSC_2	262
RankUARFCN_2	2087
RankRSCP_2	-103 dBm
RankCalRankRSCP_2	-32768
RankECNO_2	-31 dB
RankCalRankECNO_2	-32768

VERBOSE/FieldTestActivity(100): FT mode enabled
VERBOSE/FieldTestActivity(100): Response <- RIL: Query FT mode
VERBOSE/FieldTestActivity(100): Start test request
VERBOSE/FieldTestActivity(100): Request -> RIL
VERBOSE/FieldTestActivity(100): Response <- RIL

iSEC PARTNERS

Package Play – Program Rights





ps says:

radio 100 31 152088 17524 ffffffff afe0c824 S com.android.phone

iSEC PARTNERS

Intent Sniffer

- Monitoring of runtime routed broadcasts Intents
  - Doesn't see explicit broadcast Intents
  - Defaults to (mostly) unprivileged broadcasts
- Option to see recent tasks Intents (GET_TASKS)
  - When started, Activity's intents are visible!
- Can dynamically update Actions & Categories
- Types are wild-carded
- Schemes are hard-coded

iSEC PARTNERS

Intent Sniffer

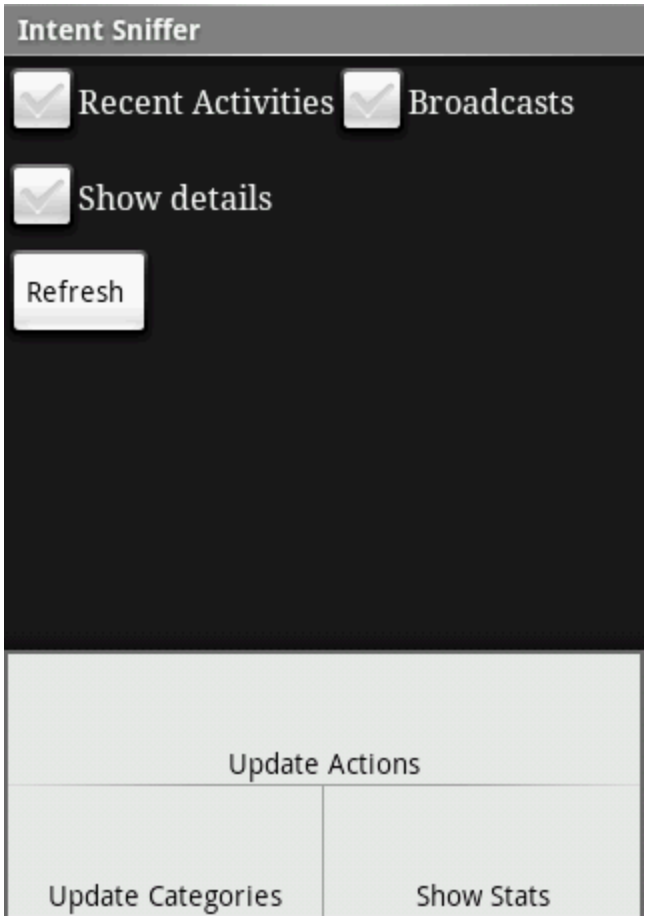
- GET_TASKS
  - Sees other Activity's startup Intents:

```
Intent { flags=0x30800000
comp={com.google.android.systemupdater/com.google.android.systemupd
ater.SystemUpdateInstallDialog} (has extras) } extras {firstPrompt -
(132810)
updateFile - (/cache/signed-kila-ota-150275.53dde318.zip)
} from recent tasks
```

  - File can't be viewed before it is executed ☹
  - Isn't in the open code
  - Perhaps for "Google Experience" devices only?

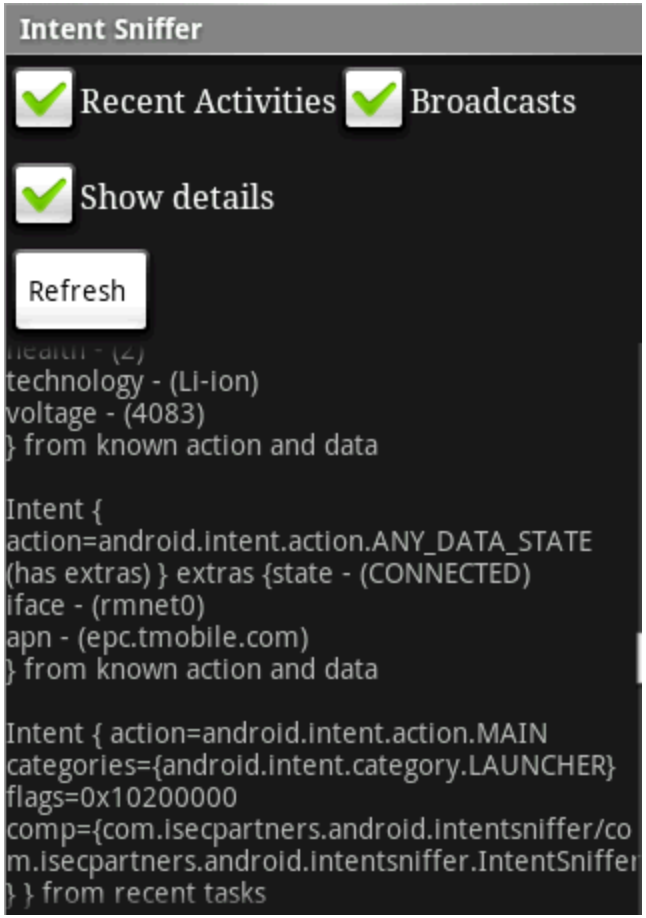
iSEC PARTNERS

Intent Sniffer



Intent Sniffer

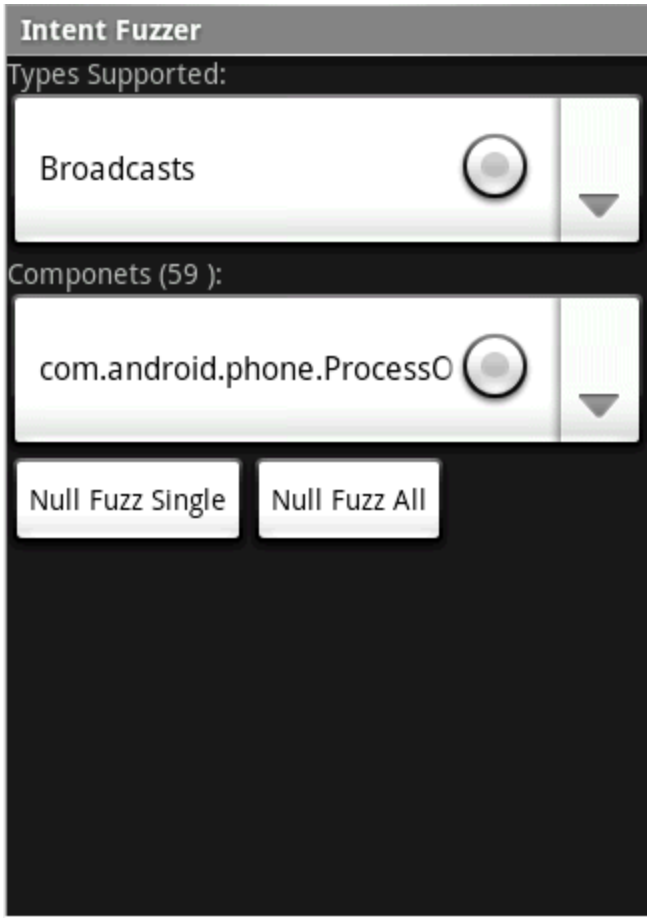
- Intents source listed at the bottom of each.
- Intents with components obviously come from recent tasks

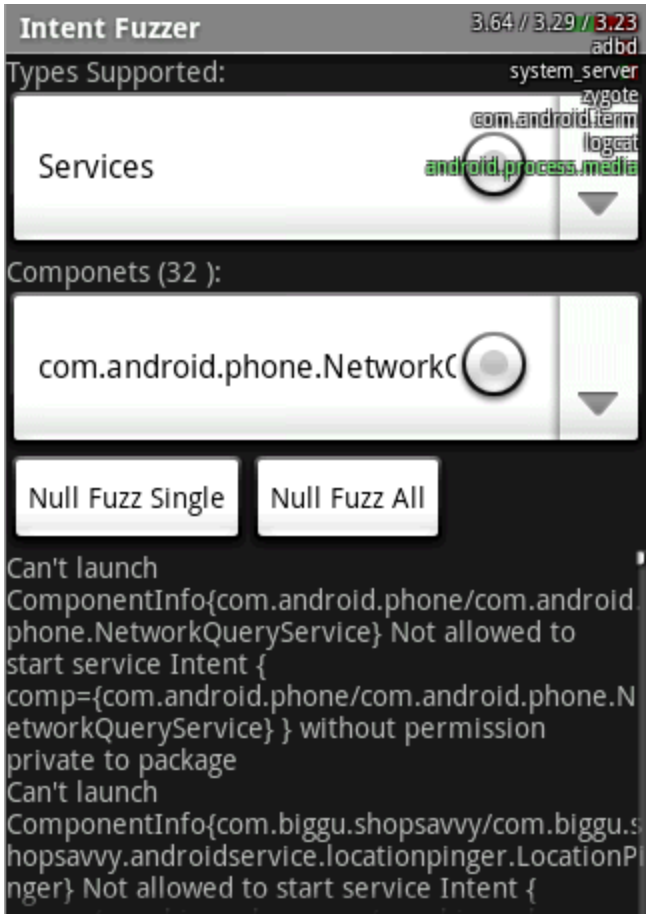


iSEC PARTNERS

Intent Fuzzing

- Fuzzing can be fun, java minimizes impacts
- Often finds crashing bugs or performance issues





iSEC PARTNERS

Concluding Thoughts

Hidden packages, root & proprietary bits

Common problems

Possible aardvark raffle

Questions

Android's Private Parts

- Platforms need to change internals to evolve
  - App developers should avoid the shakiest bits
  - Security researchers don't

- We see this marker on classes, or individual methods

  @hide

```
/**
 * @hide Broadcast intent when the volume for a particular stream type changes.
 * Includes the stream and the new volume
 *
 * @see #EXTRA_VOLUME_STREAM_TYPE
 * @see #EXTRA_VOLUME_STREAM_VALUE
 */
 @SdkConstant(SdkConstantType.BROADCAST_INTENT_ACTION)
 public static final String VOLUME_CHANGED_ACTION = "android.media.VOLUME_CHANGED_ACTION";
```

  This is to help developers avoid mistakes

  NOT a security boundary, trivially bypassed

Root lockdown

Carriers or Manufacturers

- Locking down the phone means securing for – not against users. Don't pick a fight with customers.
- People with root won't upgrade & fix systems
- Schemes for maintaining root are dangerous

Market Enabler – little program to enable market

- Needs root to set system properties
- Only asks for "INTERNET" permission
- For this to work the Linux sandbox was defeated

```
// Getting Root ;)
process = Runtime.getRuntime().exec("su");
```

iSEC PARTNERS

Proprietary bits

- Radio firmware is private & highly privileged
- Many WiFi cards are similar – GPL purity combat
- Computer bios too
- Think about the phone switches on the backend
- Do you really know what's in the heart of your CPU
  - Do you even know what VPRO is?

Keep perspective & a disassembler

Search the net for platform documentation

iSEC PARTNERS

Common Problems

- Implicit vs. Explicit Intents
- Too many or few permissions
- Data source & destination
  - Who sent this broadcast
  - Who might be able to see this
- Trusting external storage (Fat-32 no security for you)
- Users with unpassworded setuid root shells, su, etc.
- Implementing non-standardized features
  - OTA updates, application distribution & update

iSEC PARTNERS

Special Thanks

- iSEC Partners, especially Chris Palmer
  - Thanks for all your help & feedback getting this ready
- Google's Android Team
  - They are awesome
  - Special thanks to: Rich Cannings, Dianne Hackborn, Brian Swetland, David Bort
- My clients who can't be named; but who help keep my mental hamster in shape.
  - Sorry I can't list you in a compressed o+r manifest

iSEC PARTNERS

Questions?

iSEC PARTNERS

Questions?

Incase you need some sample questions:

- What is Intent reflection?
- How would I secure a root shell for users of my distribution of Android?
- How do I spy on users, without being publicly humiliated like SS8 was in the United Arab Emirates?
- How do I stop someone naughty from sending my app an Intent?
- What's the deal code signing that doesn't require a trusted root?
- What's the parallel between the browser security model and the Android security model you mentioned?

iSEC PARTNERS

Thank you for coming!

Want a copy of the presentation/tool?

Email:

blackhat@isecpartners.com

…and get all the iSEC Partners BH USA 2009 presentations and tools
It is also be available on our web site: https://www.isecpartners.com.

Contact me about Android stuff at

AndroidSecurityPaper@isecpartners.com

or come introduce yourself

iSEC PARTNERS