Attack Secure Boot of SEP

windknown@pangu

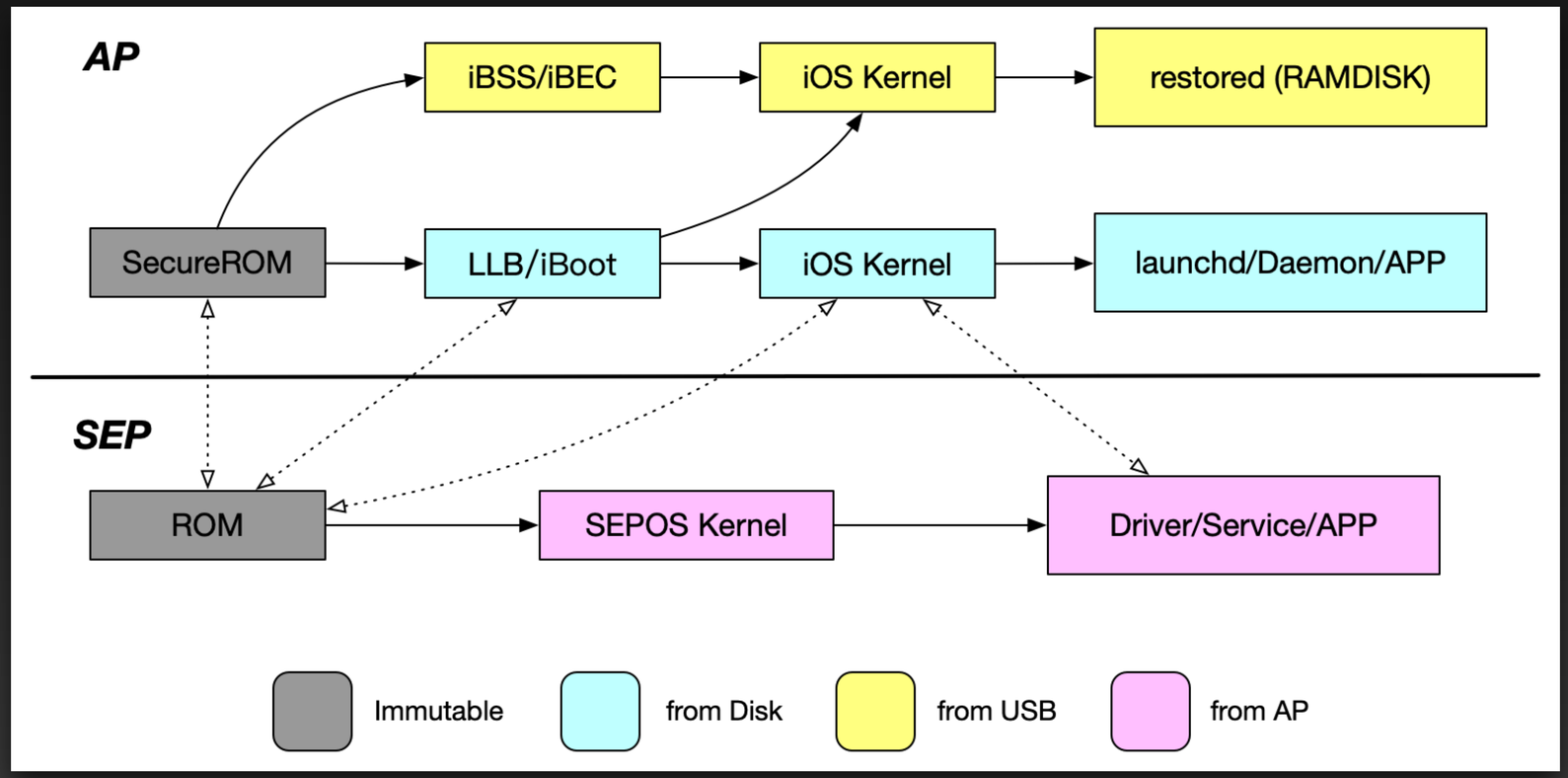
Agenda

- Secure Boot of iPhone
- SEP Hardware
- SEP Boot Process
  - Mailbox
  - Memory Layout
- Boot TZ0
  - Memory protection
  - New Memory Layout
- Attack Secure Boot
  - Access Memory of SEP
  - Control Memory of SEP

Secure Boot of iPhone

Boot Chain

- AP and SEP have separated secure boot chains
- AP has different modes (Normal/DFU/Restore/Upgrade)
- SEP is straightforward



Key Concepts

- IMG4 file = Payload (IM4P) + Manifest (IM4M)
- Payload verification
  - Manifest is verified by Apple’s Root CA public key embedded in ROM
  - The hash of each stage payload must matches the one in manifest
- Payload decryption
  - Keybags of iBSS/iBEC/LLB/iBoot are decrypted by AP GID key
  - Keybag of SEP firmware is decrypted by SEP GID key
- SEP has its own nonce for DFU/Recovery

Attack Secure Boot

- @axi0mx released source code of checkm8 exploit
  - It supports devices up to iPhone X (A11)
  - The vulnerability lies in AP ROM so Apple cannot fix it
  - Allow us to execute arbitrary code in DFU mode
- checkra1n jailbreak was released by checkra1n team
  - Lifetime tethered jailbreak ?
  - Apple tries to stop it in iOS 14
    - checkm8 does not allow us to control code running on SEP
    - SEPROM bug ?

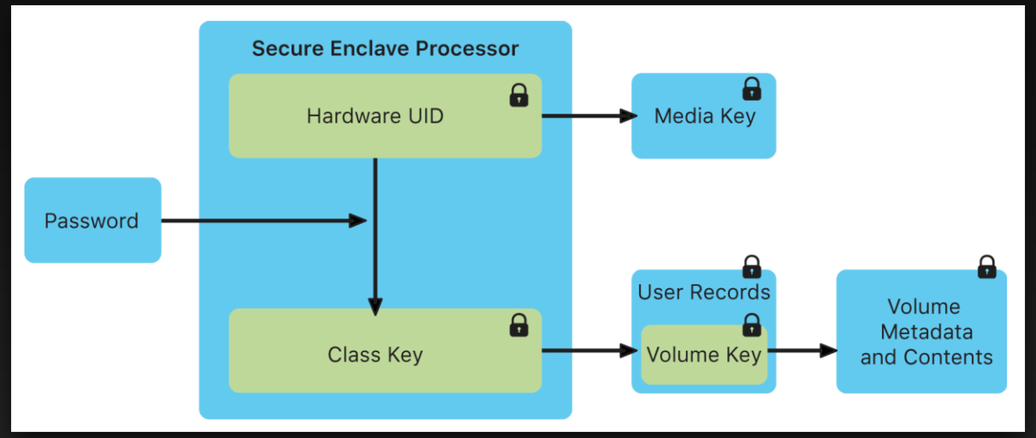
SEP Hardware

Basics of SEP Hardware

- CPU architecture
  - ARMv7 - iPhone 5S/6/6S/SE/7
  - ARMv8 - iPhone 8/X and newer devices
- Dedicated RAM
  - Relatively small size - 4096 bytes for old devices
- Dedicated peripherals
  - Crypto engine (GID/UID)
  - TRNG
- Shared peripherals with AP
  - Power manager (PMGR)
  - Mailbox
  - Shared memory

References

- “Demystifying the Secure Enclave Processor”
  - Black Hat USA 2016
- https://securerom.fun 😍
  - SEPROM binaries for A7/A8/A9/A10
- “Apple Platform Security (Fall 2019)”
  - Communication between the Secure Enclave and the application processor is tightly controlled by isolating it to an interrupt-driven mailbox and shared memory data buffers.
  - When the device starts up, an ephemeral memory protection key is created by the Secure Enclave Boot ROM, entangled with the device’s unique ID (UID), and used to encrypt the Secure Enclave’s portion of the device’s memory space.



SEP Boot Process

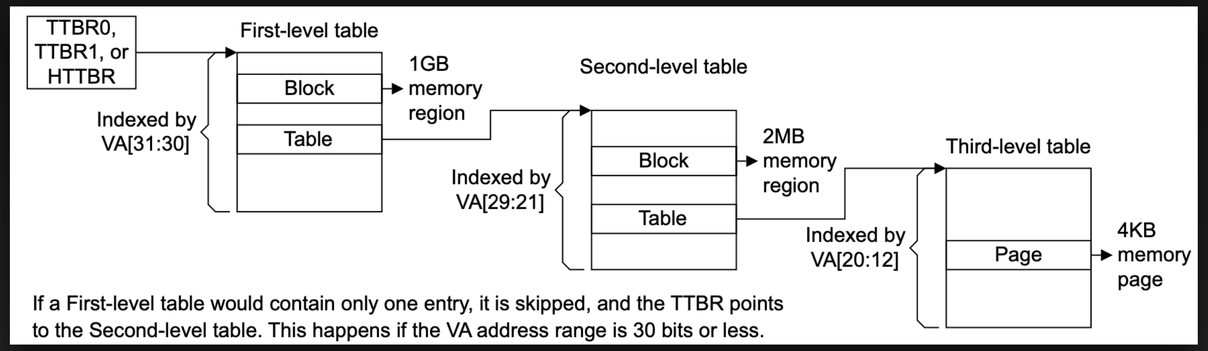
Startup

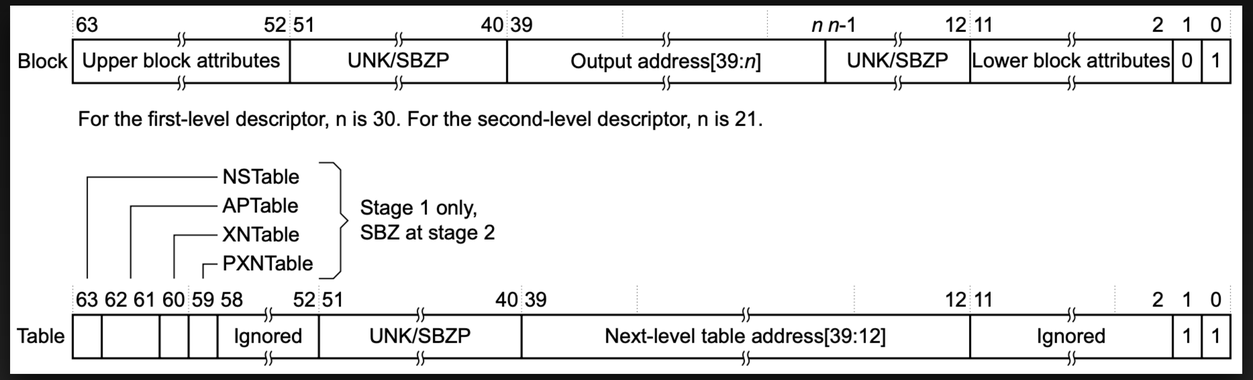
- Codes snapshots are from AppleSEPROM-230.0.0.4.1 (S8000) by default
- Instruction at PA (physical address) 0x0 is executed
  - LDR PC, =sub_4254
    - Setup MMU to enable virtual memory address translation
    - Jump to init function at 0x1000_4CDC and switch to thumb

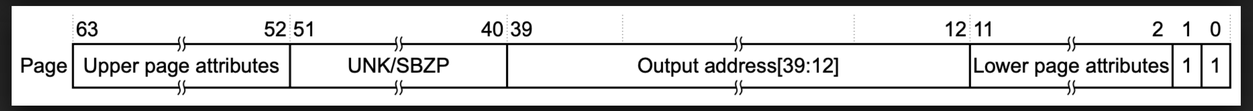
```
MRRC            p15, 0, R0,R1,c14 ; [<] CNTPCT (Counter-timer Physical Count register)
MRC             p15, 0, R2,c1,c0, 1 ; [<] ACTLR (Auxiliary Control Register)
ORR             R2, R2, #0x40
MCR             p15, 0, R2,c1,c0, 1 ; [>] ACTLR (Auxiliary Control Register)
MOV             R2, #0x4FF44F04
MCR             p15, 0, R2,c10,c2, 0 ; [>] MAIR0 (Memory Attribute Indirection Register 0)
                        ;  or PRRR (Primary Region Remap Register)
MOV             R2, #0x4044F04
MCR             p15, 0, R2,c10,c2, 1 ; [>] MAIR1 (Memory Attribute Indirection Register 1)
                        ;  or NMRR (Normal Memory Remap Register)
MOV             R2, #0xD0000A0
MOVS            R3, #2
MCRR            p15, 0, R2,R3,c2 ; [>] TTBR0 (Translation Table Base Register 0)
MOV             R2, #0x80802300
MCR             p15, 0, R2,c2,c0, 2 ; [>] TTBCR (Translation Table Base Control Register)
MOV             R2, #0x30C5187D
MCR             p15, 0, R2,c1,c0, 0 ; [>] SCTLR (System Control Register)
DSB             SY
ISB             SY
MOV             R12, #(init+1)
BX              R12     ; init
```

Virtual Memory Setup

- TTBCR.EAE=1 -> Using Large Physical Address Extension
  - 32bit VA map to 40bit PA (Long-descriptor translation)







- TTBR0 = 0x2_0D00_00A0 (64-bit format)
  - Guess SEPROM is mapped at PA 0x2_0D00_0000
  - Page tables are stored at ROM+0xA0
  - MAIR: Attr0=04 Attr1=4F Attr2=F4 Attr3=4F Attr4=04 Attr5=4F Attr6=04 Attr7=04

Virtual Memory Layout

- 1st-level table at ROM+0xA0 has two entries
  - One table and one block
- 2nd-level table is at ROM+0x1000
- 3rd-level table is at ROM+0x2000 and ROM+0x3000

ROM

Dedicated RAM (Stack+Data)

Unavailable for now

Mapped IO registers

VA	PA	Size	Access	AttrIndx
0x4000	0x2_0D00_4000	0x1000	r-x	Attr0=04 (Device memory)
0x1000_0000	0x2_0D00_0000	0x2_0000	r-x	Attr3=4F (Normal memory)
0x1013_0000	0x2_0D90_0000	0x1000	rw-	Attr5=4F (Normal memory)
0x1018_0000	0x8000_0000	0x3000	rw-	Attr4=04 (Device memory)
0x1018_8000	0x8000_0000	0x1000	rw-	Attr6=04 (Device memory)
0xC000_0000	0x2_0000_0000	0x1_0000_0000	rw-	Attr0=04 (Device memory)

Initialization

- Store CPU tick count at 0x1013_0FC0
- Set SP=0x1013_0400
- Set VBAR to 0x1000_4000
- Set CPSR.M to 0x13 (Supervisor mode)
  - Clear LR and SP for other modes
- Set all registers to 0xdeadbeef
- Clear memory 0x1013_0000 to 0x1013_0D80
  - 0x1013_0000 ~ 0x1013_0400 is stack
  - 0x1013_0400 ~ 0x1013_0D80 is bss data
- Copy data from 0x1001_3D80 to 0x1013_0D80 (size is 0xF00-0xD80)
- Call main logic function

```
ROM:10004CDC                LDR             R2, =0x10130FC0
ROM:10004CDE                STRD.W          R0, R1, [R2],#4
ROM:10004CE2                LDR.W           SP, =0x10130400
ROM:10004CE6                BL              set_VBAR
ROM:10004CEA                BL              set_DF_IF
ROM:10004CEE                BL              config_CPSR
ROM:10004CF2                BL              clear_CPSR_A
ROM:10004CF6                BL              config_CPACR
ROM:10004CFA                BL              clear_registers
ROM:10004CFE                BL              bzero_stack
ROM:10004D02                BL              copy_data
ROM:10004D06                BL              main_function
ROM:10004D0A                BL              infinite_loop
```

Main Function

- Initialize some peripherals
- Clear CPU tick counts stored in 0x1013_0FC0
  - 8 slots in total (0x40 bytes)
- Read some settings from fuses
- If it’s fresh boot
  - Initialize TRNG
  - Generate 4 bytes stack cookie
- boot() function will start to talk to AP via mailbox
  - It shall never return

```
for ( i = (void (__cdecl **)())&off_10011000; i < dword_10011004; ++i )
{
  v2 = *i;
  v2();                                         // sub_10005ED4
}
sub_10004954();
reset_cpu_counts();
sub_10004900();
init_fuses();
if ( is_fresh_boot() == 1 )
{
  init_TRNG();
  generate_random(&stack_cookie, 4);
}
store_cpu_count_by_idx(5u);
boot();
panic(76, v0);
```

Mailbox

- The only communication channel between SEP and AP
- AP functions
  - akf_start/akf_send_mbox/akf_recv_mbox/akf_stop
  - Type of SEP is 2
  - 8 bytes message could be sent and received
- Underlying implementation
  - Read/write of mapped IO registers accessible for both sides
  - Base address of AP: 0x2_0DA0_4000
  - Base address of SEP: VA 0xCDA0_0B80 (PA 0x2_0DA0_0B80)

IO Registers of Mailbox

Offset	Size	Description	Using
0x0	4 Bytes	Disable interrupt	Mask all interrupts in start (IN/OUT Empty/Nonempty)
0x4	4 Bytes	Enable interrupt	Enable OUT Nonempty in receiving / Enable IN Empty in sending
0x8	4 Bytes	Inbox status	Enable/Full/Empty/Overflow/Underflow
0x10	8 Bytes	Inbox value for AP	AP write message to
0x18	8 Bytes	Outbox value for SEP	SEP read message from
0x20	4 Bytes	Outbox status	Enable/Full/Empty/Overflow/Underflow
0x30	8 Bytes	Inbox value for SEP	SEP write message to
0x38	8 Bytes	Outbox value for AP	AP read message from

Messages of Mailbox

- Each message could contain 4 bytes data
- When talking to SEPROM, endpoint=255
- Message with opcode>100 is sent from SEP to AP
  - If a message is successfully handled, SEP will send reply message with opcode+100

```
struct __attribute__((packed)) sep_message
{
  unsigned __int8 endpoint;
  unsigned __int8 tag;
  unsigned __int8 opcode;
  unsigned __int8 param;
  unsigned int data;
};
```

Opcode	Description	Param	Data
1/2	Ping		
3	Generate nonce		
4	Report nonce	Index of nonce to get (0-4)	4 bytes of nonce (reply)
5	Boot TZ0		
6	Boot SEPOS	0 (normal) / 1 (restore)	(PA>>12) of where SEP image4 data is
7	Send ART data		(PA>>12) of where ART data is
8	Resume		
9	Set flag		0 or 1
10	Panic immediatly		
14	TEST mode?		
15	Seed DPA		
16	Generate random data		4 bytes random data (reply)
201	Status		1 or 2
255	Report panic		

Internal use on special device?

Added in A9

boot()

- Clear out nonce values (20 bytes)
- Initialize mailbox
- Enter message handler loop for first time
  - Break the loop after receiving “boot tz0” message
- boot_tz0() will setup protection for external RAM
  - Now SEP has sufficient memory to load OS
- Enter message handler loop for second time
  - Break the loop after receiving “boot sepos” message
- Continue to verify and load SEPOS

```
if ( is_fresh_boot() )
{
  clear_nonce();
  init_mailbox();
  send_msg_status_1();
  pa = 0LL;
  if ( handle_message(&pa, 1) == 5 )
  {
    if ( sub_10005FCC() || sub_1000604C() == 1 )
      panic(71, v0);
    handle_opcode_0xe(pa & 0x3FF);
    v1 = (void (*)(void))((pa >> 10) & 0xFFFFFFFC);
    __mcr(15, 0, (unsigned int)v1, 12, 0, 0);
    __isb(0xFu);
    v1();
  }
  else
  {
    if ( !sub_1000617C() )
      panic(90, v0);
    if ( sub_10006188() == 1 )
      panic(91, v0);
  }
  if ( sub_10006130() == 1 )
    sub_100045A0(30);
  boot_tz0();
  send_msg_status_2();
  v2 = handle_message(&pa, 0);
  if ( v2 != 3 )
  {
    if ( (unsigned int)(v2 - 1) <= 1 )
      load_sepos(pa, HIDWORD(pa));
    panic(120, v0);
  }
  resume();
}
sub_1000A178();
```

Message Loop

- The loop simply waits for a message to come in from AP and do proper action according to the opcode
- The 1st stage will handle following opcodes
  - 1/2/3/4/5/8/9/10/14/15/16
- The 2nd stage will handle following opcodes
  - 1/2/3/4/6/7/9/10/15/16
- AP ROM and iBoot only send ping and nonce related messages
- Kernel will send “boot tz0”/“boot sepos”/“ART”/“resume” to actually load SEPOS

Boot TZ0

TZ0

- AP needs to set TZ0/TZ1 registers before sending “boot tz0”
  - Three 4-bytes registers(base/end/lock) for each trust zone
    - Base/end registers = (PA>>12) & 0x3F_FFFF
      - Physical memory address of iPhone starts from 0x8_0000_0000
        - e.g. 0x8_7D80_0000 -> 0x7_D800
    - Once lock register is set, base/end registers can not be set again
    - A7 uses one register for base & end
      - ((end PA>>20)<<16 | (base PA>>20)) & 0x3FFF_3FFF
  - TZ1 is only needed for devices running KPP at EL3
- TZ0/TZ1 registers belong to AMCC (Apple’s Memory Cache Controller)

Memory Protection

- SEP needs more memory to load OS
  - TZ0 describes the memory region will be used by SEP
- Protecting SEP memory
  - Isolation
    - AMCC will stop AP from accessing TZ0 memory once it’s locked
  - Encryption
    - Transparent encryption/decryption by inline AES engine
  - Integrity
    - Checksum of encrypted memory

Setup TZ0

- iBoot will set TZ0/TZ1 registers before jumping to kernel entry
- TZ0 base/end registers are at 0x2_0000_0480/0x2_0000_0484
- TZ1 base/end registers are at 0x2_0000_0488/0x2_0000_048C
- Lock registers are at 0x2_0000_0490/0x2_0000_0494
- Values from testing

	TZ0 base	TZ0 end	TZ1 base	TZ1 end
A7	0x5_0000		0x3F7_03F7	
A8	0x0	0xBFF	0x3_F580	0x3_F5FF
A9	0x7_D800	0x7_E5FF	0x7_D780	0x7_D7FF
A10	0x7_D200	0x7_DDFF	-	-
A11	0xB_0FD8	0xB_C3D7	-	-

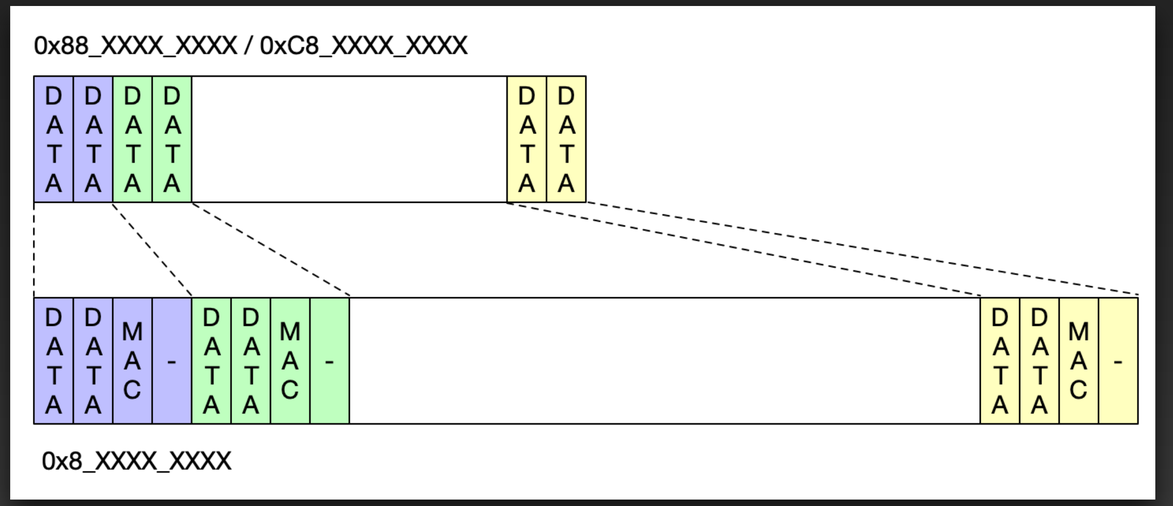
```
MEMORY[0x200000480] = *v4 >> 12;
MEMORY[0x200000484] = (*v4 + v4[1] - 1) >> 12;
MEMORY[0x200000488] = *v5 >> 12;
MEMORY[0x20000048C] = (*v5 + v5[1] - 1) >> 12;
MEMORY[0x200000490] = 1;
v2 = sub_87001AA84(v6);
if ( (MEMORY[0x200000490] & 1) == 0
   || (MEMORY[0x200000494] = 1, v2 = sub_87001AA84(v2), (MEMORY[0x200000494] & 1) == 0) )
{
LL_30:
  sub_870011B50(v2);
  sub_87001613C(&unk_870046380, "%llx:%d", v18, v19);
}
```

boot_tz0()

- Read TZ0 registers and make sure it’s locked
- Set memory ranges for encryption
- Generate AES keys and active encryption
  - Only setup encryption channel for SEPROM
- Zero out memory to be used for new page tables
- Setup new page tables and switch TTBR0
  - New TTBR0 is PA 0x2_0D90_0DA0 (VA 0x1013_0DA0)
- Map more memory

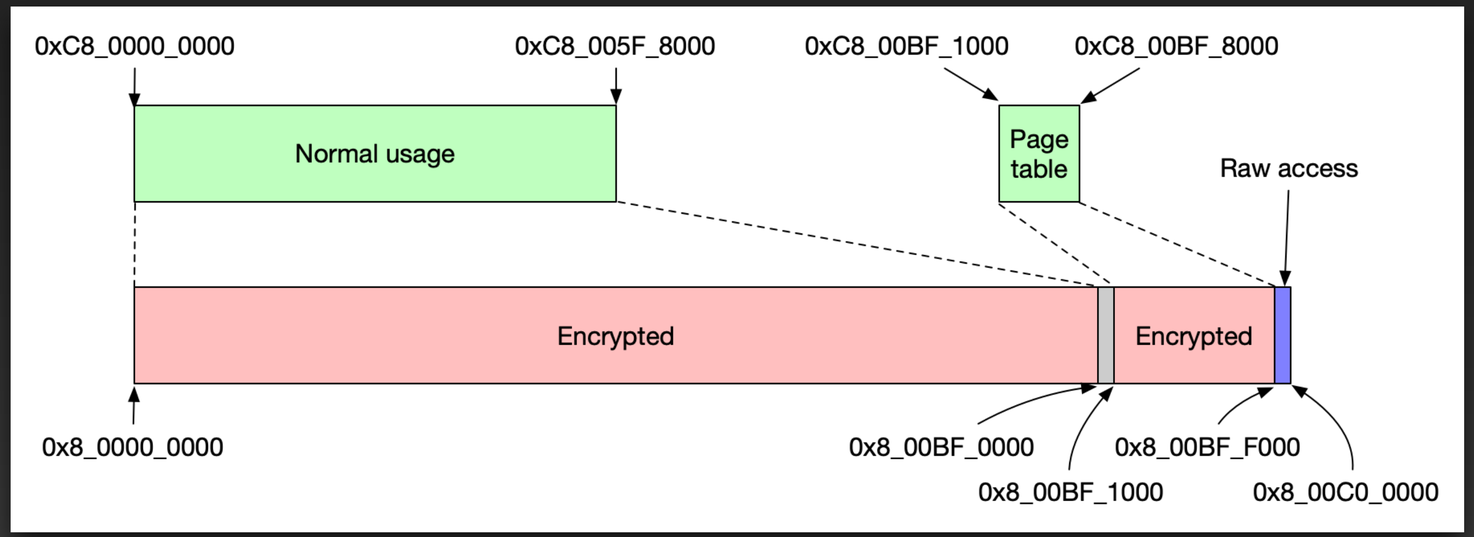
Transparent Encryption

- AES-256-XEX(XTS) mode
  - It’s designed for disk encryption
  - Require two 32 bytes keys and 16 bytes sector (no IV required)
  - Each block is 16 bytes
  - Apple adds 16 bytes MAC for every two blocks
- Keys are generated by patterns include 24 random bytes per boot
- Two encryption channels are supported
  - SEPROM is accessing PA 0xC8_XXXX_XXXX
  - SEPOS is using 0x88_XXXX_XXXX instead
- Raw encrypted memory is at PA 0x8_XXXX_XXXX
  - Useful memory size is only half of total memory



Memory Regions

- External memory usage
  - Construct 40 bits PA from TZ0 registers
    - PA = 0x8_0000_0000 + (value << 12)
    - A8 TZ0 register: 0/0xBFF -> 0x8_0000_0000 to 0x8_00C0_0000
  - Divide memory into 3 parts



Memory Mapping

- SEP needs to setup new page tables in external memory
  - BUT current page tables are immutable
- How to access external memory at first place?
  - Remember VA 0x1018_000 is already mapped to PA 0x8000_0000
  - Looks like that SEP supports to map one PA to another PA

```
phy_mem_map(0x80000000LL, v1 + 0x80000000LL - v0, v2 + v0);
phy_mem_map_enable();
memset(0x10180084, 0, 0xF5Cu);
```

```
MEMORY[0xCDA00020] = HIDWORD(pa);                // pa=0x8000_0000
MEMORY[0xCDA00018] = pa;
MEMORY[0xCDA00030] = (unsigned __int64)(pa_end - 4096) >> 32;// pa_end=0x8000_1000
MEMORY[0xCDA00028] = pa_end - 4096;
MEMORY[0xCDA00010] = (unsigned __int64)(new_pa - pa) >> 32;// new_pa=0x8_00BF_F000
MEMORY[0xCDA00008] = new_pa - pa;
```

New Page Tables

- After memory protection is turned on, SEP starts to setup new page tables
  - It maps 0xC8_00BF_3000 to 0x8000_0000 (VA 0x1018_0000) with 3 pages for storing new page tables
  - New 1st-level table is at 0x1013_0DA0 (dedicated memory)
    - The first entry now points to 0xC8_00BF_3000
  - Copy current page tables from 0x1000_1000 to 0x1018_0000 (3 pages)
  - Clear entries of VA 0x4000
  - Map VA 0x101F_0000 to PA 0xC8_00BF_3000 (3 pages)
    - Easy to modify page tables later
  - Switch TTBR0 to 0x2_0D90_0DA0
  - Map more VAs: 0x1014_0000/0x1012_E000/0x1016_0000

New Virtual Memory Layout

VA	PA	Size	Access	AttrIndx
~~0x4000~~	~~0x2_0D00_4000~~	~~0x1000~~	~~r-x~~	~~Attr0=04 (Device memory)~~
0x1000_0000	0x2_0D00_0000	0x2_0000	r-x	Attr3=4F (Normal memory)
0x1012_E000	0xC8_00BF_1000	0x2000	rw-	Attr1=4F (Normal memory)
0x1013_0000	0x2_0D90_0000	0x1000	rw-	Attr5=4F (Normal memory)
0x1014_0000	0x8_00BF_F000	0x1000	rw-	Attr1=4F (Normal memory)
0x1016_0000	0xC8_00BF_7000	0x1000	rw-	Attr1=4F (Normal memory)
0x1018_0000	0x8000_0000	0x3000	rw-	Attr4=04 (Device memory)
0x1018_8000	0x8000_0000	0x1000	rw-	Attr6=04 (Device memory)
0x101F_0000	0xC8_00BF_3000	0x3000	rw-	Attr1=4F (Normal memory)
0xC000_0000	0x2_0000_0000	0x1_0000_0000	rw-	Attr0=04 (Device memory)

Load SEPOS

- After boot TZ0, SEP is ready for loading SEPOS
- It will copy IMG4 firmware into its own protected memory
- Decode IMG4 file and do verification
- Generate AES keys again for channel used by SEPOS (0x88_XXXX_XXXX)
- Decrypt the payload
- Setup bootarg and jump to SEP firmware (+0x0)

Attack Surfaces

- Mapped IO registers
- SEP messages
- IMG4 parser

Attack Secure Boot

Memory Isolation

- With checkm8 exploit, we are capable of patching iBoot & executing arbitrary code
  - Test logics of AMCC
- Try to read TZ0 memory after it's locked
  - Return all zero
- SEP ROM external memory region (by reverse engineering)
  - 0x8_0000_0000+((uint32_t)TZ0_base<<12) ~ 0x8_0000_0000+((uint32_t)TZ0_end<<12)+0x1000
- AMCC prevents AP from visiting TZ0 memory (guessing)
  - 0x8_0000_0000+((uint32_t or uint64_t?)TZ0_base<<12) ~ 0x8_0000_0000+((uint32_t or uint64_t?)TZ0_end<<12)+0x1000

WHAT IF

TZ0_base += 0x10_0000

TZ0_end += 0x10_0000

Bypass Memory Isolation

- A8 TZ0 register: 0/0xBFF -> 0x10_0000/0x10_0BFF
- Try to read TZ0 memory again after “boot tz0” 🤩

```
0x800bf3000 mem: 0x1df5733326d36dc3 | 0x6eaa44ebefca46b4 | 0x96ca2eb6c4162c2b | 0xc616aa4d2853d3b3
0x800bf3020 mem: 0x796b2d1af7c49752 | 0xab17730f21e27436 | 0x0000000000000000 | 0x0000000000000000
0x800bf3040 mem: 0x19aecfc73d7a0d98 | 0x95ff3a2c2d9d493d | 0x19f492ccc2a45f0a | 0xaab68785940617e5
0x800bf3060 mem: 0x5ad2af07dee8798d | 0x2eac03f5c9b5aa28 | 0x0000000000000000 | 0x0000000000000000
0x800bf3080 mem: 0x89fadd5ee8d99fb0 | 0xafbe9906cd4364dd | 0x67cc9e379a4c2257 | 0x11b4c5be9e23c7c6
0x800bf30a0 mem: 0xc7fd8174b000eb7a | 0x01859f7201e20740 | 0x0000000000000000 | 0x0000000000000000
0x800bf30c0 mem: 0x22e94fa6ac149dd3 | 0xf86127ad65906803 | 0x1563da4a3ef09fa7 | 0x48c14d0d2f32e7c5
0x800bf30e0 mem: 0xb649911ea03747b4 | 0x4bd353b50a6a5f9d | 0x0000000000000000 | 0x0000000000000000
```

- SEP ROM will have same external memory
  - 0x8_0000_0000 ~ 0x8_00C0_0000
- AMCC may try protect memory (cast to uint64_t)
  - 0x9_0000_0000 ~ 0x9_00C0_0000

Test More Devices

- A8/A9/A10
  - SEP can boot successfully and AP can access SEP memory
- A7 (boot_tz0 fail immediately)
  - panic(cpu 0 caller 0xffffffff022896c9c): "AMC Error! AMC_IRERRDBG_STS=0x1020000, MCCTAGPARLOG1/2=0/0, MCCDATERRLOG=0, MCCAFERRLOG0/1=0x5ff070/0xf1409, CH0RNKCFG0/1=0x1/0, CH1RNKCFG0/1=0x1/0"@/BuildRoot/Library/Caches/com.apple.xbs/Sources/AppleS5L8960X_kext/AppleS5L8960X-159/AppleS5L8960XPlatformErrorHandler.cpp:145
  - SEPROM handles TZ0 values as uint64_t -> SEP access invalid memory address
- A11 (timeout - no status 2 reply after boot_tz0)
  - panic(cpu 0 caller 0xffffffff0232bb748): "SEP Boot Failure: status check 2 failed - 0xe00002d6”@/Library/Caches/com.apple.xbs/Sources/AppleSEPManager/AppleSEPManager-553.120.4/AppleSEPBooter.cpp:210
  - SEPROM may be vulnerable, but it hangs somewhere (due to some checks?)
  - No binary to do further investigation 🥺

Next Move

- We could now read/write SEP memory
  - BUT the memory is encrypted and verified
    - Replay attack is possible but not easy
  - Remember two pages are not encrypted
    - 0x8_00BF_0000 and 0x8_00BF_F000
    - Anything interesting to overwrite?

Generate AES Keys

- boot_tz0() first maps 0x8_00BF_F000 to 0x8000_0000 with one page
  - At this time VA 0x1018_0000 points to raw memory 0x8_00BF_F000
  - It then generates two 24 random bytes to derive AES keys
  - The random bytes are stored at following addresses
    - 0x1018_0018 / 0x1018_0048
      - Possible to race it from AP !
- A10 seems to have 0x1_0000 bytes dedicated memory
  - Page tables are no longer on external memory
  - The random bytes are stored on stack
  - No chance to race it

```
generate_tz0_random_bytes(1, 1);
generate_and_set_key(1u, 1);
generate_tz0_random_bytes(3, 1);
generate_and_set_key(3u, 1);
```

```
switch ( a1 )
{
  case 0:
    result = 0x10140000;
    if ( a2 )
      result = 0x10180000;
    break;
  case 1:
    result = 0x10140018;
    if ( a2 )
      result = 0x10180018;
    break;
  case 2:
    result = 0x10140030;
    if ( a2 )
      result = 0x10180030;
    break;
  case 3:
    result = 0x10140048;
    if ( a2 )
      result = 0x10180048;
    break;
  default:
    panic(58, v2);
}
```

Control SEPROM Memory

- Patch iBoot to add TZ0 registers with 0x10_0000
- Send “boot tz0” message
- Keep writing const values at 0x8_00BF_F018/0x8_00BF_F048 for a while
  - Same random bytes -> same AES keys
- Try to read data at 0x8_00BF_3000
  - Reboot and adjust race time until we get same encrypted data

Control SEPOS Memory

- SEPOS is using another encryption channel
  - AES keys are generated again when received “boot sepos” message
  - This time random bytes are stored at 0x1014_0000/0x1014_0030
  - Try race it
    - Failed 😑
    - Issue for racing memory between two CPUs - CACHE
      - 0x1014_0000 is marked as normal memory
      - 0x1018_0000 is marked as device memory
      - Cache enabled -> SEP can not be aware of memory modification from AP

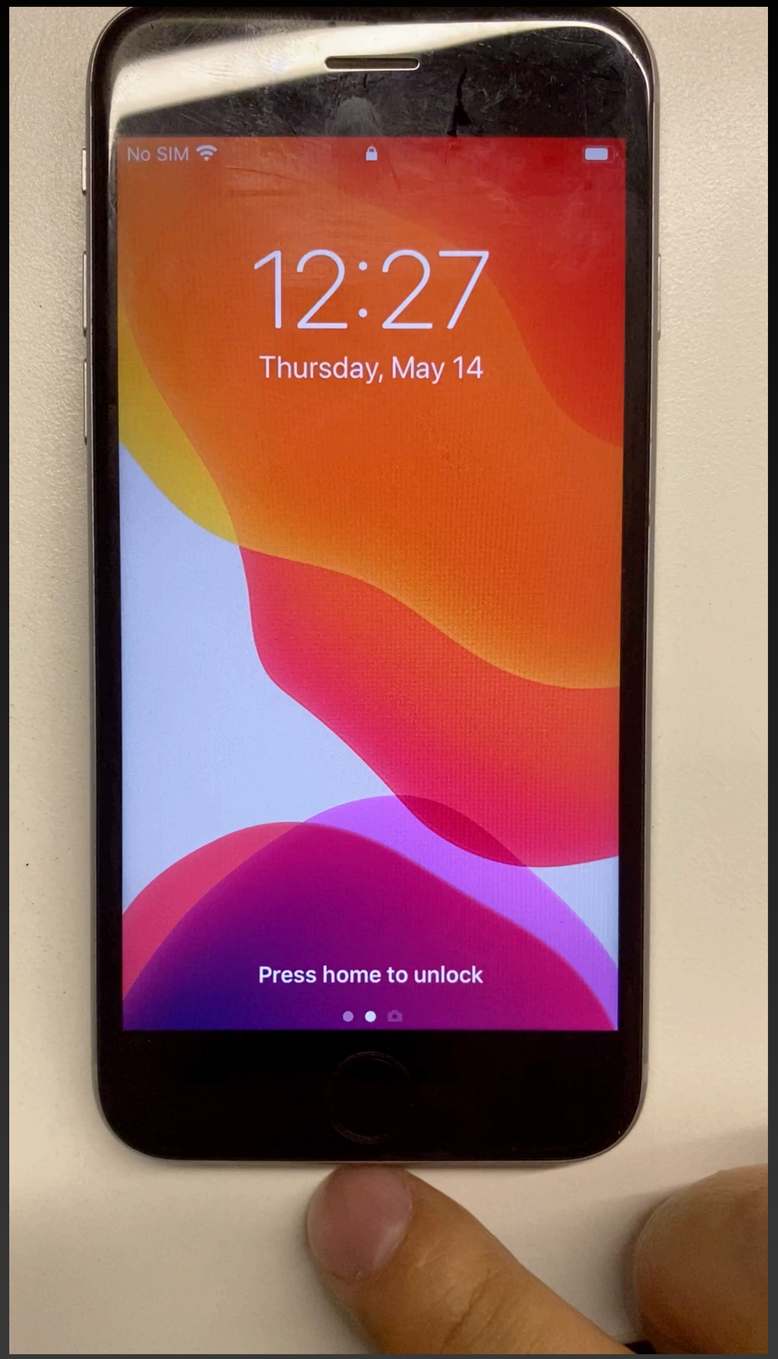
Enlarge Attack Surface

- We could force SEPROM (A8/A9) to use fixed AES keys
- SEP firmware IMG4 file will be copied into external memory
  - Send arbitrary firmware -> SEP panics -> dump encrypted memory
    - We could read encrypted memory of any plain data
- Before jumping to SEPOS, SEP will switch to another encryption channel
  - There is a time window for us to race SEP firmware memory
  - Big attack surface, many choices
    - Same target -> load arbitrary SEPOS
    - Choosing racy objects/writing exploit/testing -> huge pain 🤯🤯🤯

Review

- TZ0 memory isolation bypass
  - A8/A9/A10 are vulnerable
  - A7/A11 are not vulnerable
    - But failure results are different
- Random bytes (for deriving key) are stored on external memory
  - A7/A8/A9
- Consequences
  - A8/A9 - Load arbitrary SEPOS
  - A10- Replay attack

Demo



One More Page

- After I gave the talk at MOSEC 2020
  - An attendee told me that A8 SEPROM is not checking TZ0 lock !
  - And I found out this is TRUE 🤣
    - Patch iBoot not to set TZ0 lock -> accessing SEP memory
- The very first version (A7) actually handles TZ0 registers most correctly
- My exploit to load arbitrary SEPOS is gentle
- Looking forward to see more ways to exploit it 😜

Thanks & QA