Giving Mobile Security the Boot

Jonathan Levin

http://Technologeeks.com
Plan

- Android Boot Chain
- iOS Boot Chain
- TrustZone
- iOS & TrustZone
- Android & TrustZone
morpheus@Zepyhr$ whoami

(一点儿宣传 😊)

• 深入解析 Android
• Available (in Chinese!) End of 2016 – Including N
• http://NewAndroidBook.com/

Volume I (available)  Volume II (soo-N)
morpheus@Zephyr$ whoami

(一点儿宣传 😊)

• 深入解析Mac OS X & iOS操作系统

• Plenty of useful reversing tools
  – jtool
  – procexp
  – filemon

• But book terribly outdated!
Boot Chains of Trust
The Android Boot Sequence

- Exact flow varies with vendor, but can be generalized
- Components (except ROM) easily extracted from OTA

```
morpheus@Forge % imgtool Images/hammerhead-kot49h/bootloader-hammerhead-hхи11k.img
Boot loader detected
6 images detected, starting at offset 0x200. Size: 2568028 bytes
Image: 0 Size: 310836 bytes sb11  # Secondary Boot Loader, stage 1
Image: 1 Size: 285848 bytes tz  # TrustZone image
Image: 2 Size: 156040 bytes rpm  # Resource Power Mgmt
Image: 3 Size: 261716 bytes aboot  # Application Boot Loader
Image: 4 Size: 18100 bytes sdi
Image: 5 Size: 1535488 bytes imgdata  # RLE565 graphics used by boot loader
```
Android Boot: The BootROM

- Very specific per chipset manufacturer
- Not much is known about ROMs
- But not really relevant for our discussion, either
- Contain a hard coded public key (公钥) of manufacturer
Android Boot: SBL

• Vendor specific, but usually same operation:
  – Initialize subsystems (baseband, DSP, GPU, TZ)
  – Locate Android Boot

• Signed with private key(与私钥) of manufacturer
  – Signature is first link in chain of trust
  – May contain another public key of manufacturer or same.
Android Boot: ABoot

• Commonly* based off of open source Little Kernel
  – May be customized by vendor

• Supports FASTboot or other (e.g. ODIN) for flashing

• May or may not be unlockable (解鎖)
  – If unlocked:
    • Effaces data (to ensure user data won’t be compromised)
    • Breaks chain of trust (any kernel can be loaded)
    • Usually blows a Qfuse to indicate void warranty

* - Samsung, others have custom loaders

(C) 2016 Jonathan Levin & Technologeeks.com - Share freely, but please cite source!
Android Boot: Kernel + initrd

- Kernel is same ol’ Linux, but compiled for ARMv7/v8
- InitRD (初始化 RAM 磁盘) contains root (/) file system
  - /init daemon and other vital daemons
  - /init.rc configuration files
  - SEPolicy (SELinux的策略) which is enforced on device

- Crucial components for security so bundled together
  - Kernel + initrd is in one partition
  - Aboot verifies hash of partition before loading (if locked)

(C) 2016 Jonathan Levin & Technologeeks.com - Share freely, but please cite source!
Android Boot: DM-Verity

- Extends boot-chain by taking hash of /system
  - /system is read-only, so in theory should not be modified

- /system mounted through device mapper, as dm# device
- All I/O flows through device mapper, verifies hashes
  - Incorrect hash causes I/O error

- In practice nice idea, but utterly useless (不中用)
  - System-less root methods root but leave /system untouched.

(C) 2016 Jonathan Levin & Technologeeks.com - Share freely, but please cite source!
iOS Boot Sequence

• All boot components are encrypted
  – 32-bit: IMG3 64-bit: IMG4 (DER)

• All boot components are validated
  – Slightest error sends device to recovery (and forced upgrade!)

• 64-bit boot sequence still not broken*

• 64-bit systems bolstered with Kernel Patch Protection (9.0)
  – Feeble (but valiant) attempt to prevent runtime kernel patches

* - no public ROM/LLB/iBoot exploit presently known

(C) 2016 Jonathan Levin & Technologeeks.com - Share freely, but please cite source!
iOS Boot: The BootROM

- Read only memory component, “Apple SecureROM”
  - Contains hard-coded public key of Apple
- Wasn’t that secure in A4 devices (<= iPhone 4)
  - Limera1n allows bypass and full ROM dump
- Considerably better in A5 and later devices (>=4S)
- Virtually unknown in A7 and later devices (5S+, 64-bit)
  - Theoretically dumpable via JTAG

(C) 2016 Jonathan Levin & Technologeeks.com - Share freely, but please cite source!
iOS Boot: iLLB

- Low Level Bootloader
- Functions as stage 1 bootloader
- Provides basic USB functionality (e.g. DFU)
- Loads iBoot
iOS Boot: iBoot

- Main component of boot process
- Initializes all sub components
- Spawns several threads (poweroff, idle, USB, ...)
- Provides full USB functionality, HFS+, and more
- 64-bit version also communicates with SEP
- Locates and loads kernelcache, but refuses arguments
- Logs to serial console, then turns it off
- Turns GID access off
- Validates SHSH (< iOS5) or APTicket (>=iOS5)
iOS Boot: KernelCache

- /System/Library/Caches/com.apple.kernelcaches/

- Prelinks all kernel extensions (内核, 包括所有的扩展)

- Kernel extension loading otherwise disabled

- Benefits:
  - Speed (prelinking)
  - Security (kernel + kexts authenticated, no other kexts allowed)
Validating components: SHSH

- User updates/restores device
- iBoot gets image (IPSW), parses it, generates request

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApBoardID</td>
<td>From IPSW</td>
</tr>
<tr>
<td>ApChipID</td>
<td>From Device</td>
</tr>
<tr>
<td>ApECID</td>
<td>Exclusive Chip ID</td>
</tr>
<tr>
<td>ApProductionMode</td>
<td>true (unfortunately)</td>
</tr>
<tr>
<td>ApSecurityDomain</td>
<td>From IPSW</td>
</tr>
<tr>
<td>UDID</td>
<td>Unique Device Identifier</td>
</tr>
<tr>
<td>HostPlatformInfo</td>
<td>iTunes host OS identifier</td>
</tr>
<tr>
<td>Locality</td>
<td>en_US, zh_CN, etc..</td>
</tr>
<tr>
<td>VersionInfo</td>
<td>libauthinstall-a.b.c.d.e</td>
</tr>
</tbody>
</table>

- Apple signs with their private key.
- iBoot stores in NAND firmware partition SCAB container

Validating components: SHSH

- **Serious vulnerability: Replay**
  - Protocol is plaintext, so easy to capture blobs
  - Store safely for a rainy day
  - When you want to bypass, fake gs.apple.com (e.g. /etc/hosts)

- **Widely used before iOS 5 for downgrades (降級)**
  - iFaith
  - Saurik’s cydia server (built-in functionality)
  - TinyUmbrella (TSS Server)
Validating components: APTicket

- Same as SHSH, but image now contains ApNonce

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApBoardID</td>
<td>From IPSW</td>
</tr>
<tr>
<td>ApChipID</td>
<td>From Device</td>
</tr>
<tr>
<td>ApECID</td>
<td>Exclusive Chip ID</td>
</tr>
<tr>
<td>ApNonce</td>
<td>Random From iBoot(!)</td>
</tr>
<tr>
<td>ApProductionMode</td>
<td>true (unfortunately)</td>
</tr>
<tr>
<td>ApSecurityDomain</td>
<td>From IPSW</td>
</tr>
<tr>
<td>ApTicket</td>
<td>true</td>
</tr>
<tr>
<td>UDID</td>
<td>Unique Device Identifier</td>
</tr>
<tr>
<td>HostPlatformInfo</td>
<td>iTunes host OS identifier</td>
</tr>
<tr>
<td>Locality</td>
<td>en_US, zh_CN, etc..</td>
</tr>
<tr>
<td>VersionInfo</td>
<td>libauthinstall-a.b.c.d.e</td>
</tr>
</tbody>
</table>

- iBoot stores in firmware partition and /System/Library/Caches
  - Nonce prevents replay unless iBoot can be pwned (e.g. Odysseus)
iOS 10b1: Think different

- For the first time, kernelcache is not encrypted
- Provides a first look at “missing pieces”
  - Jettisoned segments (e.g. KLD, __PRELINK_INFO)
  - KPP: Kernel Patch Protection

after the issue gained wider attention, the company released a statement Wednesday saying it had intentionally left the kernel unencrypted—but not for security reasons.

"By unencrypting it we’re able to optimize the operating system’s performance without compromising security," an Apple spokesman said. He declined to elaborate on how exactly the performance of iOS would be improved.
iOS 10b1: Think different

- Mistake? Intentional? Only Cupertino knows... But I say:
  
  废话...

  after the issue gained wider attention, the company released a statement Wednesday saying it had intentionally left the kernel unencrypted—but not for security reasons.

  "By unencrypting it we’re able to optimize the operating system’s performance without compromising security," an Apple spokesman said. He declined to elaborate on how exactly the performance of iOS would be improved.

* Edit – Apple apparently took this seriously and did open the 32-bit chain (but NOT 64) in 10b2.
TrustZone & ELx
TrustZone 技

- Hardware support for a trusted execution environment
  - Provides a separate "secure world" 安全世界
    - Self-contained operating system
    - Isolated from "non-secure world"

- In AArch64, integrates well with Exception Levels(例外層級)
  - EL3 only exists in the secure world
  - EL2 (hypervisor) not applicable in secure world.
Trust Zone Architecture (Aarch32)

Source: ARM documentation
Android uses of TrustZone

- Cryptographic hardware backing (keystore, gatekeeper)
  - Key generation, storage and validation are all in secure world
  - Public keys accessible in non-secure world

- DRM (数字版权管理) - special case crypto hardware backing

- Hardware backed entropy
  - PRNG (随机数发生器) code

- 安全 NFC 通信通道 (Android Pay)

- Kernel and boot chain integrity
Samsung uses of TrustZone

- TrustZone is a fundamental substrate for KNOX
  - Trusted Integrity Measurement Attestation (TIMA) provides

- Client Certificate Management (CCM)
  - Extends keystore by hardware backing

- Periodic Kernel Measurement (PKM) 周期内核测量
  - Similar to iOS’s KPP – periodically checks kernel page hashes
    » 会定期检查内核校验和

- Realtime Kernel Protection (RKP) 实时内核保护
  - Intercepts events from kernel using traps to secure monitor (SMC)
  - 捕获任何恶意活动

(C) 2016 Jonathan Levin & Technologeeks.com - Share freely, but please cite source!
iOS Uses of TrustZone

• 32-bit: Apparently, none(?)
  – No SMC instructions in decrypted kernelcache

• 64-bit: KPP
  – Long thought (mistakenly) to have been in Secure Enclave
  – iLLB/iBoot also physically separated from kernel memory
Implementation (AArch32)

- Implemented by a Secure Configuration Register (SCR)

- NS = 0: 系统处于安全状态. NS =1 系统处于非安全状态

- SCR is co-processor CP15,c1

- Cannot be accessed in non-secure world:
  - Need SMC特殊指令

- MMU enforces memory separation between worlds

- Interrupts (IRQ/FIQ) can be handled by secure world
Entering TrustZone (AArch32)

- SMC to TrustZone is like SVC/SWI to supervisor mode

SMC指令

- Control transferred to a “monitor vector” in secure world

SMC指令

中断

快速中断

中止

Data Abort

Prefetch Abort

IRQ

FIQ

monitor的异常向量表

0x1C
Voluntary Transition: SMC

- SMC特殊指令 only valid while in supervisor mode
  - (i.e. requires the OS to be in kernel (内核) mode)
Exception Handling (AArch64)

架构定义了四个例外层级:

- **EL0**
  - App\_0,0
  - App\_0,n

- **EL1**
  - OS\_0 Kernel (内核)
  - OS\_1 Kernel (内核)
  - Hypervisor (管理程序)

- **EL2**
  - Secure Kernel (内核)

- **EL3**
  - Secure Monitor (我不知道怎么说是)

(C) 2016 Jonathan Levin & Technologeeks.com - Share freely, but please cite source!
Setting up Trustzone

• 32-bit:
  – CPU boots into secure world (NS=0)
  – Loader/kernel sets up monitor vector (SMC, IRQ or FIQ entries)
  – Sets up SCR NS=1 and “drops” to Normal World

• 64-bit:
  – CPU boots into EL3
  – Secure Monitor sets up VBAR_Elx (SError, IRQ or FIQ entries)
  – Drops to EL2 (Hypervisor, 管理程序) or EL1 (kernel, 内核)
A$Arch64 Exception Handling

- **Current EL, SP0**
  - 0x000: Synchronous
  - 0x200: Synchronous

- **Current EL, SPSel**
  - 0x400: Synchronous

- **Lower EL, AArch64**
  - 0x600: Synchronous

- **Lower EL, AArch32**

(C) 2016 Jonathan Levin & Technologeeks.com - Share freely, but please cite source!
## Case Study: KPP

<table>
<thead>
<tr>
<th>Address (0x000)</th>
<th>Synchronous</th>
<th>IRQ/vIRQ</th>
<th>FIQ/vFIQ</th>
<th>SError/vSError</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Synchronous</strong></td>
<td>IRQ/vIRQ</td>
<td>FIQ/vFIQ</td>
<td>SError/vSError</td>
<td></td>
</tr>
<tr>
<td><strong>Synchronous</strong></td>
<td>IRQ/vIRQ</td>
<td>FIQ/vFIQ</td>
<td>SError/vSError</td>
<td></td>
</tr>
<tr>
<td><strong>Synchronous</strong></td>
<td>IRQ/vIRQ</td>
<td>FIQ/vFIQ</td>
<td>SError/vSError</td>
<td></td>
</tr>
<tr>
<td><strong>Synchronous</strong></td>
<td>IRQ/vIRQ</td>
<td>FIQ/vFIQ</td>
<td>SError/vSError</td>
<td></td>
</tr>
</tbody>
</table>

### Current EL, SP0
- 0x000
- 0x200

### Current EL, SPSel
- 0x400

### Lower EL, AArch64
- 0x600

### Current EL, AArch32
- 0x000

---

**Case Study:** KPP

The diagram illustrates theelaboration of various statuses and addresses, reflecting their corresponding attributes and functionalities within the context of AArch64 and AArch32 architectures. These statuses are critical for understanding the execution流程 and deployment strategies in parallel computing environments.
## Case Study: KPP

<table>
<thead>
<tr>
<th>Address</th>
<th>EL, Mode</th>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000</td>
<td>Current EL, SP0</td>
<td>Synchronous</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IRQ/vIRQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FIQ/vFIQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SError/vSError</td>
</tr>
<tr>
<td>0x200</td>
<td>Current EL, SPSel</td>
<td>Synchronous</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IRQ/vIRQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FIQ/vFIQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SError/vSError</td>
</tr>
<tr>
<td>0x400</td>
<td>Lower EL, AArch64</td>
<td>Synchronous</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IRQ/vIRQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FIQ/vFIQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SError/vSError</td>
</tr>
<tr>
<td>0x600</td>
<td>Lower EL, AArch32</td>
<td>Synchronous</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IRQ/vIRQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FIQ/vFIQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SError/vSError</td>
</tr>
</tbody>
</table>

- **Check TTBR_EL1 access**
- **Check for known SMC codes**
- **Default**
Case Study: KPP

Current EL, SP0

Synchronous
IRQ/vIRQ
FIQ/vFIQ
SError/vSError

Current EL, SPSel

Synchronous
IRQ/vIRQ
FIQ/vFIQ
SError/vSError

Lower EL, AArch64

Synchronous
IRQ/vIRQ
FIQ/vFIQ
SError/vSError

Lower EL, AArch32

Synchronous
IRQ/vIRQ
FIQ/vFIQ
SError/vSError

(C) 2016 Jonathan Levin & Technologeeks.com - Share freely, but please cite source!
morpheus@zephyr (~/.//ios10)$ jtool -opcodes -d __TEXT_EXEC.__text xnu.3705.j99a| grep SMC
Opened companion File: ./xnu.3705.j99a.ARM64.33A2E481-EF0F-3779-8C96-360114BB824A
Loading symbols...
Disassembling from file offset 0x78000, Address 0xffffffff00747c000
fffffffff007483b0c  d4000223  SMC  #17
#
# Add symbol to companion file, for easy reference later:
morpheus@zephyr (~/.//ios10)$ echo 0xffffffff007483b0c:_smc >> ./xnu.3705.j99a.*
#
# Find All calls to SMC
morpheus@zephyr (~/.//ios10)$ jtool -d __TEXT_EXEC.__text xnu.3705.j99a| grep -B 4 " _smc"
Opened companion File: ./xnu.3705.j99a.ARM64.33A2E481-EF0F-3779-8C96-360114BB824A
Loading symbols...
Disassembling from file offset 0x78000, Address 0xffffffff00747c000
fffffffff0074c002c  MOVZ  W0, 0x801  ; ->R0 = 0x801
fffffffff0074c0030  MOVZ  X1, 0x0     ; ->R1 = 0x0
fffffffff0074c0034  MOVZ  X2, 0x0     ; ->R2 = 0x0
fffffffff0074c0038  MOVZ  X3, 0x0     ; ->R3 = 0x0
fffffffff0074c003c  BL    _smc       ; 0xffffffff007483b0c
...
KPP Checks

On entry:

• Iterates over Kernel, all kexts
• Checks all __TEXT segments, and __const sections
• Takes checksums, kept in EL3
• Checksums verified during checks
KPP Weakness (patched in 9.2)

- Plenty of pointers in __DATA sections not protected

- Example: AMFI MACF hooks
  - Pangu 9 patches MACF hooks
  - Moved in 9.2 to __DATA.__const

- Maybe there’s still more pointers?
  - Ask organizers of conference 😊
iOS 10 changes

- XNU Mach-O binary re-segmented
  - This means that “leaked” KPP no longer works
- Checks for hard coded __DATA.__PRELINK_INFO, ...

morpheus@zephyr (~/..../ios10)$ jtool -d kpp | grep "
Opened companion File: ./kpp.ARM64.35324088-001A-383E-976E-C4EBD990F3A8
Loading symbols...
Disassembling from file offset 0x1000, Address 0x4100001000
  410000429c  ADR  x22, #12662  
  41000042a8  ADR  x25, #12635  
  41000042c8  ADR  x1, #12570  
  41000042e0  ADR  x1, #12564  
  41000044fc  ADR  x23, #12280  
  410000451c  ADR  x1, #12140  
  4100004570  ADR  x1, #11976  
  4100004588  ADR  x0, #11952  
  410000459c  ADR  x1, #11997  
  4100004638  ADR  x25, #11723  
  41000046bc  ADR  x1, #11565  
  41000048d0  ADR  x1, #11066  
  41000049ac  ADR  x1, #10846  

(C) 2016 Jonathan Levin & Technologeeks.com - Share freely, but please cite source!
iOS 10 changes

- Decoy? Another “mistake”? *shrug*
- Implementation is very likely now part of iBoot, (EL3 inaccessible)

(C) 2016 Jonathan Levin & Technologeeks.com - Share freely, but please cite source!
Android & TrustZone

• BootROM/SBL loads TZ image of “secure OS”
  – Usually in a TZ partition on flash
  – Backup (identical) usually also present

• Trustzone kernel usually an ELF image
  – Actual implementation is vendor-specific
  – Examples: Nvidia, Qualcomm

• Linux Kernel communicates with TZ kernel via driver
• Driver exports character device to user mode
• (Usually) dedicated daemon to communicate with kernel
Android & TrustZone

com.android.application

keystore

HAL

gatekeeper

HAL

TZ Daemon (e.g. qseecomd)

SVC指令

Kernel

TZ Driver

SMC指令

TZ OS

3rd Party
Android
Vendor
Linux

(C) 2016 Jonathan Levin & Technologeeks.com - Share freely, but please cite source!
Android & TrustZone: examples

- NVidia (Nexus 9):

```
root@flounder:~ # ls -Ll /dev/block/platform/pciact/pciact.3/block
brw------- 1 root root 259, 13 Nov 30 23:26 APP -> ..29 /system
brw------- 1 root root 259, 14 Nov 30 23:26 CAC -> ..30 /cache
brw------- 1 root root 259, 7 Nov 30 23:26 CDR -> ..23
brw------- 1 root root 259, 4 Nov 30 23:26 DIA -> ..20
brw------- 1 root root 179, 5 Nov 30 23:26 DTB -> ..5 (normally) Device Tree (but empty)
brw------- 1 root root 259, 5 Nov 30 23:26 ELF1 -> ..21
brw------- 1 root root 259, 6 Nov 30 23:26 ELF2 -> ..22
brw------- 1 root root 179, 3 Nov 30 23:26 EKS -> ..1
brw------- 1 root root 179, 11 Nov 30 23:26 EXT -> ..11
brw------- 1 root root 179, 12 Nov 30 23:26 FST -> ..12
brw------- 1 root root 259, 17 Nov 30 23:26 GPT -> ..33 GUID Partition Table (backup)
brw------- 1 root root 179, 1 Nov 30 23:26 KEY -> ..1
brw------- 1 root root 259, 0 Nov 30 23:26 LNX -> ..16 boot.img (with HTC wrap)
brw------- 1 root root 259, 9 Dec 1 01:25 MD1 -> ..25
brw------- 1 root root 259, 10 Nov 30 23:26 MD2 -> ..26
brw------- 1 root root 259, 2 Nov 30 23:26 MPG -> ..10 Manufacturing Data
brw------- 1 root root 259, 1 Nov 30 23:26 MSC -> ..17 Misc
brw------- 1 root root 179, 10 Nov 30 23:26 NCT -> ..10
brw------- 1 root root 259, 12 Nov 30 23:26 OTA -> ..28 OTA Updates
brw------- 1 root root 179, 14 Nov 30 23:26 PG1 -> ..14
brw------- 1 system system 259, 11 Dec 5 01:04 PST -> ..27 Persistent
brw------- 1 system system 179, 8 Nov 30 23:26 RCA -> ..8
brw------- 1 root root 179, 6 Nov 30 23:26 RV1 -> ..6?
brw------- 1 root root 179, 13 Nov 30 23:26 RV2 -> ..13
brw------- 1 root root 259, 16 Nov 30 23:26 RV3 -> ..32
brw------- 1 root root 259, 3 Nov 30 23:26 SER -> ..19
brw------- 1 root root 259, 15 Nov 30 23:26 SOS -> ..12 recovery.img (cute :-)
brw------- 1 root root 179, 9 Nov 30 23:26 SPF -> ..9
brw------- 1 root root 179, 2 Nov 30 23:26 TOS -> ..2 ARM TrustZone
```

(C) 2016 Jonathan Levin & Technologeeks.com - Share freely, but please cite source!
Android & TrustZone: Samsung

```
root@s6# ls -l dev/block/platform/15570000.ufs/by-name
1rwxrwxrwx root  root  2016-05-27 08:53 BOOT -> /dev/block/sda5
1rwxrwxrwx root  root  2016-05-27 08:53 BOTA0 -> /dev/block/sda1
1rwxrwxrwx root  root  2016-05-27 08:53 BOTA1 -> /dev/block/sda2
1rwxrwxrwx root  root  2016-05-27 08:53 CACHE -> /dev/block/sda16
1rwxrwxrwx root  root  2016-05-27 08:53 DNT -> /dev/block/sda10
1rwxrwxrwx root  root  2016-05-27 08:53 EFS -> /dev/block/sda3
1rwxrwxrwx root  root  2016-05-27 08:53 HIDDEN -> /dev/block/sda17
1rwxrwxrwx root  root  2016-05-27 08:53 OTA -> /dev/block/sda7
1rwxrwxrwx root  root  2016-05-27 08:53 PARAM -> /dev/block/sda4
1rwxrwxrwx root  root  2016-05-27 08:53 PERSDATA -> /dev/block/sda13
1rwxrwxrwx root  root  2016-05-27 08:53 PERSISTENT -> /dev/block/sda11
1rwxrwxrwx root  root  2016-05-27 08:53 RADIO -> /dev/block/sda8
1rwxrwxrwx root  root  2016-05-27 08:53 RECOVERY -> /dev/block/sda6
1rwxrwxrwx root  root  2016-05-27 08:53 SBFS -> /dev/block/sda14
1rwxrwxrwx root  root  2016-05-27 08:53 STEADY -> /dev/block/sda12
1rwxrwxrwx root  root  2016-05-27 08:53 SYSTEM -> /dev/block/sda15
1rwxrwxrwx root  root  2016-05-27 08:53 TOMBSTONES -> /dev/block/sda9
1rwxrwxrwx root  root  2016-05-27 08:53 USERDATA -> /dev/block/sda18

root@s6# cat partitions | grep -v sda
major minor #blocks name
 7      0    32768 loop0
 8     16   4096  sdb  # Boot loader
 8     32   4096  sdc  # CryptoManager
253      0  2097152 vnswap0
```

(C) 2016 Jonathan Levin & Technologeeks.com - Share freely, but please cite source!
Reversing

• From Secure World: (安全世界)
  – If you can get TZ (or iBoot 😊) image, start at VBAR_EL3
  – Find SMC/ handler (Synchronous)
  – Find IRQ/FIQ handlers

• From Non-Secure World: (非安全世界)
  – Get kernel or bootloader
  – disarm and look for SMC calls
# disarm will automatically find strings when used as arguments

root@s6# JCOLOR=1 disarm /dev/sdb | less -R

```
...
0x0003fac4  0xd00002e0  ADRP X0, 94    ; X0 = 0x9d000
0x0003fac8  0x9112e000  ADD X0, X0, #1208 ; X0 = X0 + 0x4b8 = 0x9d4b8
0x0003facc  0x94001461  BL 0x44c50 ; = 0x44c50(" This is a non-secure chip. Skip...")
...
```

# So now we know 03fac4 is called on non-secure chip.. Search back using "?0x3fac4"

# disarm will attempt to auto guess the arguments to SMC as well

```
0x0003f9f4  0x12801de0  MOVN X0, #239
0x0003f9f8  0x52800001  MOVZ W1, 0x0
0x0003f9fc  0x2a1403e2  MOV X2, X20     ; X2 = X20 (0xf7120)
0x0003fa00  0xa9bf7bfd  STP X29, X30, 
0x0003fa04  0x44c50      BL 0x44c50      ; = 0x44c50(" This is a non-secure chip. Skip...")
0x0003fa08  0xa8c17bfd  LDP X29, X30, 
0x0003fa10  0x8c17bfd  LDP X29, X30, [SP],#16
0x0003fa14  0x54000580  B.EQ 0x3fac4
```

# can also grep SMC

```
...
0x0004f014  0xd4000003  SMC #0 ; (X0=0xc2001014, X1=0x0, X2=0xf7120..)
0x0004f044  0xd4000003  SMC #0 ; (X0=0xc2001014, X1=0x0, X2=0xf7120..)
0x0004f098  0xd4000003  SMC #0 ; (X0=0xc2001014, X1=0x0, X2=0xf7120..)
0x0004f0c8  0xd4000003  SMC #0 ; (X0=0xc2001014, X1=0x0, X2=0xf7120..)
```

Trusty

• Google’s attempt to standardize TEE Oses

• Used by Nvidia (+ LK)

• Supplies:
  – gatekeeper, keymaster, NVRAM modules
  – Kernel driver
  – LK base
  – Trusty OS

• https://android-review.googlesource.com/#/admin/projects/?filter=trusty
Linux Kernel Support

• Generic Trustzone driver integrated into 3.10

• Qualcomm ( msm) kernels have SCM driver
  – Secure Channel Manager
  – Creates a character device which qseecomd opens

• Driver issues SMC instructions, passes command buffers
  – Terrible buggy driver
  – Terrible buggy daemon
    • Amazing exploit and explanation – Masterful hack, and a great read!

(C) 2016 Jonathan Levin & Technologeeks.com - Share freely, but please cite source!
### Android Vulnerabilities

<table>
<thead>
<tr>
<th>CVE</th>
<th>Bug(s)</th>
<th>Severity</th>
<th>Updated versions</th>
<th>Date reported</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2015-6639</td>
<td>ANDROID-24446875*</td>
<td>Critical</td>
<td>5.0, 5.1.1, 6.0, 6.0.1</td>
<td>Sep 23, 2015</td>
</tr>
<tr>
<td>CVE-2015-6647</td>
<td>ANDROID-24441554*</td>
<td>Critical</td>
<td>5.0, 5.1.1, 6.0, 6.0.1</td>
<td>Sep 27, 2015</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CVE</th>
<th>Bug(s)</th>
<th>Severity</th>
<th>Updated versions</th>
<th>Date reported</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2016-0825</td>
<td>ANDROID-20860039*</td>
<td>High</td>
<td>6.0.1</td>
<td>Google Internal</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CVE</th>
<th>Android bugs</th>
<th>Severity</th>
<th>Updated Nexus devices</th>
<th>Date reported</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2016-2432</td>
<td>25913059*</td>
<td>Critical</td>
<td>Nexus 6, Android One</td>
<td>Nov 28, 2015</td>
</tr>
</tbody>
</table>
References

• ARM TrustZone documentation:

• *OS Internals (Vol. III) – Security & Insecurity of Apple’s OSes
  – The unplanned 300+pg tome that started with a single chapter..
  – Available August 2016!
Contents at a glance

Part I: Defensive Mechanisms and Technologies

1. Authentication
2. Auditing (MacOS)
3. Authorization - authd and GateKeeper (MacOS)
4. Authorization - KAuth
5. MACF - The Mandatory Access Control Framework
6. Code Signing
7. AppleMobileFileIntegrity (MacOS 10.10+, iOS)
8. Sandboxing
9. System Integrity Protection (MacOS 10.11)
10. Privacy
11. The secure boot chain (iOS)
12. Encryption

Part II: Vulnerabilities and Exploitation

13. MacOS: Vulnerabilities, past and present
   ○ rootpipe
   ○ dyld issues in 10.10.x
   ○ tpwn
14. iOS: Jailbreaking
15. evasi0n (6.x)
16. evasi0n 7 (7.0.x)
17. Pangu Axe (7.1.x)
18. XuanYuan Sword (8.0-8.1)
19. TaiG (8.1.2), TaiG (8.4)
20. Pangu 9 (9.0.x) and 9.1