Giving Mobile Security the Boot

Jonathan Levin

http://Technologeeks.com

Plan

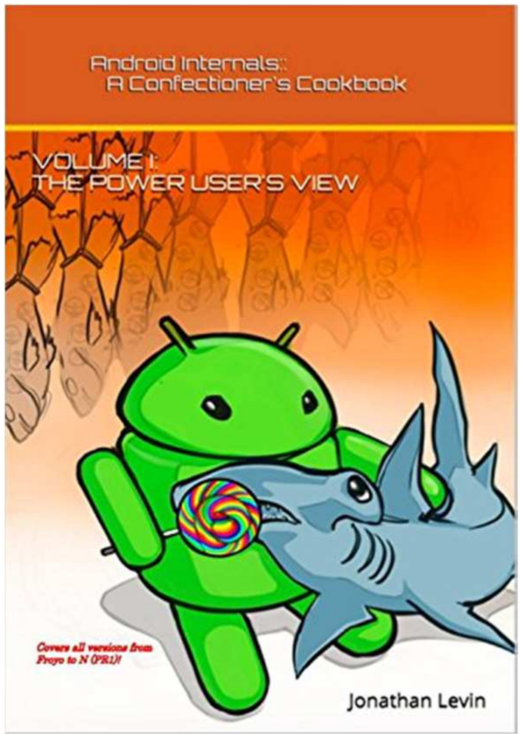
- Android Boot Chain
- iOS Boot Chain
- TrustZone
- iOS & TrustZone
- Android & TrustZone

morpheus@Zepyhr$ whoami

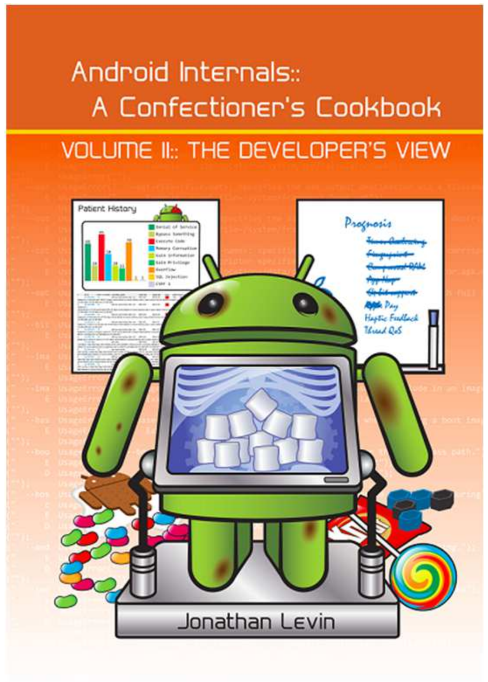
(一点儿宣传 ☺)

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

Volume I (available)



Volume II (soo-N)



morpheus@Zepyhr$ whoami

(一点儿宣传 ☺)

- 深入解析Mac OS X & iOS操作系统
- http://NewOSXBook.com/

- Plenty of useful reversing tools
  - jtool
  - procexp
  - filemon

- But book terribly outdated!

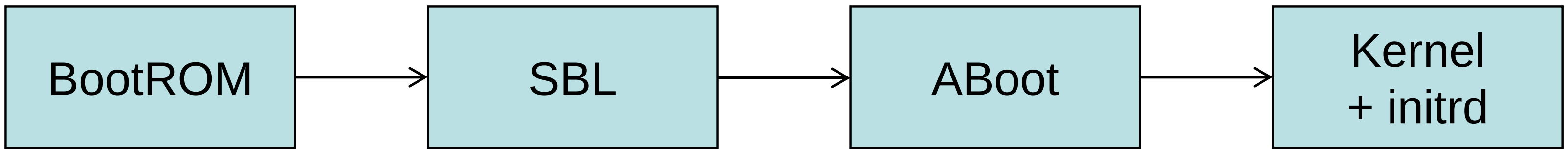
Boot Chains of Trust

The Android Boot Sequence

- Exact flow varies with vendor, but can be generalized

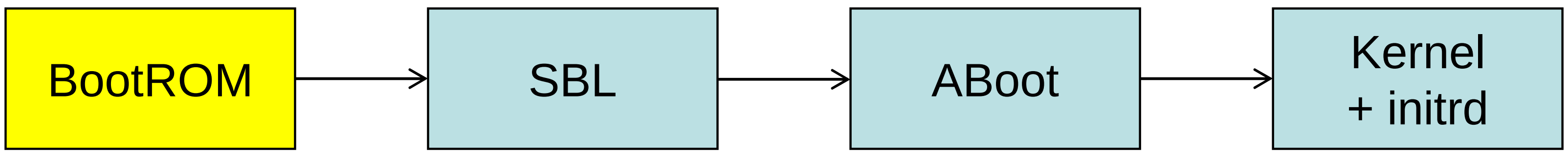
- Components (except ROM) easily extracted from OTA

```
morpheus@Forge (~)% imgtool Images/hammerhead-kot49h/bootloader-hammerhead-hhz11k.img
Boot loader detected
6 images detected, starting at offset 0x200. Size: 2568028 bytes
Image: 0        Size:  310836 bytes     sbl1     # 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

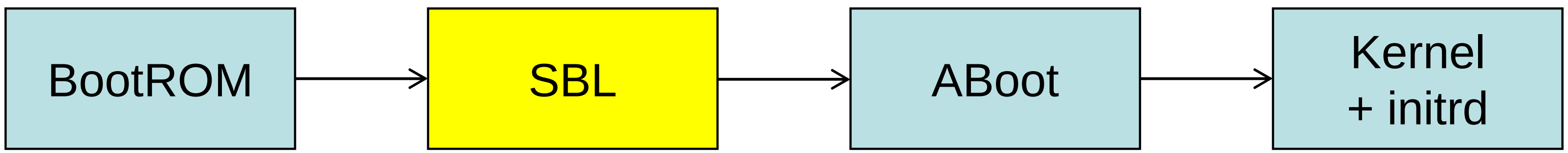
- Very specific per chipset manufacturer
- Not much is known about ROMs
- But not really relevant for our discussion, either
- Contain a hard coded public key(公钥) of manufacturer



Android Boot: SBL

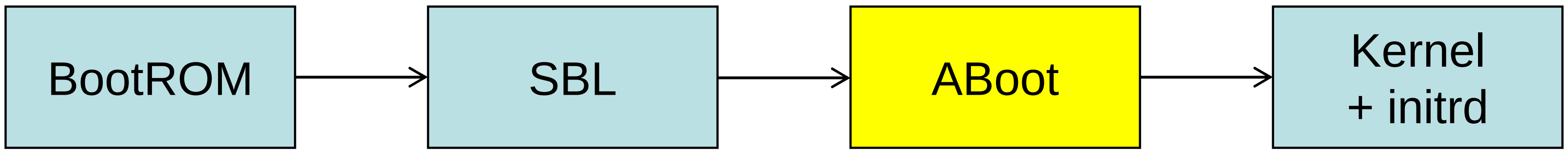
- Vendor specific, but usually same operation:
  - Initialize subsystems (baseband, DSP, GPU, TZ)
  - Locate Android Boot

- 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

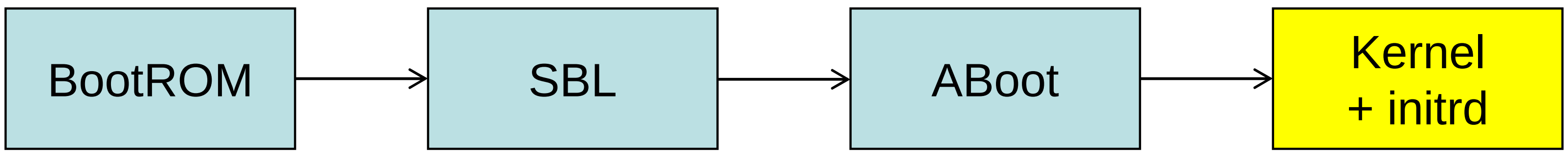


* - Samsung, others have custom loaders

Android Boot: Kernel + initrd

- Kernel is same ol’ Linux, but compiled for ARMv7/v8
- InitRD (初始 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)

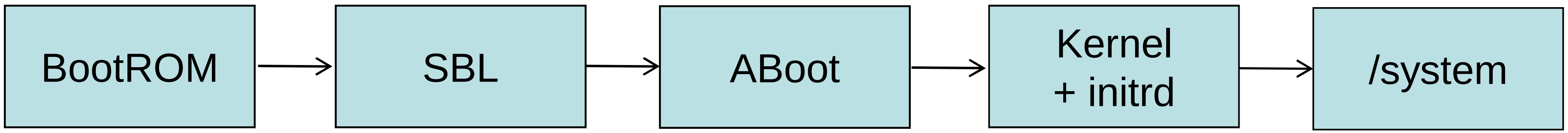


Android Boot: DM-Verity

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

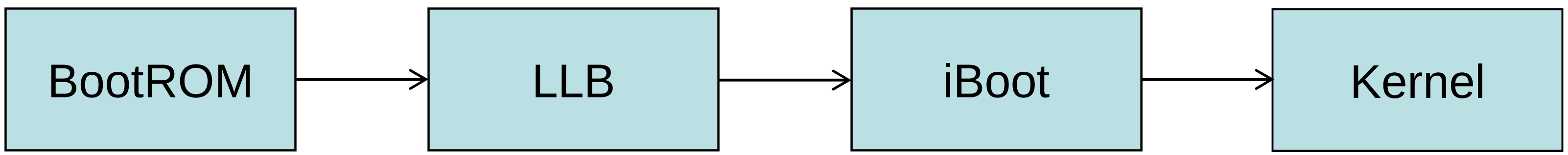
- /system mounted through device mapper, as dm# device
- All I/O flows through device mapper, verifies hashes
  - Incorrect hash causes I/O error

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



iOS Boot Sequence

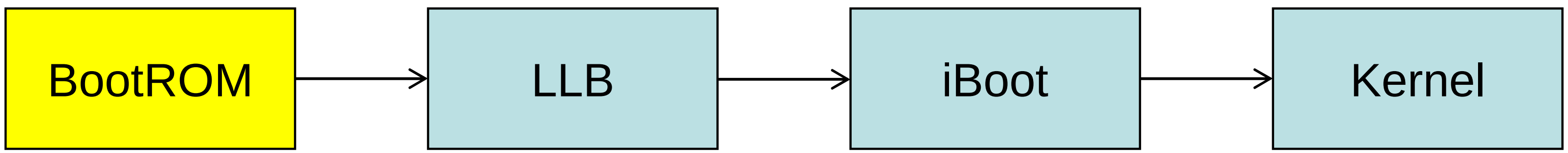
- All boot components are encrypted
  - 32-bit: IMG3 64-bit: IMG4 (DER)
- All boot components are validated
  - Slightest error sends device to recovery (and forced upgrade!)
- 64-bit boot sequence still not broken*
- 64-bit systems bolstered with Kernel Patch Protection (9.0)
  - Feeble (but valiant) attempt to prevent runtime kernel patches



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

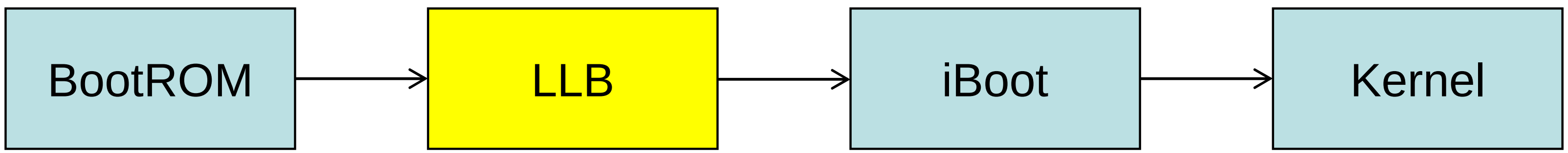
iOS Boot: The BootROM

- Read only memory component, “Apple SecureROM”
  - Contains harded 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



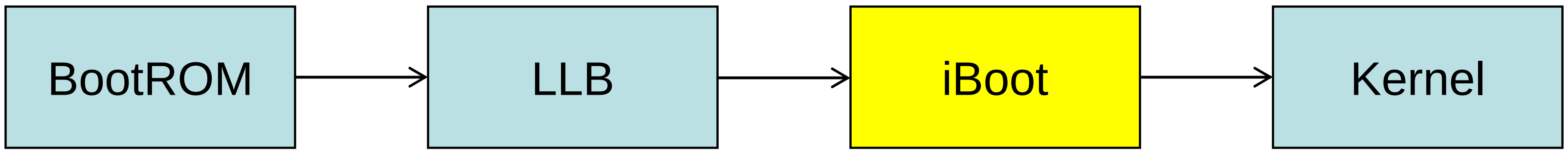
iOS Boot: iLLB

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



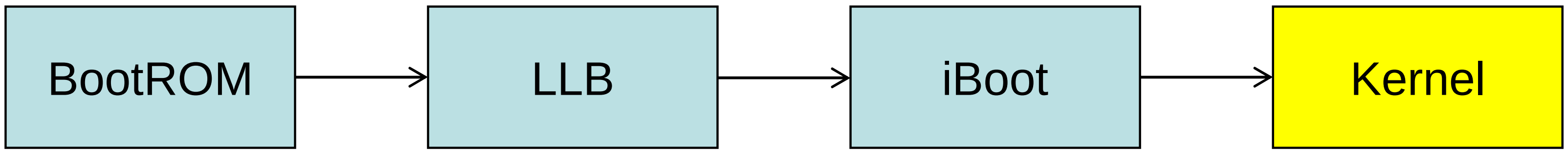
iOS Boot: iBoot

- 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

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

- iTunes POSTs to http://www.gs.apple.com
- Apple signs with their private key.
- iBoot stores in NAND firmware partition SCAB container

https://www.theiphonewiki.com/wiki/SHSH_Protocol

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

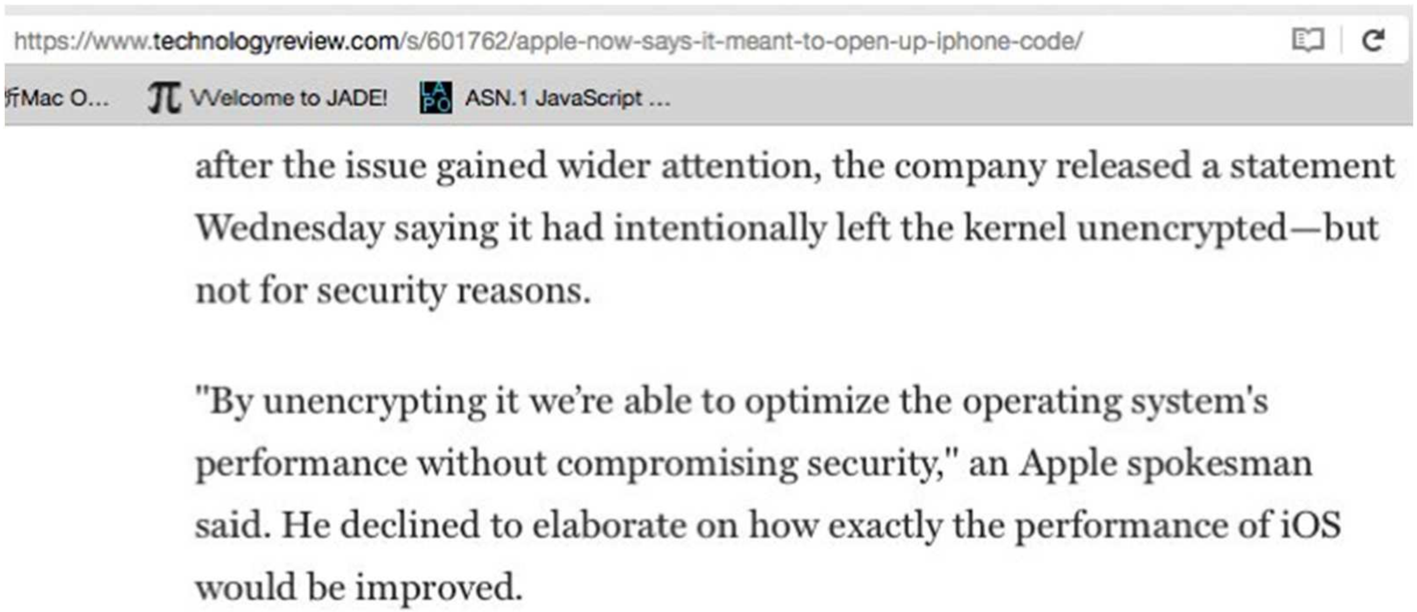
- Same as SHSH, but image now contains ApNonce

Key	Value
ApBoardID	From IPSW
ApChipID	From Device
ApECID	Exclusive Chip ID
ApNonce	Random From iBoot(!) 隨機產生
ApProductionMode	true (unfortunately)
ApSecurityDomain	From IPSW
ApTicket	true
UDID	Unique Device Identifier
HostPlatformInfo	iTunes host OS identifier
Locality	en_US, zh_CN, etc..
VersionInfo	libauthinstall-a.b.c.d.e

- 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



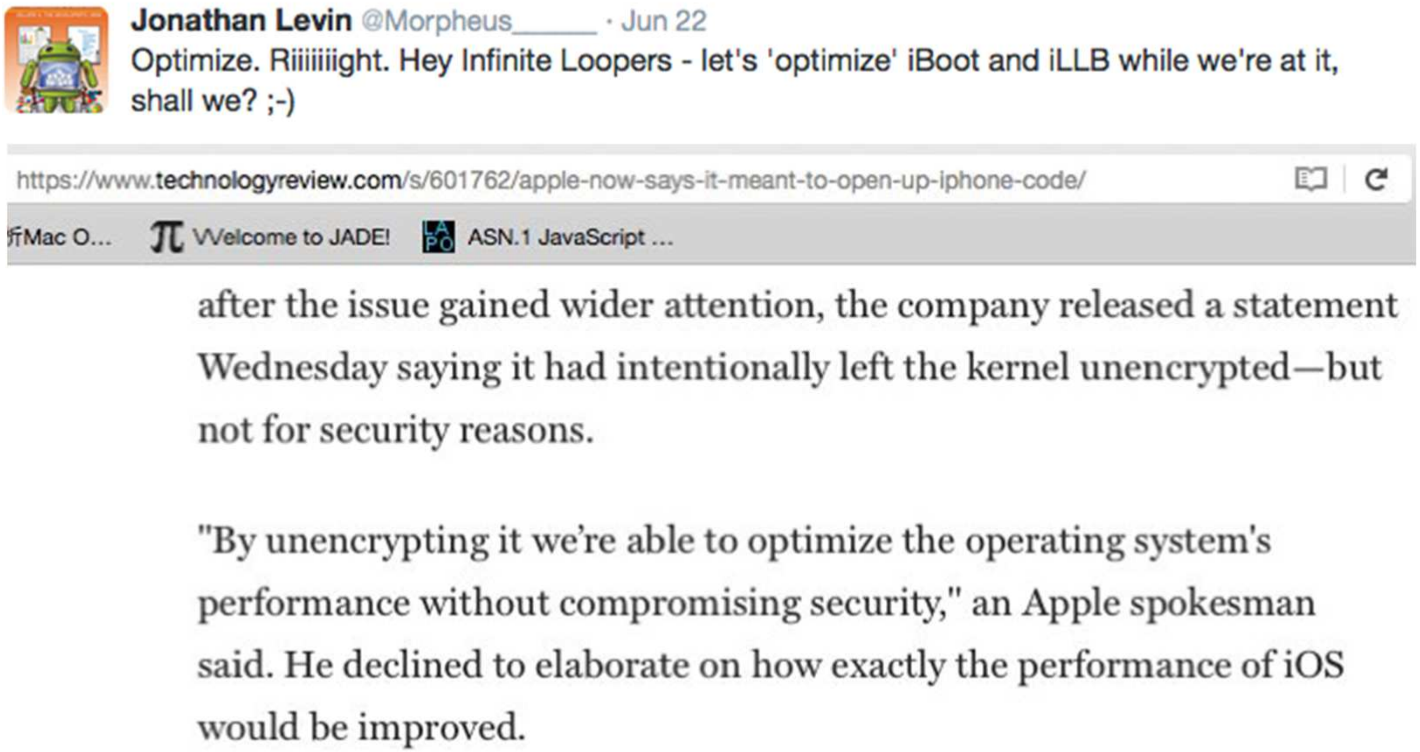
after the issue gained wider attention, the company released a statement Wednesday saying it had intentionally left the kernel unencrypted—but not for security reasons.

"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...



---

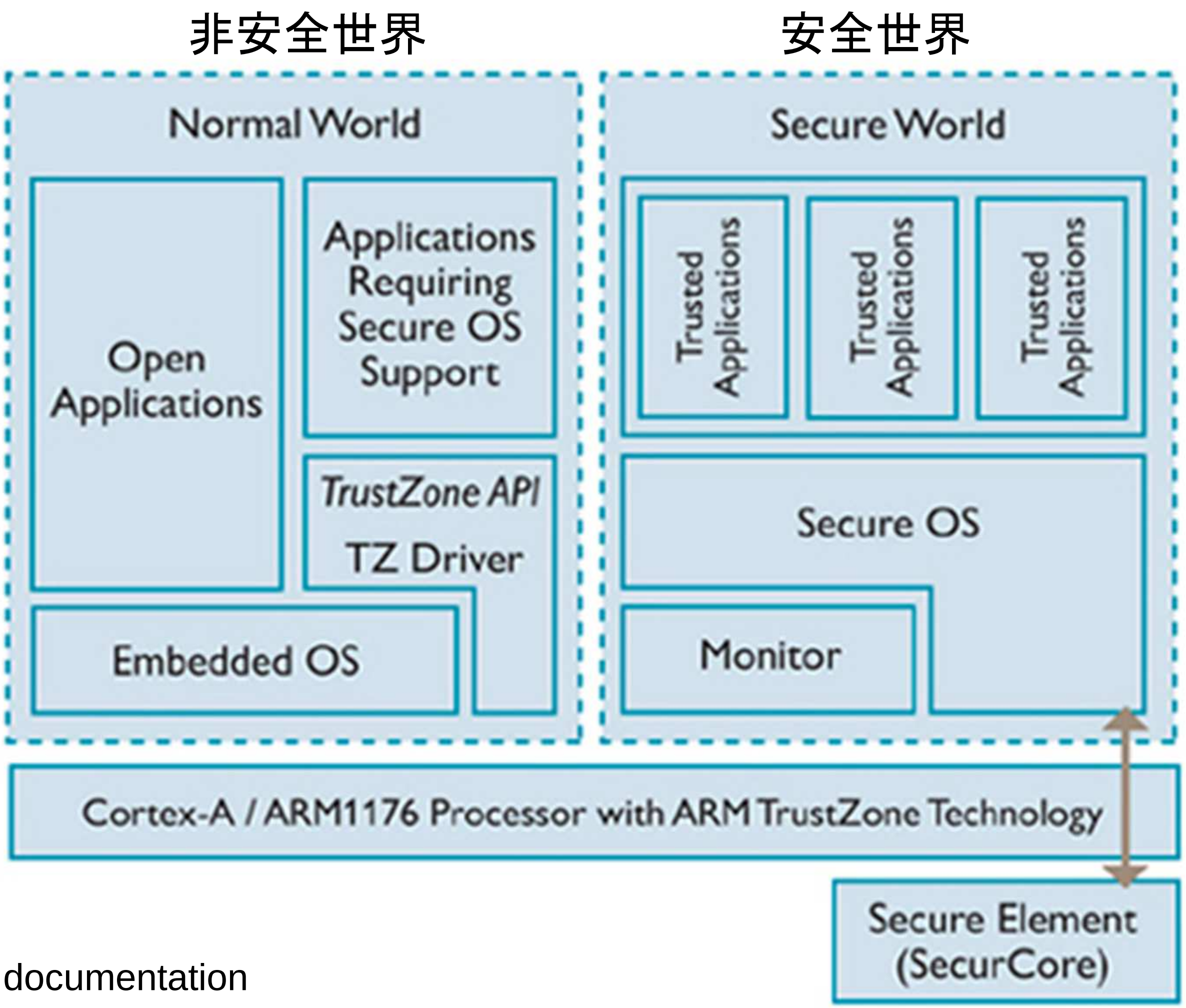
* Edit – Apple apparently took this seriously and did open the 32-bit chain (but NOT 64) in 10b2.

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

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)

安全配置寄存器

- Implemented by a Secure Configuration Register (SCR)

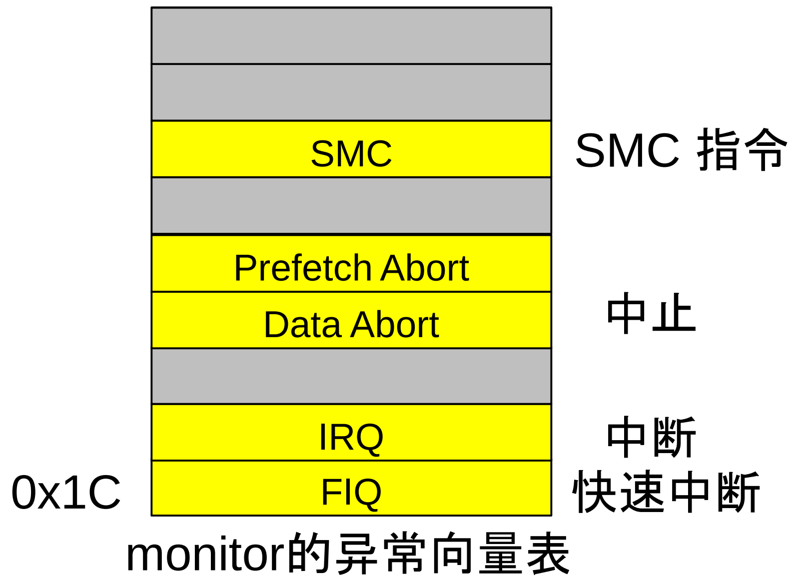
- NS = 0: 系统处于安全状态. NS =1 系统处于非安全状态
- SCR is co-processor CP15,c1
- Cannot be accessed in non-secure world:
  - Need SMC特殊指令
- MMU enforces memory separation between worlds
  - http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0301h/Chdfjdgi.html
- Interrupts (IRQ/FIQ) can be handled by secure world

Entering TrustZone (AArch32)

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

  SMC是一个特殊指令，类似于软件中断指令（SWI）

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



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)

C6.6.165 SMC

Secure Monitor Call causes an exception to EL3.

SMC is available only for software executing at EL1 or higher. It is UNDEFINED in EL0.

If the values of HCR_EL2.TSC and SCR_EL3.SMD are both 0, execution of an SMC instruction at EL1 or higher generates a Secure Monitor Call exception, using the EC value 0x17, that is taken to EL3. When EL3 is using AArch32, this exception is taken to Monitor mode.

If the value of HCR_EL2.TSC is 1, execution of an SMC instruction in a Non-secure EL1 state generates an exception that is taken to EL2, regardless of the value of SCR_EL3.SMD. When EL2 is using AArch32, this is a Hyp Trap exception that is taken to Hyp mode. For more information, see *Traps to EL2 of Non-secure EL1 execution of SMC instructions* on page D1-1506.

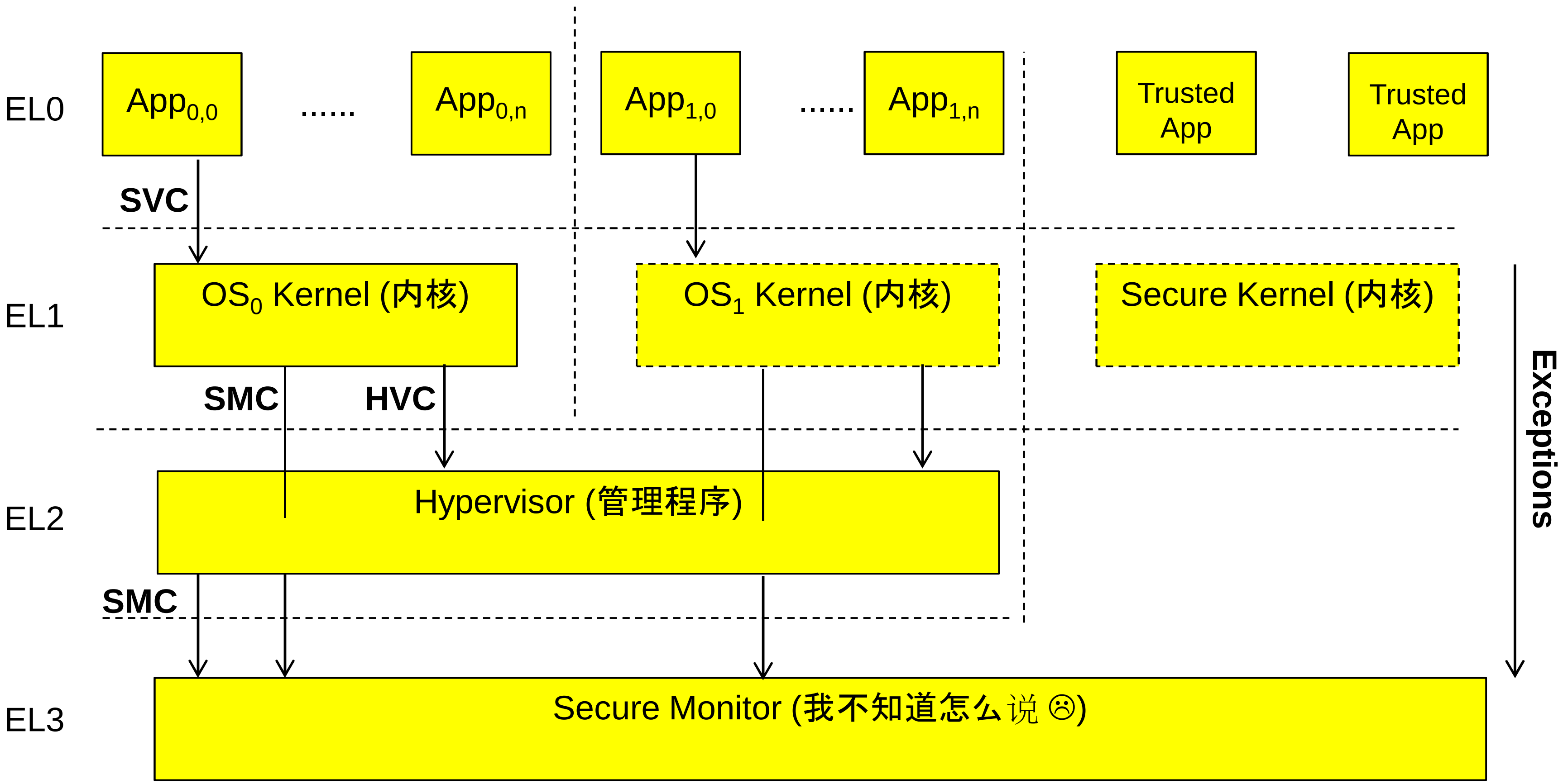
If the value of HCR_EL2.TSC is 0 and the value of SCR_EL3.SMD is 1, the SMC instruction is:

- UNDEFINED in Non-secure state.
- CONSTRAINED UNPREDICTABLE if executed in Secure state at EL1 or higher.

31 30 29 28	27 26 25 24	23 22 21	20 … 5	4 3 2	1 0
1 1 0 1	0 1 0 0	0 0 0	imm16	0 0 0	1 1

D 4 0 3

Exception Handling (AArch64)



架構定義了四個例外層級

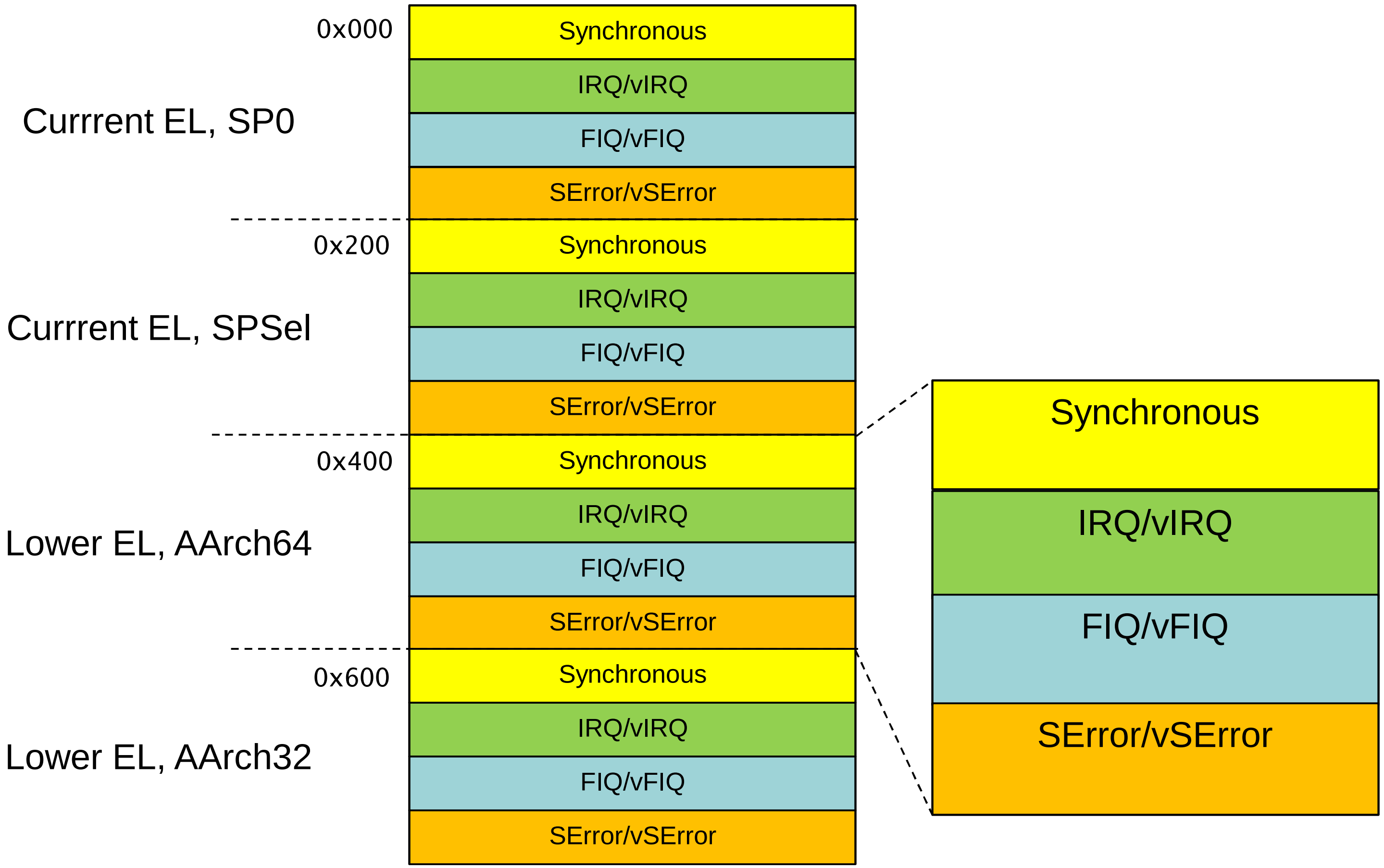
(特权模型分离技术)

Setting up Trustzone

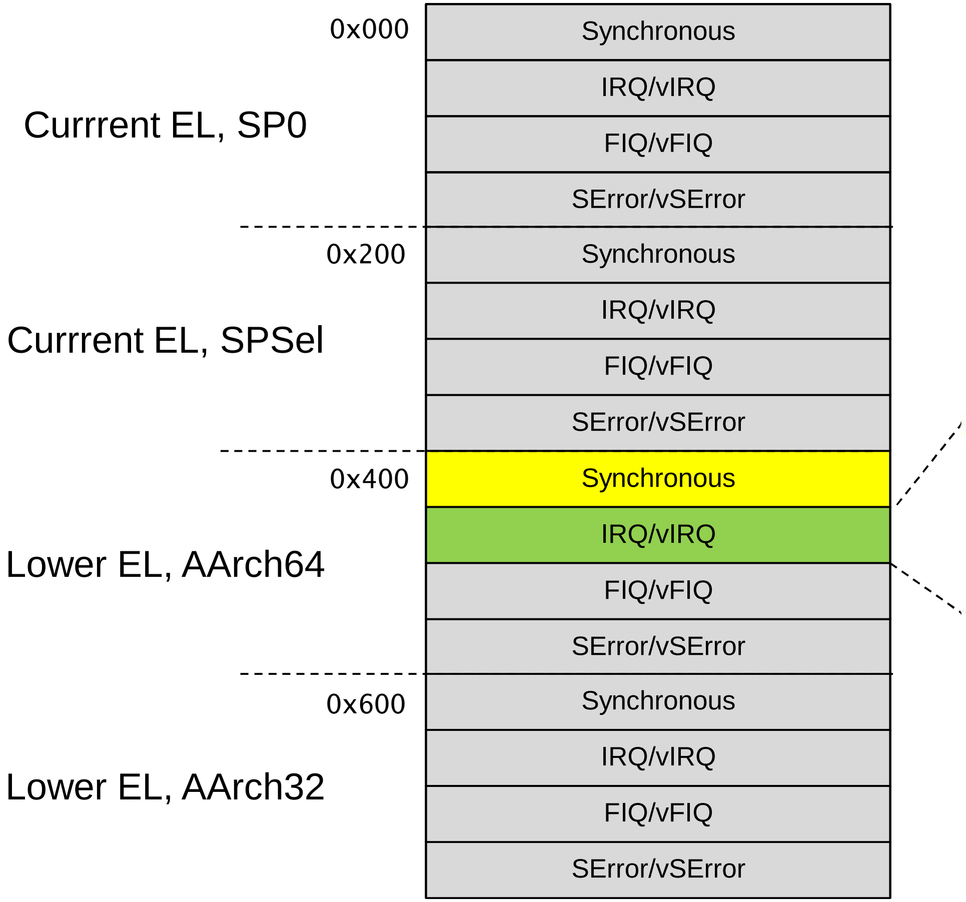
- 32-bit:
  - CPU boots into secure world (NS=0)
  - Loader/kernel sets up monitor vector (SMC, IRQ or FIQ entries)
  - Sets up SCR NS=1 and “drops” to Normal World

- 64-bit:
  - CPU boots into EL3
  - Secure Monitor sets up VBAR_Elx (SError, IRQ or FIQ entries) (异常向量表基地址寄存器指定)
  - Drops to EL2 (Hypervisor, 管理程序) or EL1 (kernel,内核)

AArch64 Exception Handling

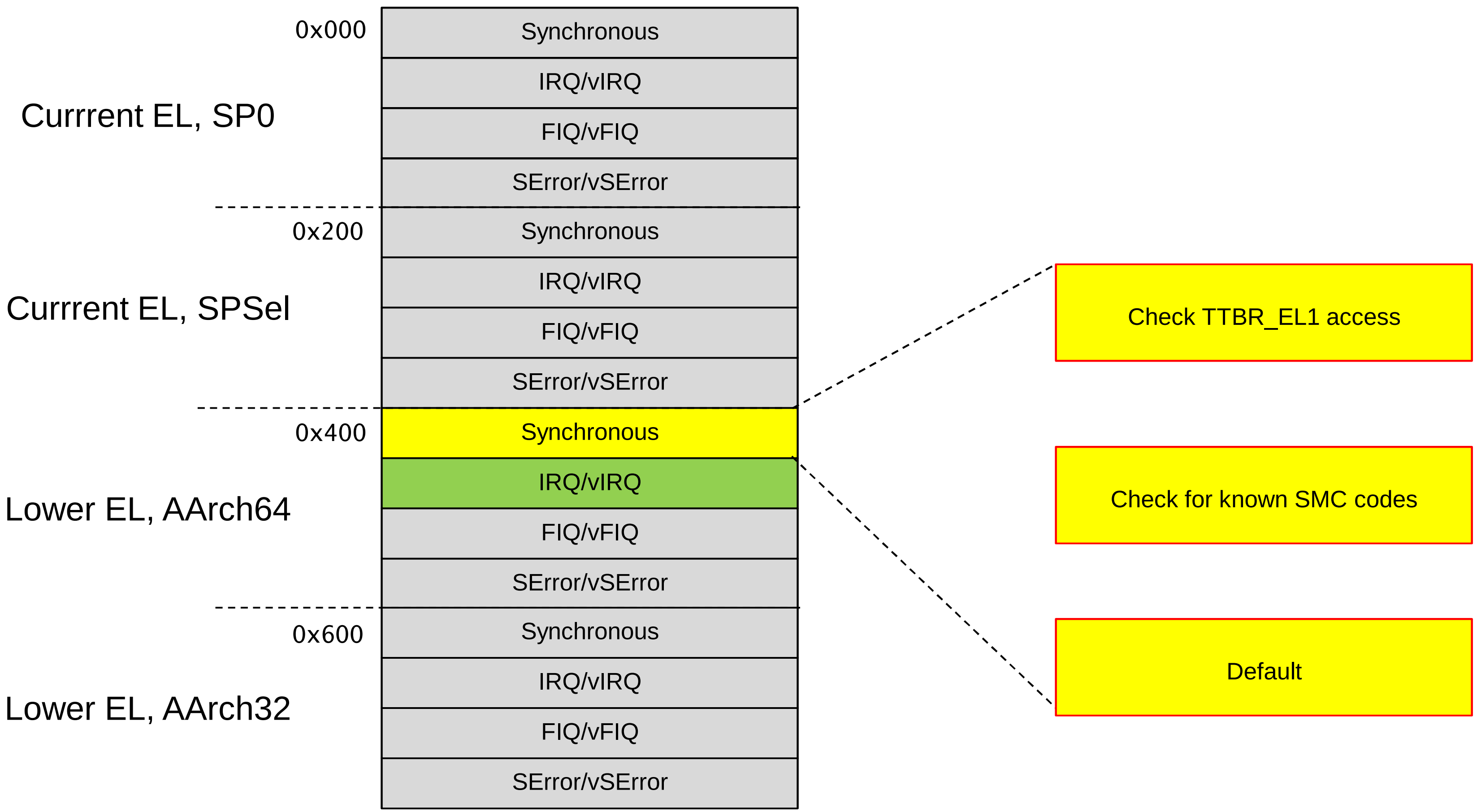


Case Study: KPP

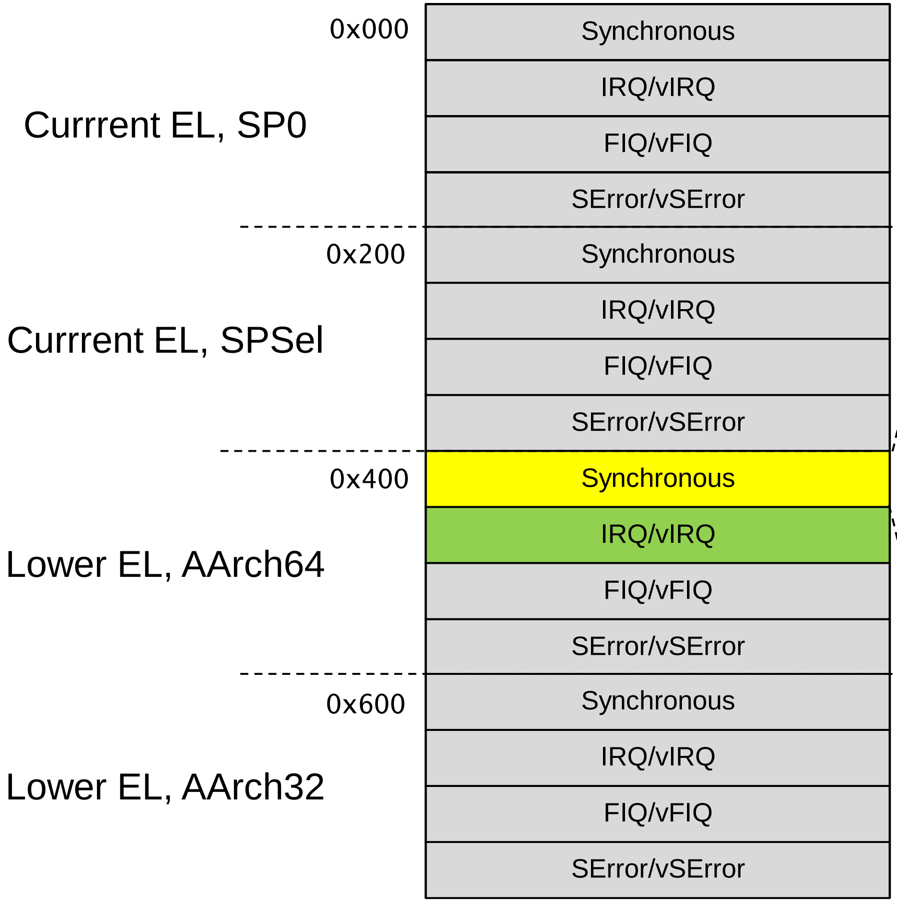



```
IRQ/vIRQ_Lower_EL:
  4100003480    MSR    TPIDR_EL3, X0       ; // Thread Pointer/ID Register
  4100003484    MOVZ   X0, 0x431           ; //    ; ->R0 = 0x431
  4100003488    MSR    SCR_EL3, X0         ; // NS,(RES1!=3),RW (lower level AArch64)
  410000348c    MOVZ   X0, 0x10, LSL #16         ; //    ; ->R0 = 0x100000
  4100003490    MSR    CPACR_EL1, X0       ; // FPEN=1 (el0 fp/simd trap)
  4100003494    MOVZ   X0, 0x8000, LSL #16       ; //    ; ->R0 = 0x80000000
  4100003498    MSR    CPTR_EL3, X0        ; // !TFP,TCPAC,!TTA
  410000349c    MRS    X0, TPIDR_EL3       ; // Thread Pointer/ID Register
  41000034a0    ERET                       ; // ;
```

Case Study: KPP



Case Study: KPP




```
;
; function #23
;
_do_synchronous_LowerEL:
  4100004e28    STP    X20, X19, [SP,#-32]!        ;
  4100004e2c    STP    X29, X30, [SP,#16]          ;
  4100004e30    ADD    X29, SP, #16                ; X29 = SP + 0x10
  4100004e34    SUB    SP, SP, 48                  ; SP -= 0x30 (stack frame)
  4100004e38    ORR    W19, WZR, #0x0              ; X19 = 0x0
;
; compares ESR_EL3 to MSR,MRS (0x18), with 0x340400:
; op0=3,op2=1, CRn=2, !CRm (TTBR_EL1)
;
 check_if_TTBR_EL1_access:
  4100004e3c    MRS    X8, ESR_EL3                 ;
  4100004e40    MOVZ   W9, 0x6234, LSL #16         ; X9 = 0x62340000
  4100004e44    MOVK   X9, 0x400                   ; X9 = 0x62340400
  4100004e48    CMP    W8, W9                      ;
  4100004e4c    B.NE   check_if_SMC      ; 0x410000503c
...
;
; Compare_ESR_EL3_to_SMC (0x17) 0x5e000011 with imm 0x11!
;
check_if_SMC:
  410000503c    MOVZ   W9, 0x5e00, LSL #16         ; X9 = 0x5e000000
  4100005040    MOVK   X9, 0x11                    ; X9 = 0x5e000011
  4100005044    CMP    W8, W9                      ;
  4100005048    B.EQ   ; SMC_handler     ; 0x4100005558
Not_SMC_or_unknown_SMC:
  410000504c    MRS    X8, SPSR_EL3                ;
  4100005050    AND    X9, X8, #0xffffffff         ;
  4100005054    MSR    SPSR_EL1, X9                ;
  4100005058    MRS    X9, ELR_EL3                 ;
...
SMC_handler:
  4100005558    CMP    X0, #2050           ;
  410000555c    B.EQ     SMC_2050_handler          ; 0x4100005808
  4100005560    CMP    X0, #2049           ;
  4100005564    B.EQ    SMC_2049_enforce_handler          ; 0x4100005860
  4100005568    CMP    X0, #2048           ;
  410000556c    B.NE    SMC_unknown_handler        ; 0x410000504c
SMC_2048_unknown_handler:
  4100005570    ADR    X8, #56624          ;  X8 = 0x41000132a0 can_enforce_if_this_is_1
  4100005574    NOP                        ; // ;
...
```

KPP: Kernel Side

```
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 0xfffffff00747c000
fffffff007483b0c    d4000223    SMC    #17             ;
#
# Add symbol to companion file, for easy reference later:
morpheus@zephyr (~/.../iOS10)$ echo 0xfffffff007483b0c:_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 0xfffffff00747c000
fffffff0074c002c    MOVZ   W0, 0x801                ; ->R0 = 0x801
fffffff0074c0030    MOVZ   X1, 0x0                  ; ->R1 = 0x0
fffffff0074c0034    MOVZ   X2, 0x0                  ; ->R2 = 0x0
fffffff0074c0038    MOVZ   X3, 0x0                  ; ->R3 = 0x0
fffffff0074c003c    BL     _smc                     ; 0xfffffff007483b0c
...
fffffff00756e780    ADD    X1, X9, X11              ; 0xfffffff107488193
fffffff00756e784    ORR    W0, WZR, #0x800          ; ->R0 = 0x800
fffffff00756e788    MOVZ   X2, 0x0                  ; ->R2 = 0x0
fffffff00756e78c    MOVZ   X3, 0x0                  ; ->R3 = 0x0
fffffff00756e790    BL     _smc                     ; 0xfffffff007483b0c
```

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    "<key>_PrelinkExecutableLoadAddr</key>" ; R22 = ..
  41000042a8  ADR    X25, #12635        "__DATA"  ; ->R25 = 0x4100007403
  41000042c8  ADR    X1, #12570         "__TEXT"  ; ->R1 = 0x41000073e2
  41000042e0  ADR    X1, #12564         "__PRELINK_INFO"    ; ->R1 = 0x41000073f4
  41000044fc  ADR    X23, #12280        "???.kext"          ; ->R23 = 0x41000074f4
  410000451c  ADR    X1, #12140         "<key>_PrelinkExecutableLoadAddr</key><integer
ID="%u" size="64">0x%llx</integer><key>_PrelinkKmodInfo</key>"       ; ->R1 = 0x4100007488
  4100004570  ADR    X1, #11976
"<key>_PrelinkBundlePath</key><string>/System/Library/Extensions/"    ; ->R1 = 0x4100007438
  4100004588  ADR    X0, #11952
"<key>_PrelinkBundlePath</key><string>/System/Library/Extensions/"    ; ->R0 = 0x4100007438
  410000459c  ADR    X1, #11997         ".kext</string>"    ; ->R1 = 0x4100007479
  4100004638  ADR    X25, #11723        "__DATA"  ; ->R25 = 0x4100007403
  41000046bc  ADR    X1, #11565         "__firmware"        ; ->R1 = 0x41000073e9
  41000048d0  ADR    X1, #11066         "__const" ; ->R1 = 0x410000740a
  41000049ac  ADR    X1, #10846         "__const" ; ->R1 = 0x410000740a
```

iOS 10 changes

```
morpheus@zeyphr(~/.../iOS10)$ jtool -v -l ~/Documents/iOS/9b/kernel.dump.9.3.0 | grep SEGM
LC 00: LC_SEGMENT_64  Mem: 0xffffff8006804000-0xffffff8006cec000  File: 0x0-0x4e8000        r-x/r-x __TEXT
LC 01: LC_SEGMENT_64  Mem: 0xffffff8006cec000-0xffffff8006db0000  File: 0x4e8000-0x540000   rw-/rw- __DATA
LC 02: LC_SEGMENT_64  Mem: 0xffffff8006db0000-0xffffff8006db4000  File: 0x540000-0x544000   rw-/rw- __KLD
LC 03: LC_SEGMENT_64  Mem: 0xffffff8006db4000-0xffffff8006db8000  File: 0x544000-0x548000   rw-/rw- __LAST
LC 04: LC_SEGMENT_64  Mem: 0xffffff8006e14000-0xffffff80082a8000  File: 0x5a4000-0x1a38000  rw-/rw- __PRELINK_TEXT
LC 05: LC_SEGMENT_64  Mem: 0xffffff8006db8000-0xffffff8006db8000  File: Not Mapped          rw-/rw- __PRELINK_STATE
LC 06: LC_SEGMENT_64  Mem: 0xffffff80082a8000-0xffffff800834c000  File: 0x1a38000-0x1ad9b18 rw-/rw- __PRELINK_INFO
LC 07: LC_SEGMENT_64  Mem: 0xffffff8006db8000-0xffffff8006e113a8  File: 0x548000-0x5a13a8   r--/r-- __LINKEDIT
```

```
morpheus@Zephyr (~/.../iOS10)$ jtool -v -l xnu.3705.j99a |grep SEG
LC 00: LC_SEGMENT_64  Mem: 0xfffffff007404000-0xfffffff007460000 File: 0x0-0x5c000         r-x/r-x  __TEXT
LC 01: LC_SEGMENT_64  Mem: 0xfffffff007460000-0xfffffff00747c000 File: 0x5c000-0x78000     rw-/rw-  __DATA_CONST
LC 02: LC_SEGMENT_64  Mem: 0xfffffff00747c000-0xfffffff0078dc000 File: 0x78000-0x4d8000    r-x/r-x  __TEXT_EXEC
LC 03: LC_SEGMENT_64  Mem: 0xfffffff0078dc000-0xfffffff0078e0000 File: 0x4d8000-0x4dc000   rw-/rw-  __KLD
LC 04: LC_SEGMENT_64  Mem: 0xfffffff0078e0000-0xfffffff0078e4000 File: 0x4dc000-0x4e0000   rw-/rw-  __LAST
LC 05: LC_SEGMENT_64  Mem: 0xfffffff0078e4000-0xfffffff007994000 File: 0x4e0000-0x514000   rw-/rw-  __DATA
LC 06: LC_SEGMENT_64  Mem: 0xfffffff004004000-0xfffffff005a7c000 File: 0x574000-0x1fec000 rw-/rw-  __PRELINK_TEXT
LC 07: LC_SEGMENT_64  Mem: 0xfffffff007994000-0xfffffff007994000 File: Not Mapped   rw-/rw-       __PLK_TEXT_EXEC
LC 08: LC_SEGMENT_64  Mem: 0xfffffff007994000-0xfffffff007994000 File: Not Mapped   rw-/rw-       __PRELINK_DATA
LC 09: LC_SEGMENT_64  Mem: 0xfffffff007994000-0xfffffff007994000 File: Not Mapped   rw-/rw-       __PLK_DATA_CONST
LC 10: LC_SEGMENT_64  Mem: 0xfffffff007994000-0xfffffff007994000 File: Not Mapped   rw-/rw-        __PLK_LINKEDIT
LC 11: LC_SEGMENT_64  Mem: 0xfffffff0079f4000-0xfffffff007ab0000 File: 0x1fec000-0x20a5bac rw-/rw- __PRELINK_INFO
LC 12: LC_SEGMENT_64  Mem: 0xfffffff007994000-0xfffffff0079f07a0 File: 0x514000-0x5707a0 r--/r--    __LINKEDIT
```

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

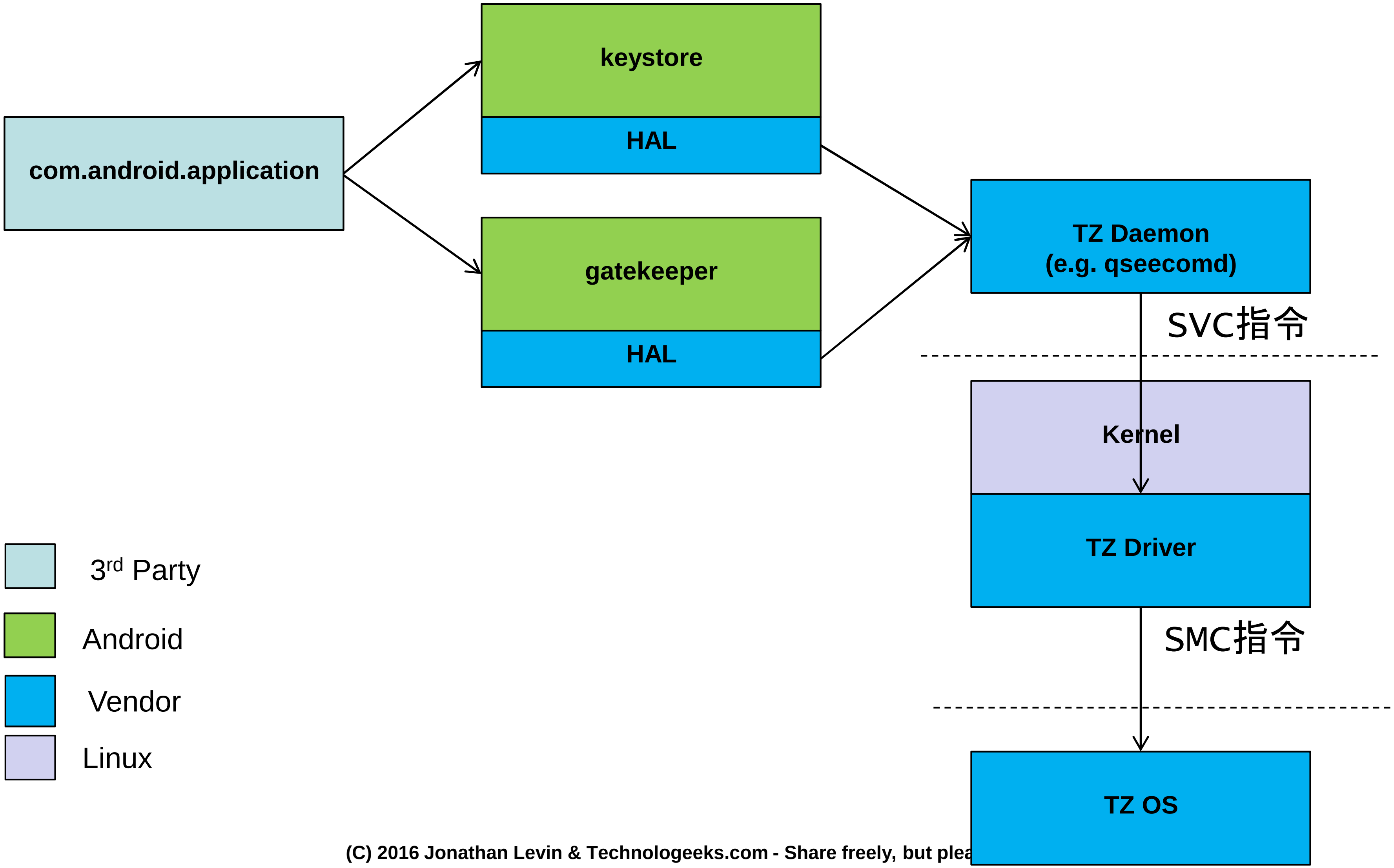
Android & TrustZone

- 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



Android & TrustZone: examples

- NVidia (Nexus 9):

```
root@flounder: /#  ls -Ll /dev/block/platform/sdhci-tegra.3/by-name/
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-rw----    1 system   system   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-rw----    1 system   system   259,   5 Nov 30 23:26 EF1 -> ..21
brw-rw----    1 system   system   259,   6 Nov 30 23:26 EF2 -> ..22
brw-------    1 root     root     179,   3 Nov 30 23:26 EKS -> ..3
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 MFG -> ..18  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-rw----    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     179,  15 Nov 30 23:26 SOS -> ..15 recovery.img (cute :-)
brw-------    1 root     root     179,   9 Nov 30 23:26 SP1 -> ..9
brw-------    1 root     root     179,   2 Nov 30 23:26 TOS -> ..2  ARM TrustZone
brw-------    1 root     root     259,  15 Nov 30 23:26 UDA -> ..31 User data (i.e /data)
brw-------    1 root     root     259,   8 Nov 30 23:26 VNR -> ..24 /vendor
brw-------    1 root     root     179,   4 Nov 30 23:26 WB0 -> ..4
brw-------    1 root     root     179,   7 Nov 30 23:26 WDM ->7
```

Android & TrustZone: Samsung

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

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

```
# disarm will automatically find strings when used as arguments
root@s6# JCOLOR=1 disarm /dev/sdb1 | 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, [SP,#-16]!
0x0003fa04      0xd4000003      SMC #0              ; (X0=0xfffffffffffff10, X1=0x0, X2=0xf7120..)
0x0003fa08      0xa8c17bfd      LDP X29, X30, [SP],#16
0x0003fa0c      0x3100041f      CMN W0, #1
0x0003fa10      0x2a0003e2      MOV X2, X0          ; X2 = X0 (?)
0x0003fa14      0x54000580      B.EQ 0x3fac4
# can also grep SMC
...
0x0004f014      0xd4000003      SMC #0 ; (X0=0xc2001014, X1=0x0, X2=0x22..)
0x0004f044      0xd4000003      SMC #0 ; (X0=0xc2001014, X1=0x0, X2=0x21..)
0x0004f098      0xd4000003      SMC #0 ; (X0=0xc2001014, X1=0x0, X2=0x20..)
0x0004f0c8      0xd4000003      SMC #0 ; (X0=0xc2001014, X1=0x0, X2=0x1f..)
...
```

Simple but effective ARM64 disassembler (http://NewAndroidBook.com/tools/disarm.html)

Trusty

- Google's attempt to standardize TEE Oses
  - https://source.android.com/security/trusty/index.html
- 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
  - http://bits-please.blogspot.com/ - Step by step hack of QCOM TZ
    - Amazing exploit and explanation – Masterful hack, and a great read!

Android Vulnerabilities

CVE	Bug(s)	Severity	Updated versions	Date reported
CVE-2015-6639	ANDROID-24446875*	Critical	5.0, 5.1.1, 6.0, 6.0.1	Sep 23, 2015
CVE-2015-6647	ANDROID-24441554*	Critical	5.0, 5.1.1, 6.0, 6.0.1	Sep 27, 2015

CVE	Bug(s)	Severity	Updated versions	Date reported
CVE-2016-0825	ANDROID-20860039*	High	6.0.1	Google Internal

CVE	Android bugs	Severity	Updated Nexus devices	Date reported
CVE-2016-2431	24968809*	Critical	Nexus 5, Nexus 6, Nexus 7 (2013), Android One	Oct 15, 2015
CVE-2016-2432	25913059*	Critical	Nexus 6, Android One	Nov 28, 2015

References

- ARM TrustZone documentation:
  - http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0301h/Chdfjdgi.html

- ***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

Part II: Vulnerabilities and Exploitation