Demystifying the Secure Enclave Processor

Tarjei Mandt (@kernelpool)
Mathew Solnik (@msolnik)
David Wang (@planetbeing)

OFFCELL RESEARCH

azimuth SECURITY

About Us

- Tarjei Mandt
  - Senior Security Researcher, Azimuth Security
  - tm@azimuthsecurity.com
- Mathew Solnik
  - Director, OffCell Research
  - ms@offcellresearch.com
- David Wang
  - Senior Security Researcher, Azimuth Security
  - dw@azimuthsecurity.com

Introduction

- iPhone 5S was a technological milestone
  - First 64-bit phone
- Introduced several technological advancements
  - Touch ID
  - M7 motion coprocessor
  - Security coprocessor (SEP)
- Enabled sensitive data to be stored securely
  - Fingerprint data, cryptographic keys, etc.

Secure Enclave Processor

- Security circuit designed to perform secure services for the rest of the SOC
  - Prevents main processor from gaining direct access to sensitive data
- Used to support a number of different services
  - Most notably Touch ID
- Runs its own operating system (SEPOS)
  - Includes its own kernel, drivers, services, and applications

Secure (?) Enclave Processor

- Very little public information exists on the SEP
  - Only information provided by Apple
- SEP patent only provides a high level overview
  - Doesn't describe actual implementation details
- Several open questions remain
  - What services are exposed by the SEP?
  - How are these services accessed?
  - What privileges are needed?
  - How resilient is SEP against attacks?

Talk Outline

- Part 1: Secure Enclave Processor
  - Hardware Design
  - Boot Process
- Part 2: Communication
  - Mailbox Mechanism
  - Kernel-to-SEP Interfaces
- Part 3: SEPOS
  - Architecture / Internals
- Part 4: Security Analysis
  - Attack Surface and Robustness

Hardware Design

Demystifying the Secure Enclave Processor

SEP’s ARM Core: Kingfisher

- Dedicated ARMv7a “Kingfisher” core
  - Even EL3 on AP’s core won’t doesn’t give you access to SEP
- Appears to be running at 300-400mhz~
- One of multiple kingfisher cores in the SoC
  - 2-4 Other KF cores - used for NAND/SmartIO/etc
  - Other cores provide a wealth of arch knowledge
- Changes between platforms (A7/A8/A9)
  - Appears like anti-tamper on newer chips

Dedicated Hardware Peripherals

- SEP has its own set of peripherals accessible by memory-mapped IO
  - Built into hardware that AP cannot access
    - Crypto Engine & Random Number Generator
    - Security Fuses
    - GID/UID Keys
- Dedicated IO lines -
  - Lines run directly to off-chip peripherals
    - GPIO
    - SPI
    - UART
    - I2C

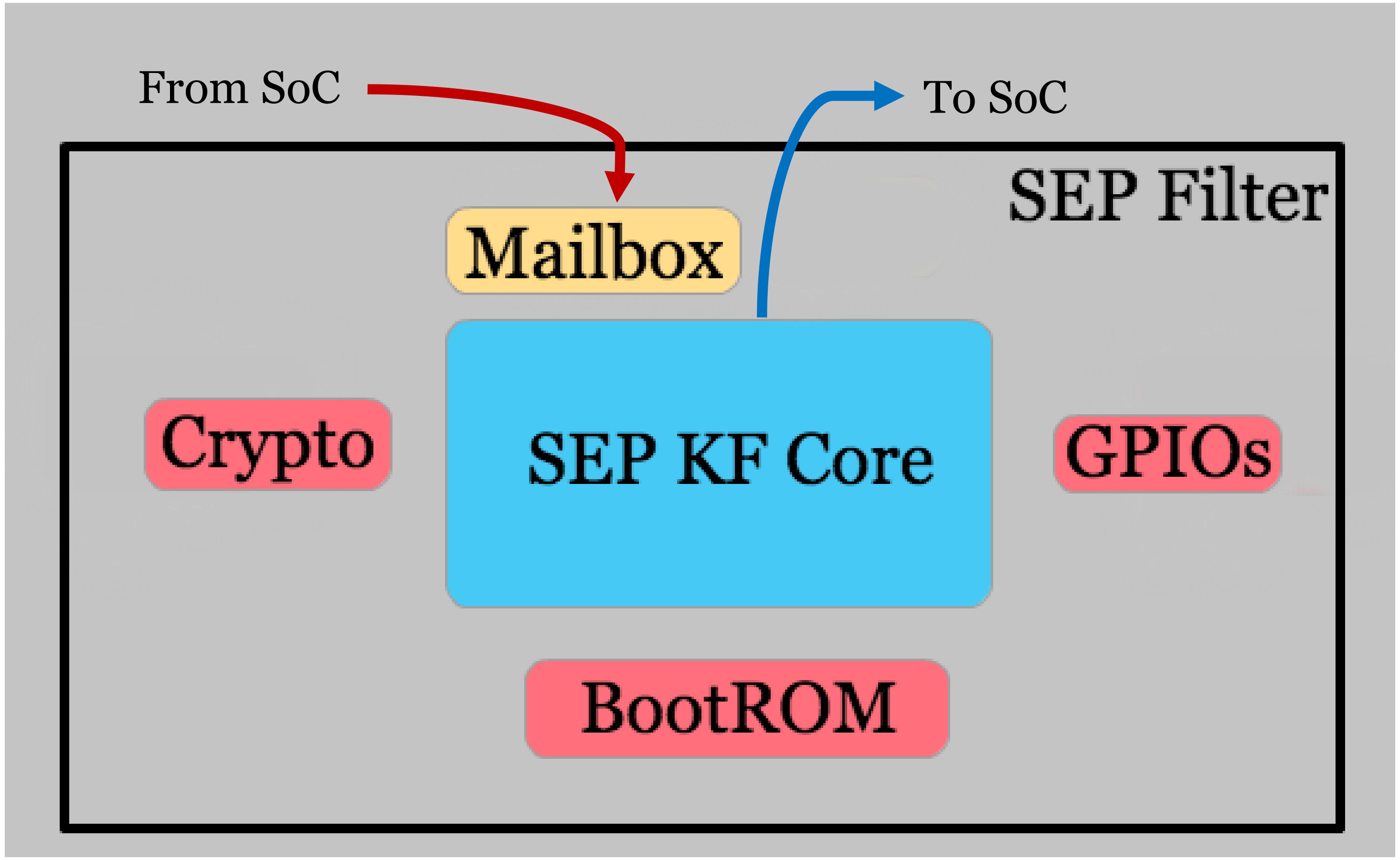
Shared Hardware Peripherals

- SEP and AP share some peripherals
- Power Manager (PMGR)
  - Security fuse settings are located in the PMGR
  - Lots of other interesting items
- Memory Controller
  - Can be poked at via iOS kernel
- Phase-locked loop (PLL) clock generator
  - Nothing to see here move along…
- Secure Mailbox
  - Used to tranfer data between cores
- External Random Access Memory (RAM)

Physical Memory

- Dedicated BootROM (and some SRAM)
  - BootROM physically located at 0x2_0da0_0000
- Uses inline AES to encrypt external RAM
  - Most likely to prevent physical memory attacks against off SoC RAM chips (iPads)
- Hardware “filter” to prevent AP to SEP memory access
  - Only SEP’s KF core has this filter

SEP KF Filter Diagram



Boot Process

Demystifying the Secure Enclave Processor

SEP Initialization – First Stage

- AP comes out of reset. AP BootROM releases SEP from reset.
  - This is irreversible. No hardware register to reset or stop SEP accessible by AP.
- Initially uses 4096 bytes of static RAM for stack and variables.
- Uses page tables in ROM.
  - Needs Large Physical Address Extension.
- Starts a message loop.

SEP Initialization – Second Stage

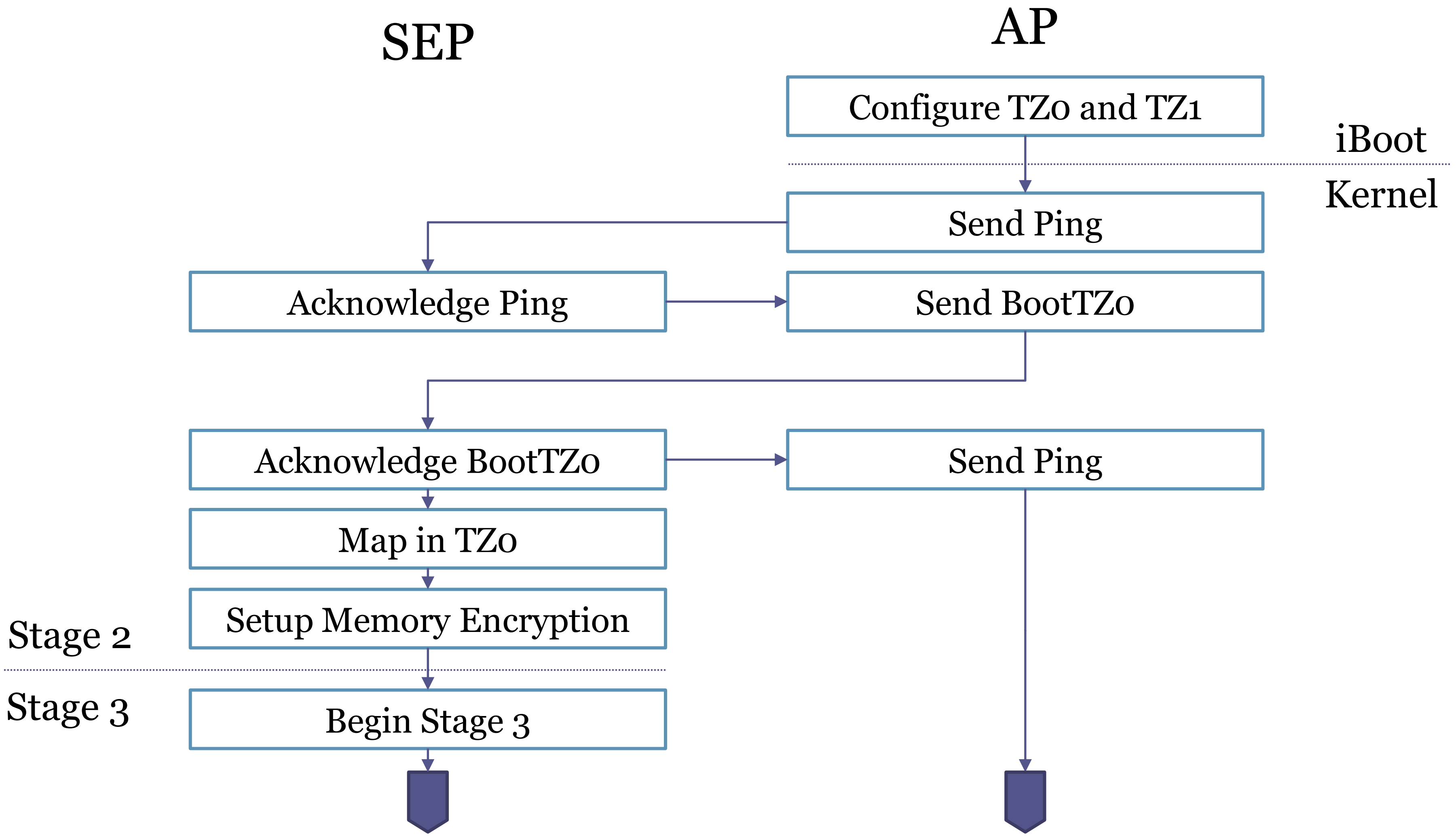
- Listens for messages in the mailbox.
- 8-byte messages that have the same format SEPOS uses.
- All messages use endpoint 255 (EP_BOOTSTRAP)

Opcode	Description
1, 2	"Status check" (Ping)
3	Generate nonce
4	Get nonce word
5	"BootTZ0" (Continue boot)

Memory Protections

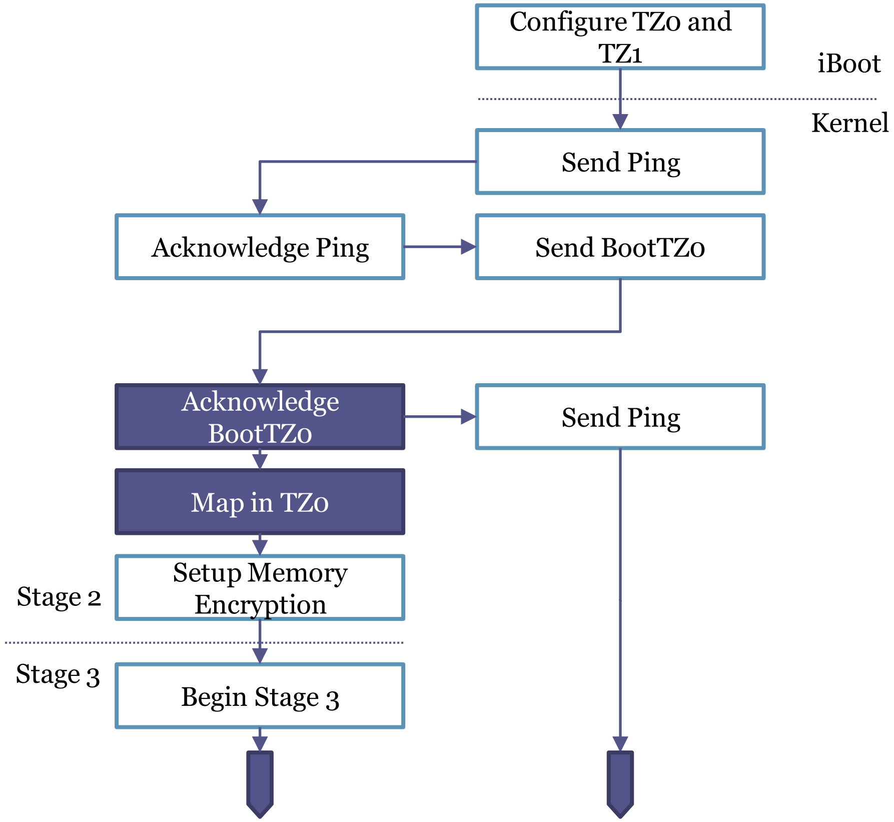
- SEP needs more RAM than 4096 bytes of SRAM, so it needs external RAM.
- RAM used by SEP must be protected against AP tampering.
- Two regions configurable by AP are setup:
  - TZ0 is for the SEP.
  - TZ1 is for the AP's TrustZone (Kernel Patch Protection).
- SEP must wait for AP to setup TZ0 to continue boot.

SEP Boot Flow



SEP Memory Protection Bootstrap

- SEP doesn’t take AP’s word for it that TZ0 is locked.
  - Checks hardware registers for lock.
  - Then reads size and address of TZ0 from other hardware registers.
- Impossible to change these hardware registers after TZ0 is locked.
- Spin processor on failure.



Memory Encryption Modes

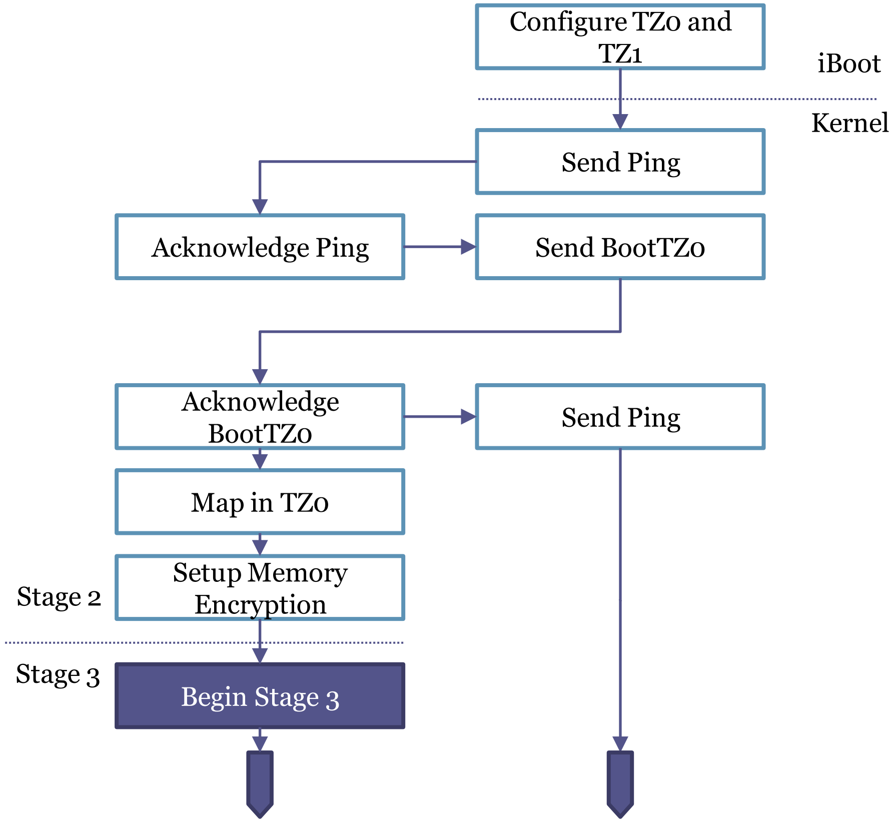
- Appears to support ECB, CBC, and **XEX**.
- Capable of AES-128 or **AES-256**.
- Supports two channels.
  - BootROM uses channel 1.
  - SEPOS uses channel 0.
- All access to certain ranges of physical addresses get encrypted/decrypted transparently.
  - After boot, SEPOS has all page mappings into the encrypted range (except for hardware regs and memory shared with AP).

Key Generation

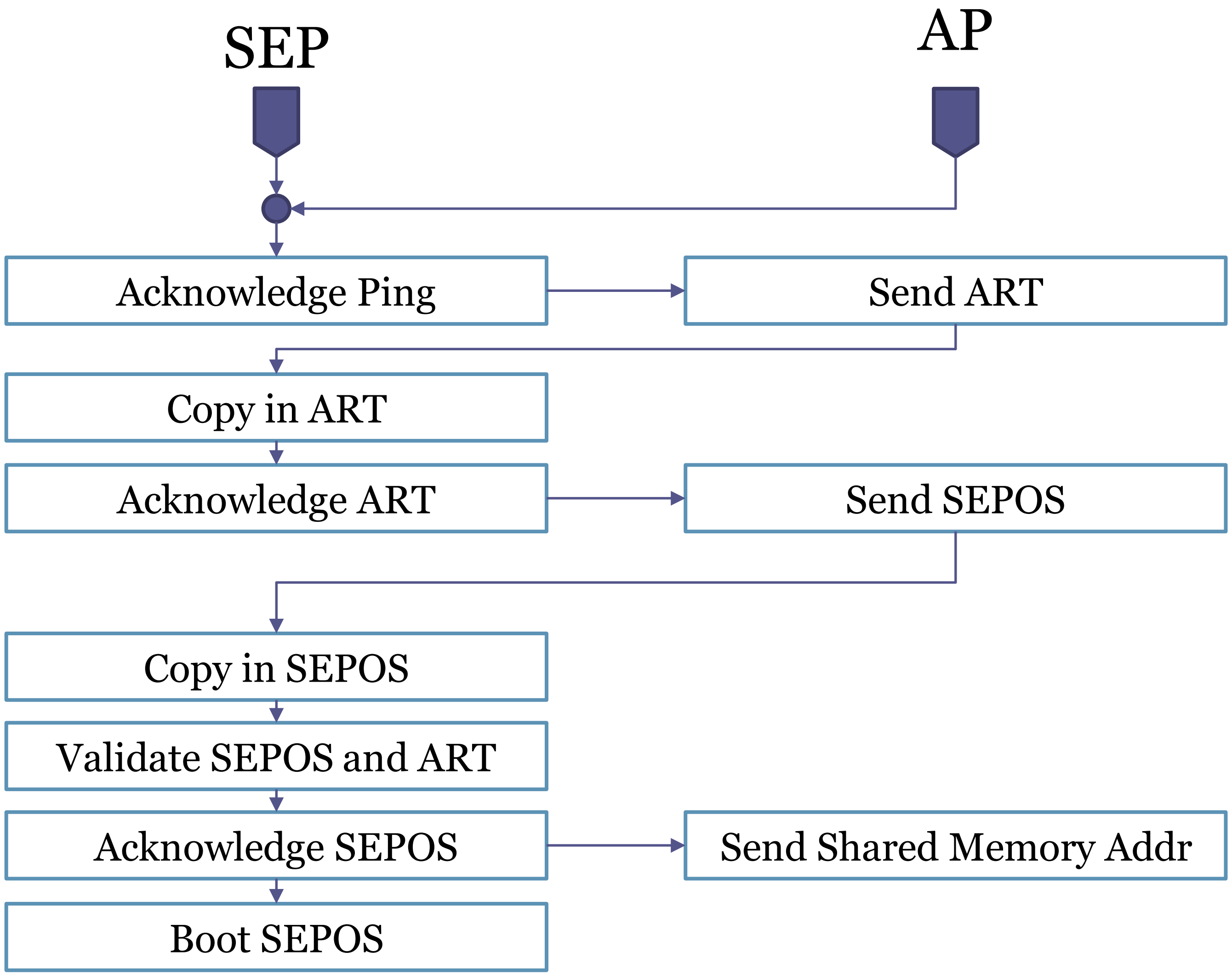
- Keys are generated by “tangling”:
  - True Random Number Generator output
  - Static ”type” value.
- With protected (unreadable) registers:
  - UID, GID, Seed A, Seed B.
    - Seed B tangled with UID == GenID_2B
- Encrypt the following using GenID_2B to generate key:
  - [4 byte magic = 0xFF XK1][4 bytes of 0s][192-bits of randomness]

Beginning Stage 3

- After memory encryption is setup, SEP re-initializes to use encrypted memory:
  - Page tables
  - Stack
  - Data
- Begins a new message loop with no shared code between it and the initial low-capability bootstrap.



SEP Boot Flow: Stage 3



Boot-loading: Img4

- SEP uses the “IMG4” bootloader format which is based on ASN.1 DER encoding
  - Very similar to 64bit iBoot/AP Bootrom
  - Can be parsed with ”openssl -asn1parse”

- Three primary objects used by SEP
  - Payload –
    - Contains the encrypted sep-firmware
  - Restore –
    - Contains basic information when restoring SEP
  - Manifest (aka the AP ticket) -
    - Effectively the Alpha and the Omega of bootROM configuration (and security)

Img4 - Manifest

- The manifest (APTicket) contains almost all the essential information used to authenticate and configure SEP(OS).
- Contains multiple hardware identifier tags
  - ECID
  - ChipID
  - Others
- Is also used to change runtime settings in both software and hardware
  - DPRO – Demote Production
  - DSEC – Demote Security
  - Others…

Reversing SEP's Img4 Parser: Stage 1

- How can you reverse something you cannot see?
  - Look for potential code reuse!
- Other locations that parse IMG4
  - AP BootROM – A bit of a pain to get at
  - iBoot – Dump from phys memory - 0x8700xx000
    - Not many symbols…
    - But sometimes it only takes 1…

```
X8, #aImg4decodecopy@PAGE ; "Img4DecodeCopyManifestHash((const Img4 "...
X8, X8, #aImg4decodecopy@PAGEOFF ; "Img4DecodeCopyManifestHash((const Img4
X8, [SP,#0x3C0+var_3A8]
X8, #0x187
loc_83D8099B4
```

(iBoot from n51)

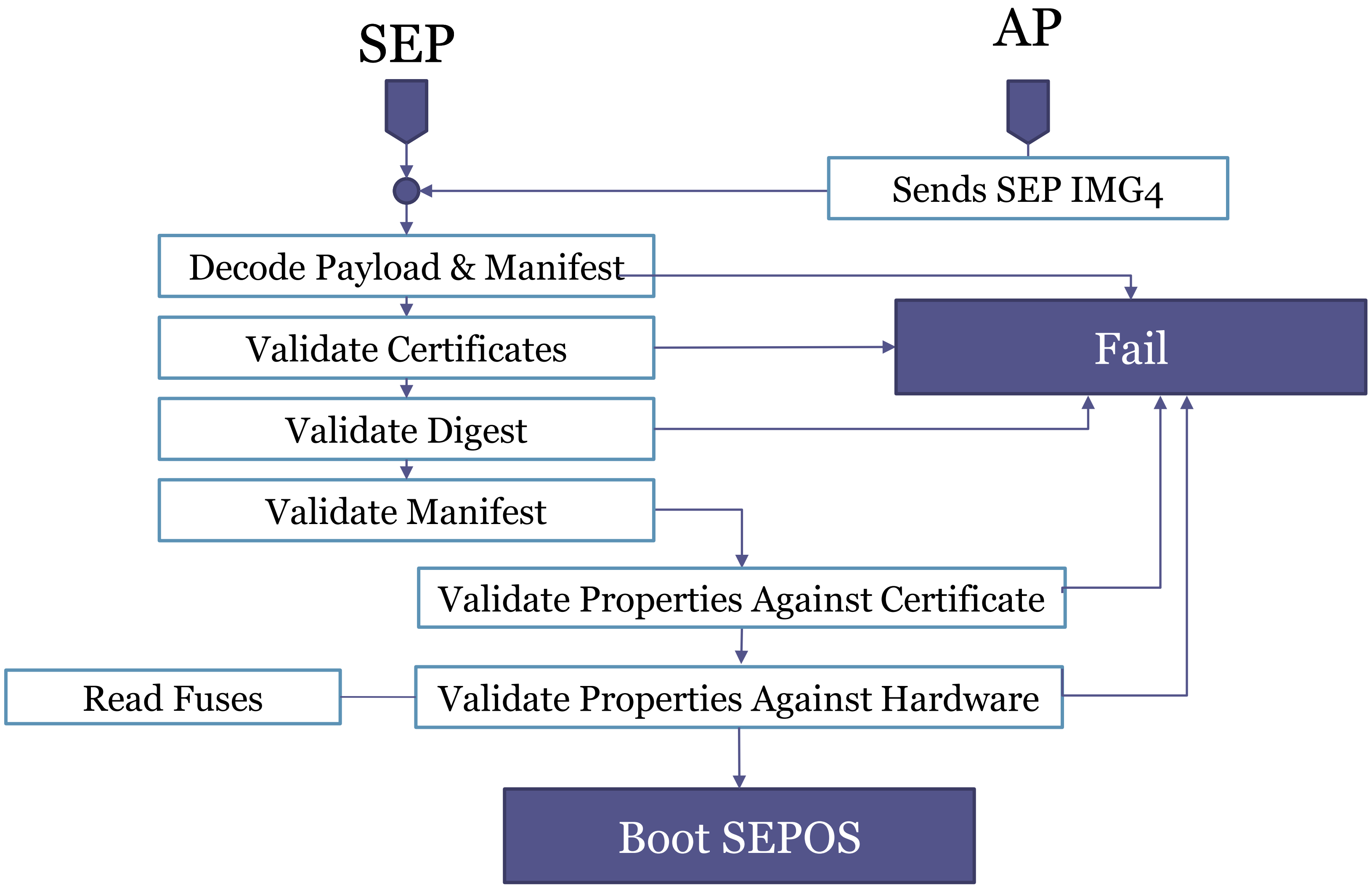
Reversing SEP's Img4 Parser: Stage 2

- Another file also contains the "Img4Decode" symbol
  - /usr/libexec/seputil
- Userland IMG4 parser with many more symbols
  - May not be exact – but bindiff shows it is very close
- From symbols found in seputil we can deduce:
  - The ASN'1 decoder is based on libDER
    - Which Apple so kindly releases as OpenSource.
  - The RSA portion is handled by CoreCrypto
- LibDER + CoreCrypto = SEP's IMG4 Parsing engine
  - We now have a great base to work with

Img4 Parsing Flow

- SEP BootROM copies in the sep-firmware.img4 from AP
- Initializes the DER Decoder
  - Decodes Payload, Manifest, and Restore Info
- Verifies digests and signing certificates
  - Root of trust cert is hardcoded at the end of BootROM
- Verifies all properties in manifest
  - Checks against current hardware fusing
- If all items pass – load and execute the payload

Img4 Parsing Flow



Communication

Demystifying the Secure Enclave Processor

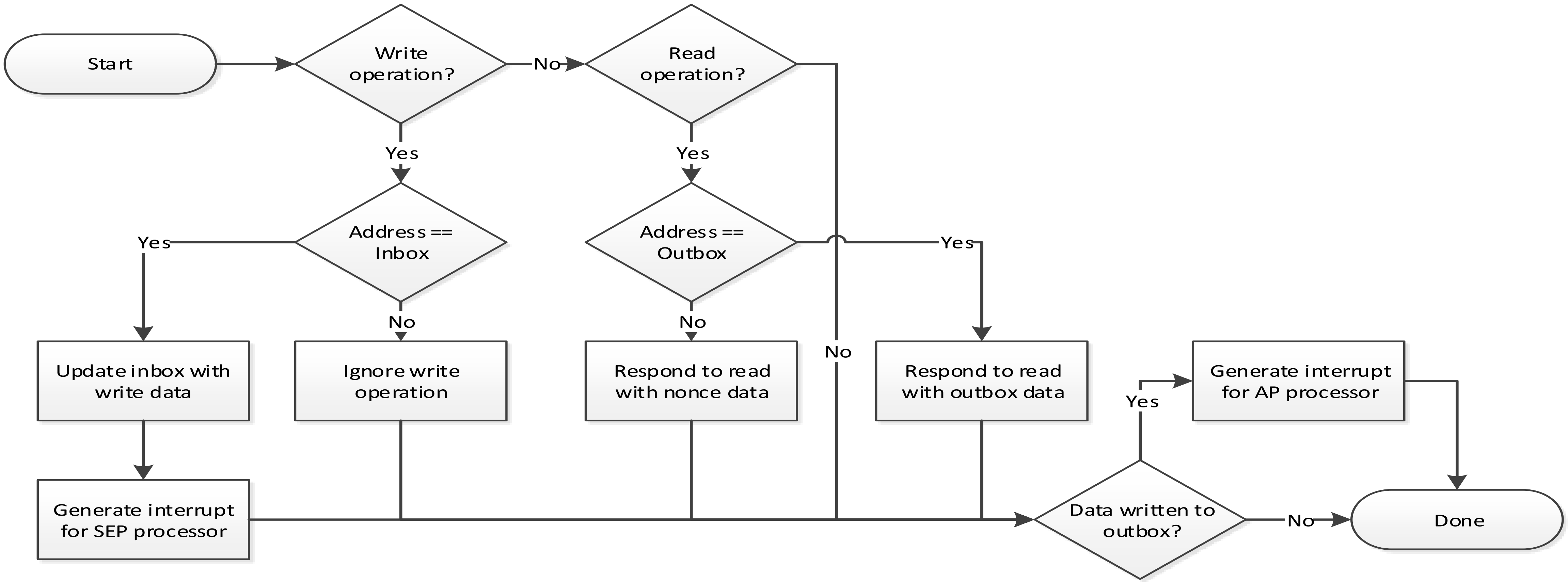
Secure Mailbox

- The secure mailbox allows the AP to communicate with the SEP
  - Features both an inbox (request) and outbox (reply)
- Implemented using the SEP device I/O registers
  - Also known as the SEP configuration space

Interrupt-based Message Passing

- When sending a message, the AP writes to the inbox of the mailbox
- This operation triggers an interrupt in the SEP
  - Informs the SEP that a message has been received
- When a reply is ready, the SEP writes a message back to the outbox
  - Another interrupt is generated in order to let the AP know a message was received

Mailbox Mechanism



Mailbox Message Format

- A single message is 8 bytes in size
- Format depends on the receiving endpoint
- First byte is always the destination endpoint

```
struct {
    uint8_t    endpoint;      // destination endpoint number
    uint8_t    tag;           // message tag
    uint8_t    opcode;        // message type
    uint8_t    param;         // optional parameter
    uint32_t   data;          // message data
} sep_msg;
```

SEP Manager

- Provides a generic framework for drivers to communicate with the SEP
  - Implemented in AppleSEPManager.kext
  - Builds on the functionality provided by the IOP
- Enables drivers to register SEP endpoints
  - Used to talk to a specific SEP app or service
  - Assigned a unique index value
- Also implements several endpoints of its own
  - E.g. the SEP control endpoint

SEP Endpoints (1/2)

Index	Name	Driver
0	AppleSEPControl	AppleSEPManager.kext
1	AppleSEPLogger	AppleSEPManager.kext
2	AppleSEPARTStorage	AppleSEPManager.kext
3	AppleSEPARTRequests	AppleSEPManager.kext
4	AppleSEPTracer	AppleSEPManager.kext
5	AppleSEPDebug	AppleSEPManager.kext
6	<not used>	
7	AppleSEPKeyStore	AppleSEPKeyStore.kext

SEP Endpoints (2/2)

Index	Name	Driver
8	AppleMesaSEPDriver	AppleMesaSEPDriver.kext
9	AppleSPIBiometricSensor	AppleBiometricSensor.kext
10	AppleSEPCredentialManager	AppleSEPCredentialManager.kext
11	AppleSEPPairing	AppleSEPManager.kext
12	AppleSSE	AppleSSE.kext
254	L4Info	
255	Bootrom	SEP Bootrom

Control Endpoint (EP0)

- Handles control requests issued to the SEP
- Used to set up request and reply out-of-line buffers for an endpoint
- Provides interface to generate, read, and invalidate nonces
- The SEP Manager user client provides some support for interacting with the control endpoint
  - Used by the SEP Utility (/usr/libexec/seputil)

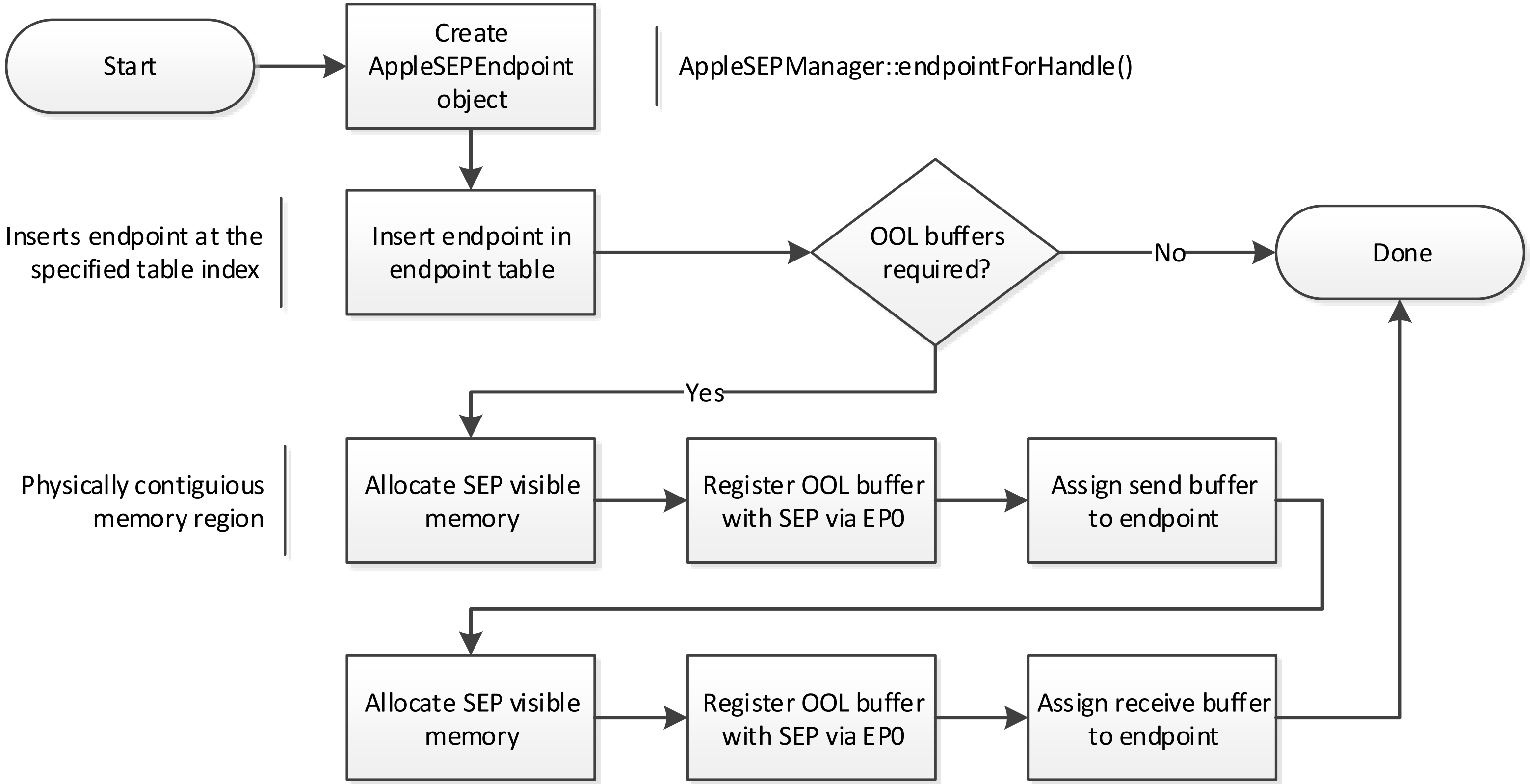
Control Endpoint Opcodes

Opcode	Name	Description
0	NOP	Used to wake up SEP
2	SET_OOL_IN_ADDR	Request out-of-line buffer address
3	SET_OOL_OUT_ADDR	Reply out-of-line buffer address
4	SET_OOL_IN_SIZE	Size of request buffer
5	SET_OOL_OUT_SIZE	Size of reply buffer
10	TTYIN	Write to SEP console
12	SLEEP	Sleep the SEP

Out-of-line Buffers

- Transferring large amounts of data is slow using the interrupt-based mailbox
  - Out-of-line buffers used for large data transfers
- SEP Manager provides a way to allocate SEP visible memory
  - AppleSEPManager::allocateVisibleMemory(...)
  - Actually allocates a portion of physical memory
- Control endpoint is used to assign the request/reply buffer to the target endpoint

Endpoint Registration (AP)



Drivers Using SEP

- Several drivers now rely on the SEP for their operation
- Some drivers previously located in the kernel have had parts moved into the SEP
  - Apple(SEP)KeyStore
  - Apple(SEP)CredentialManager
- Most drivers have a corresponding app in the SEP

SEPOS

Demystifying the Secure Enclave Processor

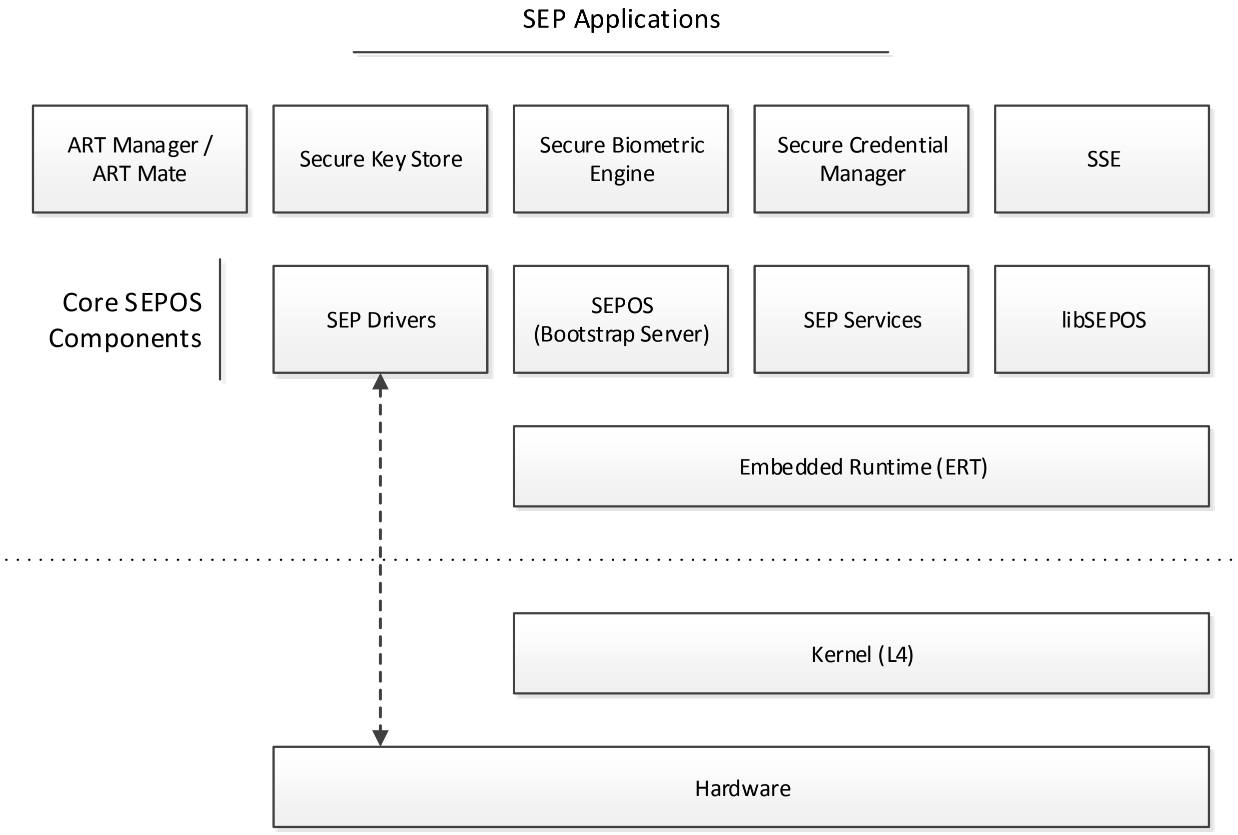
L4

- Family of microkernels
- First introduced in 1993 by Jochen Liedtke
  - Evolved from L3 (mid-1980s)
- Developed to address the poor performance of earlier microkernels
  - Improved IPC performance over L3 by a factor 10-20 faster
- Numerous variants and implementations
  - E.g. L4-embedded optimized for embedded systems

SEPOS

- Based on Darbat/L4-embedded (ARMv7)
  - Custom modifications by Apple
- Implements its own drivers, services, and applications
  - Compiled as macho binaries
- The kernel provides only a minimal set of interfaces
  - Major part of the operating system implemented in user-mode

SEPOS Architecture



Kernel (L4)

- Initializes the machine state to a point where it is usable
  - Initializes the kernel page table
  - Sets up the kernel interface page (KIP)
  - Configures the interrupts on the hardware
  - Starts the timer
  - Initializes and starts the kernel scheduler
  - Starts the root task
- Provides a small set (~20) of system calls

System Calls (1/2)

Num	Name	Description
0x00	L4_Ipc	Set up IPC between two threads
0x00	L4_Notify	Notify a thread
0x04	L4_ThreadSwitch	Yield execution to thread
0x08	L4_ThreadControl	Create or delete threads
0x0C	L4_ExchangeRegisters	Exchange registers wit another thread
0x10	L4_Schedule	Set thread scheduling information
0x14	L4_MapControl	Map or free virtual memory
0x18	L4_SpaceControl	Create a new address space
0x1C	L4_ProcessorControl	Sets processor attributes

System Calls (2/2)

Num	Name	Description
0x20	L4_CacheControl	Cache flushing
0x24	L4_IpcControl	Limit ipc access
0x28	L4_InterruptControl	Enable or disable an interrupt
0x2C	L4_GetTimebase	Gets the system time
0x30	L4_SetTimeout	Set timeout for ipc sessions
0x34	L4_SharedMappingControl	Set up a shared mapping
0x38	L4_SleepKernel	?
0x3C	L4_PowerControl	?
0x40	L4_KernelInterface	Get information about kernel

Privileged System Calls

- Some system calls are considered privileged
  - E.g. memory and thread management calls
- Only root task (SEPOS) may invoke privileged system calls
  - Determined by the space address of the caller
- Check performed by each individual system call where needed
  - is_privileged_space()

Privileged System Calls

```
SYS_SPACE_CONTROL (threadid_t space_tid, word_t control, fpage_t kip_area,
            fpage_t utcb_area)
{
    TRACEPOINT (SYSCALL_SPACE_CONTROL,
          printf("SYS_SPACE_CONTROL: space=%t, control=%p, kip_area=%p, "
                 "utcb_area=%p\n",  TID (space_tid),
                 control, kip_area.raw, utcb_area.raw));

    // Check privilege
    if (EXPECT_FALSE (! is_privileged_space(get_current_space())))
    {
          get_current_tcb ()->set_error_code (ENO_PRIVILEGE);
          return_space_control(0, 0);
    }

    ...
}
```

Check for root task in L4_SpaceControl system call

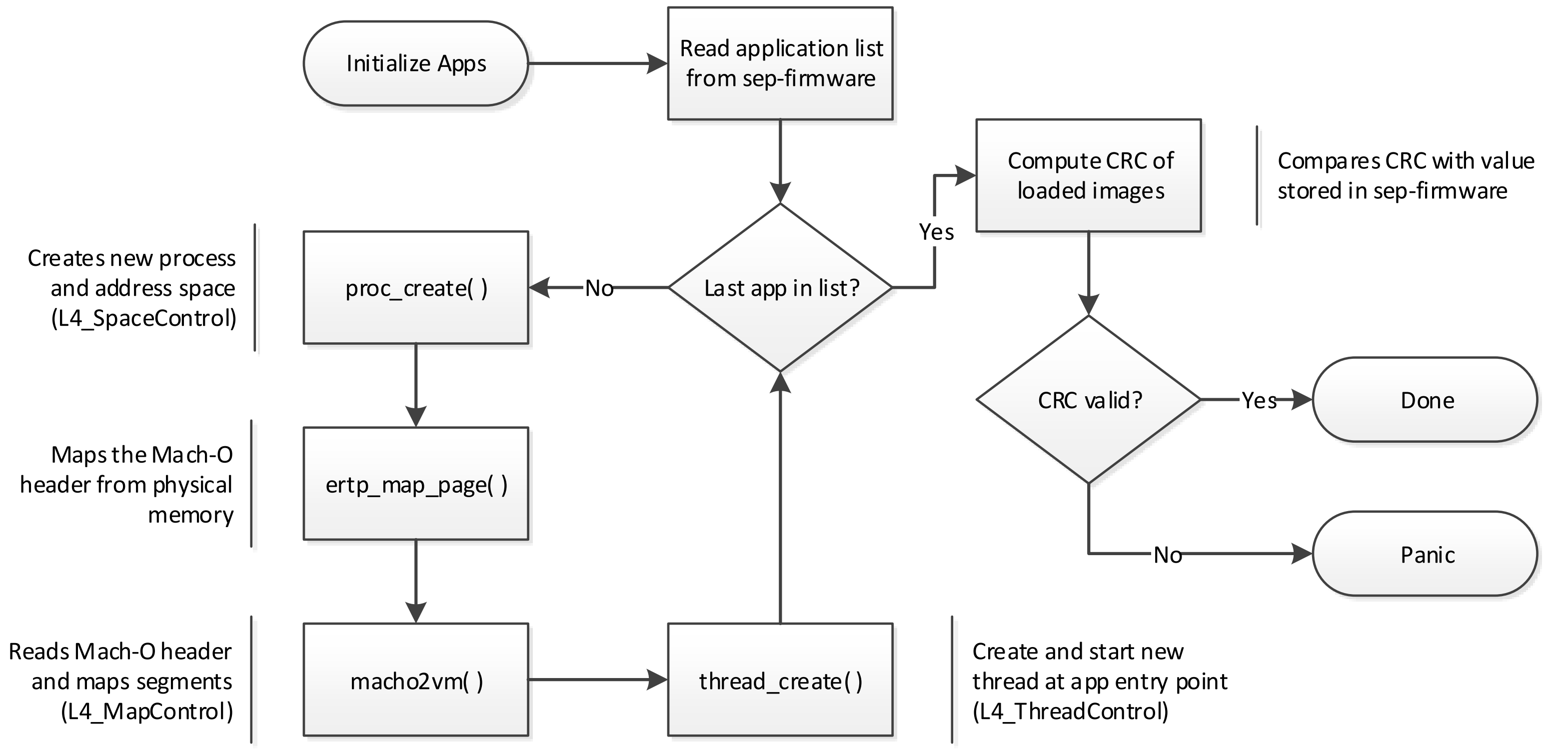
```
INLINE bool is_privileged_space(space_t *
space)
{
    return (is_roottask_space(space);
}
```

from darbat 0.2 source

SEPOS (INIT)

- Initial process on boot (root task)
  - Can call any privileged L4 system call
- Initializes and starts all remaining tasks
  - Processes an application list embedded by the sep-firmware
- Maintains a context structure for each task
  - Includes information about the virtual address space, privilege level, threads, etc.
- Invokes the bootstrap server

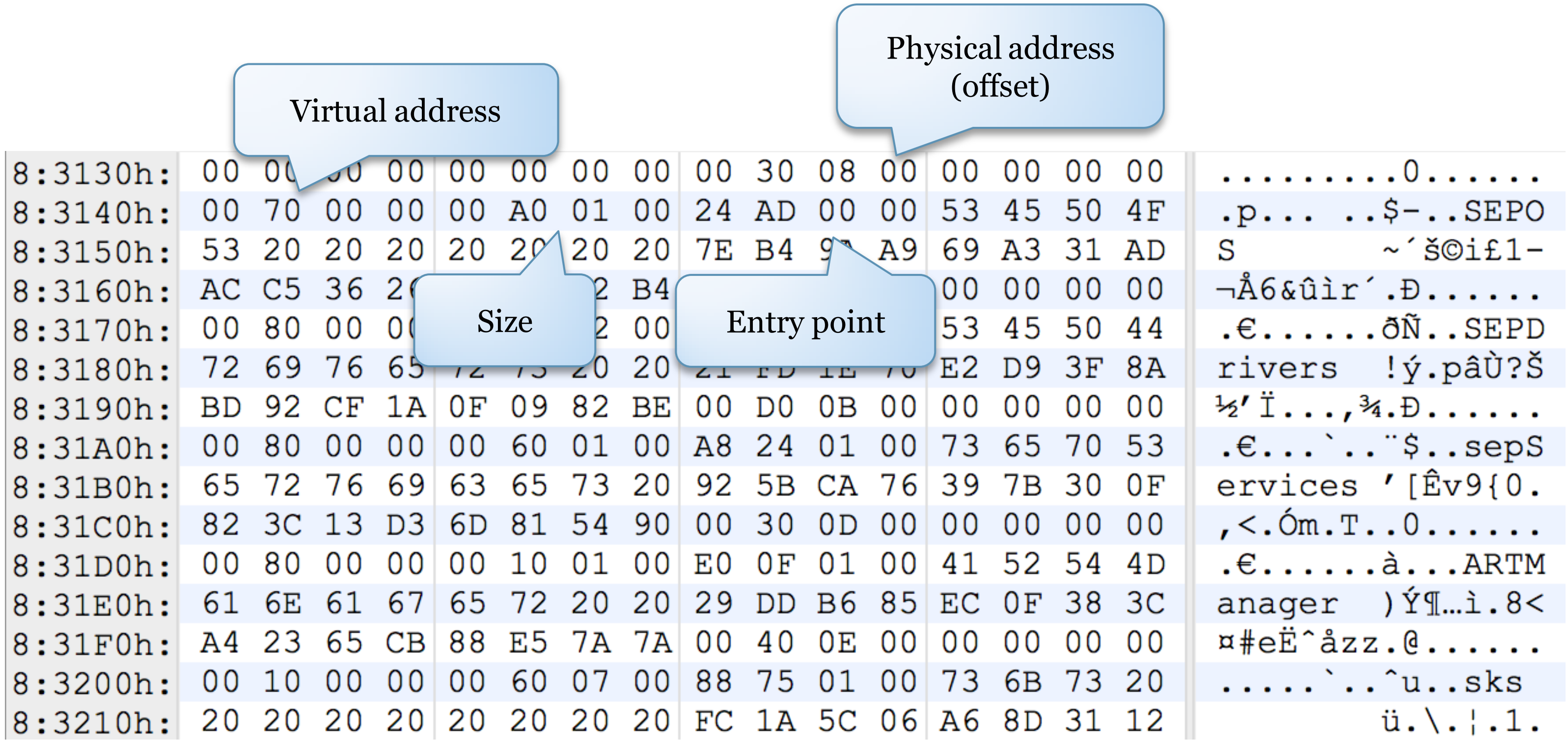
SEPOS App Initialization



Application List

- Includes information about all applications embedded by the SEP firmware
  - Physical address (offset)
  - Virtual base address
  - Module name and size
  - Entry point
- Found 0xEC8 bytes prior to the SEPOS binary in the sep-firmware image

Application List



Bootstrap Server

- Implements the core functionality of SEPOS
  - Exports methods for system, thread and object (memory) management
- Made available to SEP applications over RPC via the embedded runtime
  - ert_rpc_bootstrap_server()
- Enable applications to perform otherwise privileged operations
  - E.g. create a new thread

Privileged Methods

- An application must be privileged to invoke certain bootstrap server methods
  - Query object/process/acl/mapping information
- Privilege level is determined at process creation
  - Process name >= 'A ' and <= 'ZZZZ'
  - E.g. "SEPD" (SEPDrivers)
- Check is done by each individual method
  - proc_has_privilege(int pid);

sepos_object_acl_info( )

```
int sepos_object_acl_info(int *args)
{
  int result;
  int prot;
  int pid;

  args[18] = 1;
  *((_BYTE *)args + 104) = 1;
  result = proc_has_privilege( args[1] );
  if ( result == 1 )
  {
    result = acl_get( args[5], args[6], &pid, &prot);
    if ( !result )
    {
      args[18] = 0;
      args[19] = prot;
      args[20] = pid;
      result = 1;
      *((_BYTE *)args + 104) = 1;
    }
  }
  return result;
}
```

Call to check if sender's pid is privileged

Entitlements

- Some methods also require special entitlements
  - sepos_object_create_phys()
  - sepos_object_remap()
- Seeks to prevent unprivileged applications from mapping arbitrary physical memory
- Assigned to a process on launch
  - Separate table used to determine entitlements

Entitlement Assignment

```
int proc_create( int name )
{
    int privileged = 0;

    ...

    if ( ( name >= 'A   ' ) && ( name <= 'ZZZZ' ) )
         privileged = 1;

    proctab[ pid ].privileged   = privileged;
    proctab[ pid ].entitlements= 0;

    while ( privileged_tasks[ 2 * i ] != name )
         if ( ++i == 3 )
              return pid;

    proctab[ pid ].entitlements= privileged_tasks[ 2 * i + 1 ];

    return pid;
}
```

```
; _DWORD privileged_tasks[10]
privileged_tasks DCD 'SEPD'
; int[]
                 DCD 2
                 DCD 'ARTM'
                 DCD 6
                 DCD 'Debu'
                 DCD 6
                 DCD 0
                 DCD 0
```

Entitlement Assignment

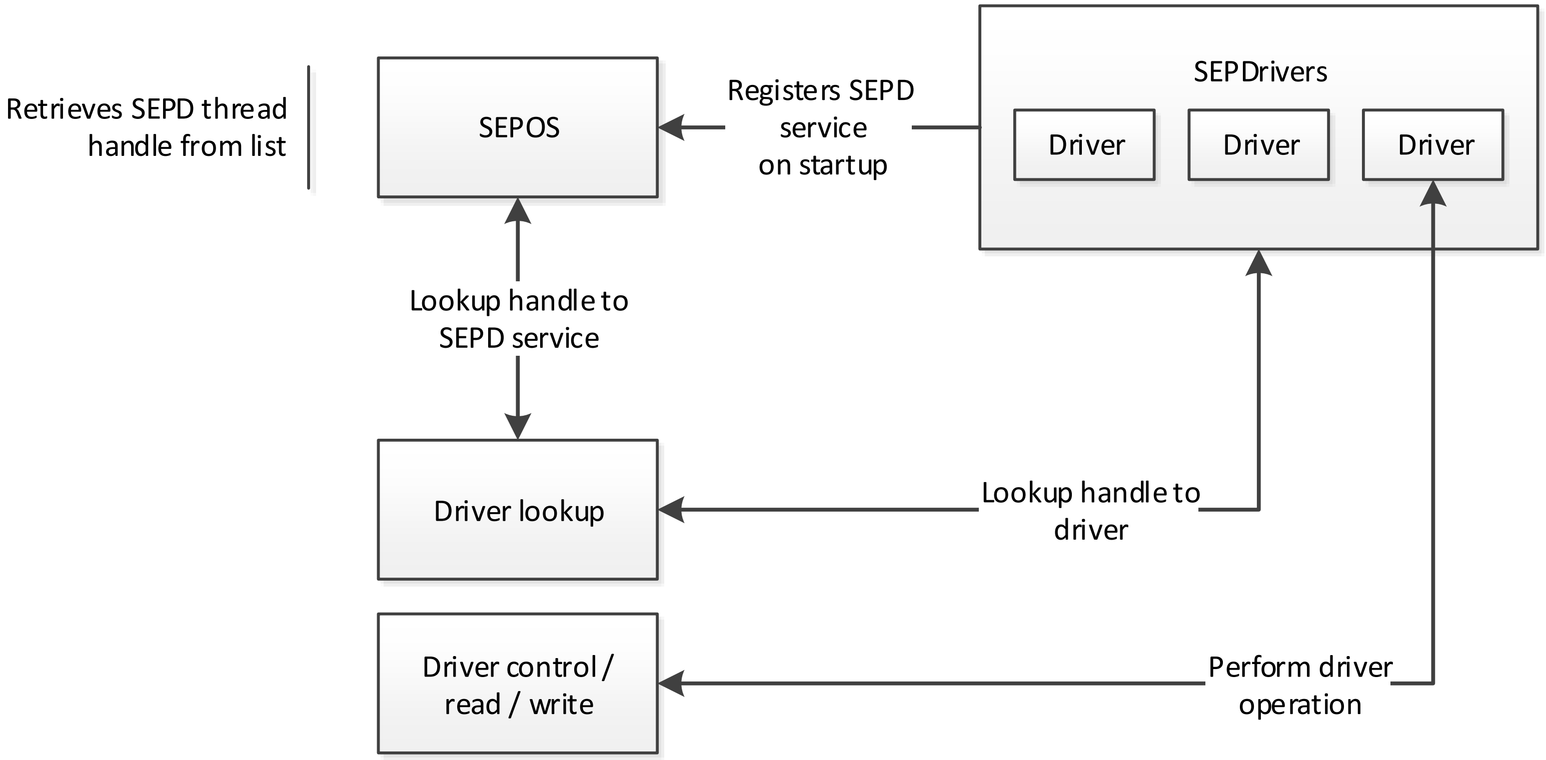
Task Name	Entitlements
SEPDrivers	MAP_PHYS
ARTManager/ARTMate	MAP_PHYS | MAP_SEP
Debug	MAP_PHYS | MAP_SEP

- MAP_PHYS (2)
  - Required in order to access (map) a physical region
- MAP_SEP (4)
  - Same as above, but also needed if the physical region targets SEP memory

SEP Drivers

- Hosts all SEP drivers
  - AKF, TRNG, Expert, GPIO, PMGR, etc.
  - Implemented entirely in user-mode
- Maps the device I/O registers for each driver
  - Enables low-level driver operations
- Exposed to SEP applications using a dedicated driver API
  - Includes functions for lookup, control, read, and write

Driver Interaction



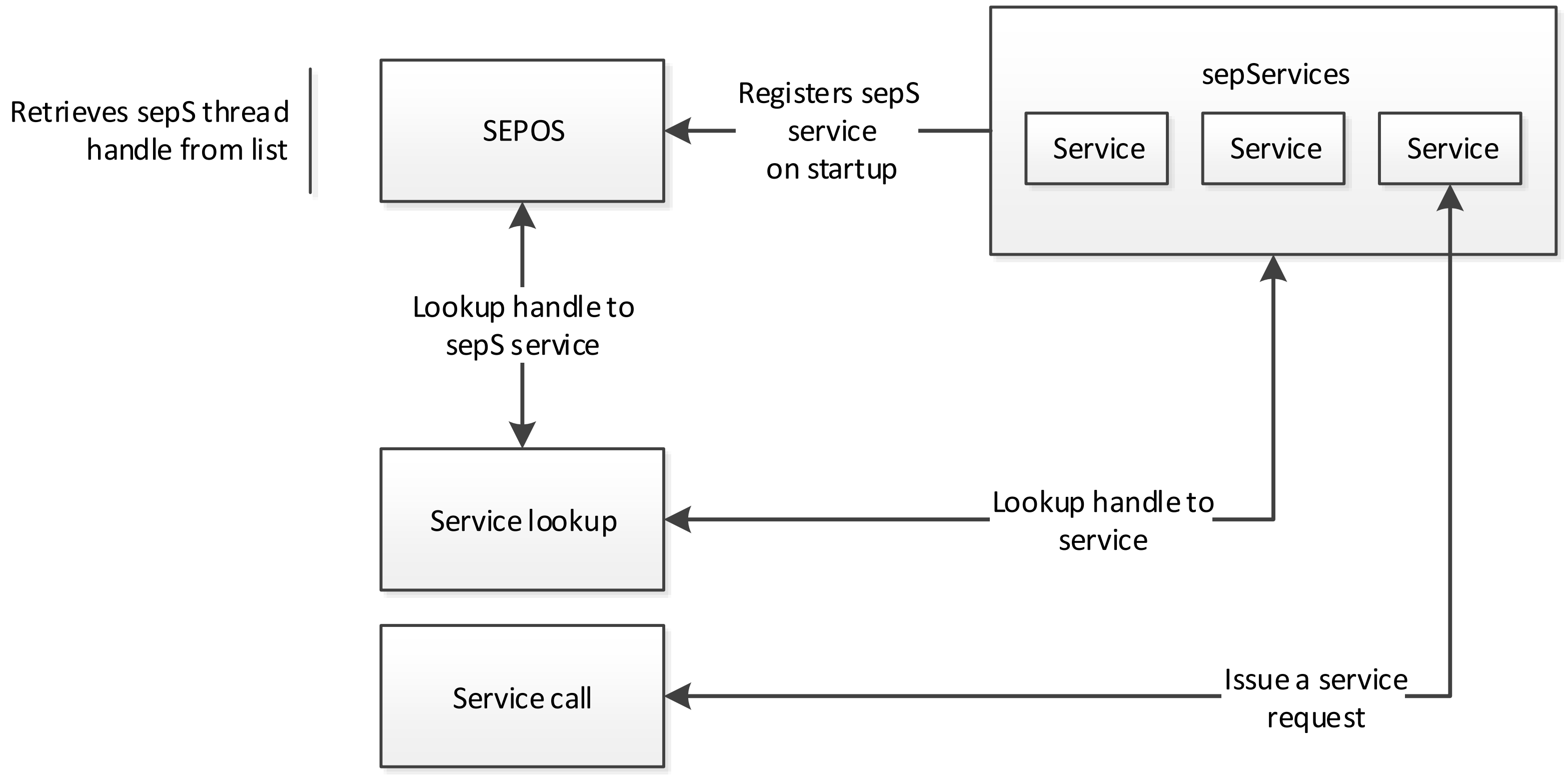
AKF Driver

- Manages AP/SEP endpoints in SEPOS
- Handles control (EP0) requests
  - E.g. sets up objects for reply and response OOL buffers
- SEP applications may register new endpoints to handle specific AP requests
  - AKF_ENDPOINT_REGISTER (0x412C) control request

SEP Services

- Hosts various SEP related services
  - Secure Key Generation Service
  - Test Service
  - Anti Replay Service
  - Entitlement Service
- Usually implemented on top of drivers
- Service API provided to SEP applications
  - service_lookup(...)
  - service_call(...)

Service Interaction



SEP Applications

- Primarily designed to support various drivers running in the AP
  - AppleSEPKeyStore → sks
  - AppleSEPCredentialManager → scrd
- Some apps are only found on certain devices
  - E.g. SSE is only present on iPhone 6 and later
- May also be exclusive to development builds
  - E.g. Debug application

Attacking SEP

Demystifying the Secure Enclave Processor

Attack Surface: SEPOS

- Mostly comprises the methods in which data is communicated between AP and SEP
  - Mailbox (endpoints)
  - Shared request/reply buffers
- Assumes that an attacker already has obtained AP kernel level privileges
  - Can execute arbitrary code under EL1

Attack Surface: AKF Endpoints

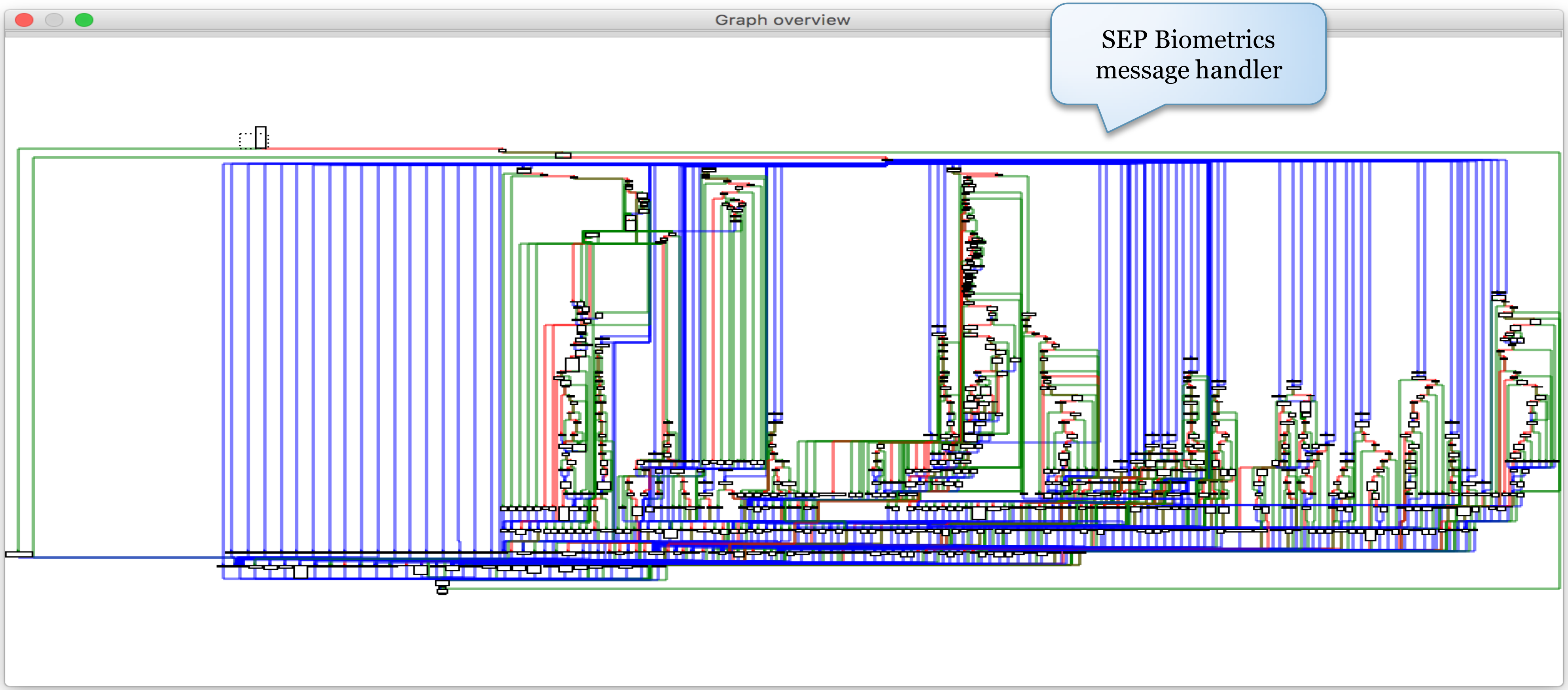
- Every endpoint registered with AKF is a potential target
  - Includes both SEP drivers and applications
- Does not require an endpoint to be registered with the SEP Manager (AP)
  - Can write messages to the mailbox directly
  - Alternatively, we can register our own endpoint with SEP Manager

Attack Surface: AKF Endpoints

Endpoint	Owner	OOL In	OOL Out	Notes
0	SEPD/ep0			
1	SEPD/ep1		✓	
2	ARTM	✓	✓	iPhone 6 and prior
3	ARTM	✓	✓	iPhone 6 and prior
7	sks	✓	✓	
8	sbio/sbio	✓	✓	
10	scrd/scrd	✓	✓	
12	sse/sse	✓	✓	iPhone 6 and later

List of AKF registered endpoints (iOS 9) and their use of out-of-line request and reply buffers

Attack Surface: Endpoint Handler



Attack Robustness

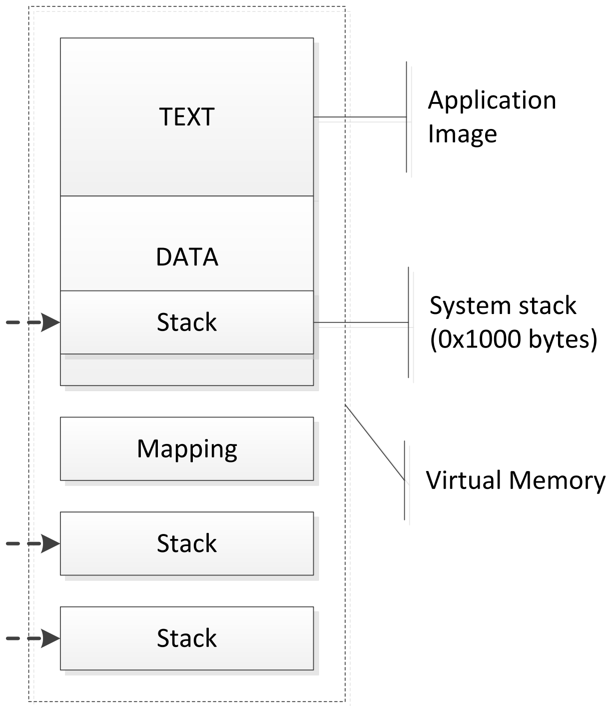
- How much effort is required to exploit a SEP vulnerability?
  - E.g. stack/heap corruption
- Determined by several factors
  - Address space layout
  - Allocator (heap) hardening
  - Exploit mitigations
  - And more

Address Space Layout

- SEP applications are loaded at their preferred base address
  - No image base randomization
  - Typically based at 0x1000 or 0x8000 (depending on presence of pagezero segment)
- Segments without a valid memory protection mask (!= 0) are ignored
  - E.g. __PAGEZERO is never "mapped"

Stack Corruptions

- The main thread of a SEP application uses an image embedded stack
  - A corruption could overwrite adjacent DATA segment data
- Thread stacks of additional threads spawned by SEPOS are mapped using objects
  - Allocated with gaps → "guard pages"



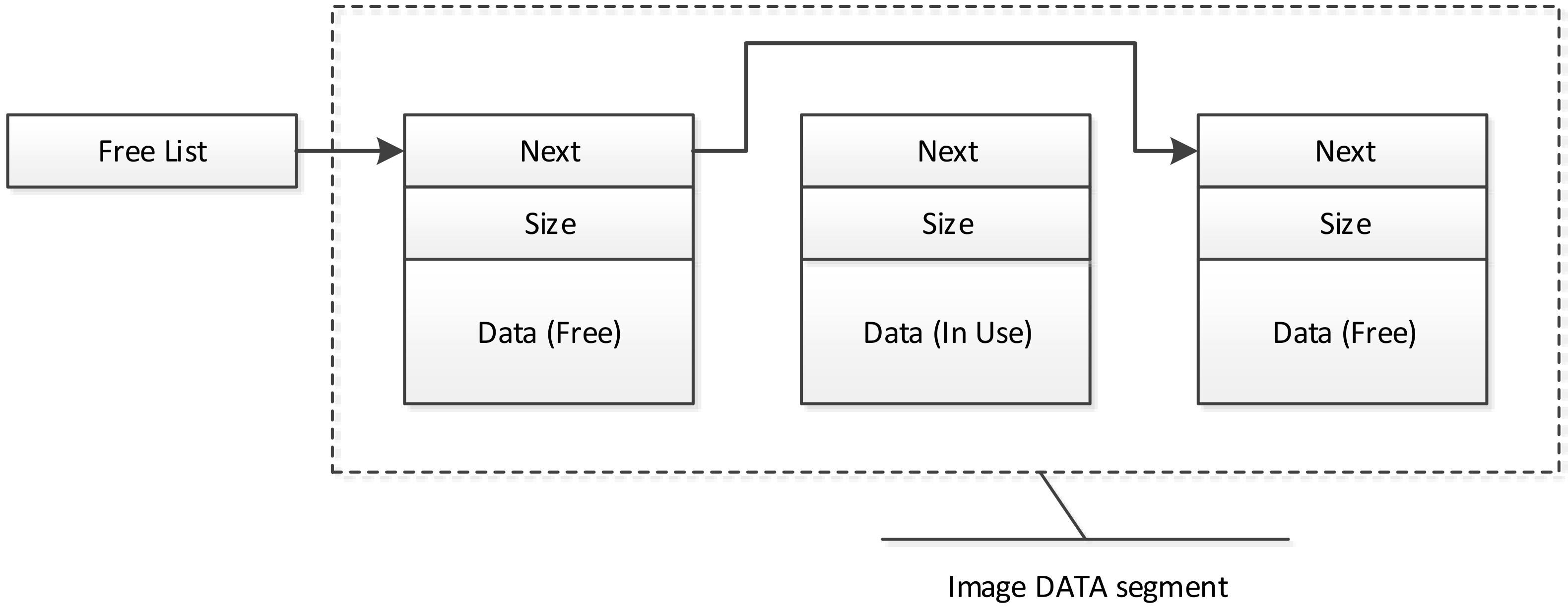
Stack Corruptions

- SEP applications are compiled with stack cookie protection
  - Cookie value is fixed to ‘GARD’
    - FIXED in iOS 10 (sort of)
  - Trivial to forge/bypass
- Stack addresses are in most cases known
  - Main thread stack is at a known address
  - Addresses of subsequent thread stacks are predictable

Heap Corruptions: malloc()

- Runtime allocator leveraged by SEP applications
  - K&R implementation
- Singly linked free list (ordered by size) with header that includes pointer and block size
  - struct Header { void * ptr, size_t size };
  - Coalesces adjacent elements on free()
- Size of heap determined on initialization
  - malloc_init(malloc_base, malloc_top);
  - Non-expandable

Heap Corruptions: malloc()



Heap Corruptions: malloc()

- No protection of heap metadata
  - Free list pointers can be overwritten
  - Block size can be corrupted
- Allocation addresses are predictable
  - Malloc area embedded by __DATA segment in application image
  - Allocations made in sequential order

No-Execute Protection

- SEPOS implements no-execute protection
- Always set when a page is not marked as executable
    - space_t::map_fpage()
    - Sets both XN and PXN bits in page table entries
- Non-secure (NS) bit also set for all pages outside SEP memory region

SEPOS Mitigations Summary

Mitigation	Present	Notes
Stack Cookie Protection	Yes	Fixed in iOS 10
Memory Layout Randomization		
User	No	
Kernel	No	Image base: 0xF0001000
Stack Guard Pages	Yes/No	Not for main thread
Heap Metadata Protection	No	
Null-Page Protection	No	Must be root task to map page
No-Execute Protection	Yes	Both XN and PXN

Attack Surface: BootROM

- Effectively only two major attack surfaces
  - IMG4 Parser
    - Memory Corruption
    - Logic Flaws
  - Hardware based
- Only minor anti-exploit mitigations present
  - No ASLR
  - Basic stack guard
  - One decent bug = game over

Attacking IMG4

- ASN.1 is a very tricky thing to pull off well
  - Multiple vulns in OpenSSL, NSS, ASN1C, etc
- LibDER itself actually rather solid
  - "Unlike most other DER packages, this one does no malloc or copies when it encodes or decodes"
    - – LibDER's readme.txt
  - KISS design philosophy
- But the wrapping code that calls it may not be
  - Audit seputil and friends
  - Code is signifigantly more complex then libDER itself

Attack Surface: Hardware

- Memory corruption attacks again data receivers on peripheral lines
  - SPI
  - I2C
  - UART
- Side Channel/Differential Power Analysis
  - Stick to the A7 (newer ones are more resistant)
- Glitching
  - Standard Clock/Voltage Methods
  - Others

External RAM

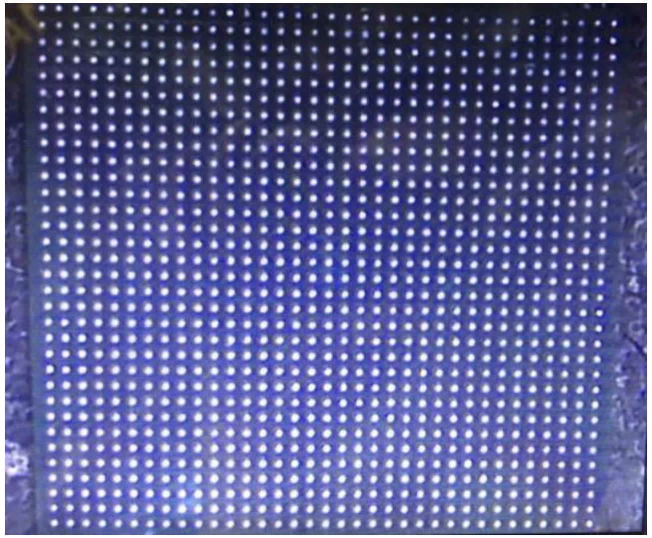
- Encrypted memory has no validation.
  - Can corrupt bits of SEP memory
- When generating the encryption key the “random component” is temporarily stored unencrypted in external RAM.
  - This may allow an attacker to influence generation of the final memory encryption key

Attacking the Fuse Array

- Potentially one of the most invasive attack vectors
  - Requires a lot of patience
  - High likelihood of bricking
- Laser could be used
  - Expensive method - not for us
- Primary targets
  - Production Mode
  - Security Mode

End Game: JTAG

- Glitch the fuse sensing routines
  - Requires a 2000+ pin socket
  - Need to bypass CRC and fuse sealing
  - “FSRC” Pin - A line into fuse array?



A8 SoC Pins

- Attack the IMG4 Parser
  - What exactly do DSEC and DPRO really do?

Conclusion

Demystifying the Secure Enclave Processor

Conclusion

- SEP(OS) was designed with security in mind
  - Mailbox interface
  - Privilege separation
- However, SEP(OS) lacks basic exploit protections
  - E.g. no memory layout randomization
- Some SEP applications expose a significant attack surface
  - E.g. SEP biometrics application

Conclusion (Continued)

- Overall hardware design is light years ahead of competitors
  - Hardware Filter
  - Inline Encrypted RAM
  - Generally small attack surface
- But it does have its weaknesses
  - Shared PMGR and PLL are open to attacks
  - Inclusion of the fuse source pin should be re-evaluated
  - The demotion functionality appears rather dangerous
    - Why does JTAG over lightning even exist?

Thanks!

- Ryan Mallon
- Daniel Borca
- Anonymous reviewers

Bonus Slides

Demystifying the Secure Enclave Processor

SEPOS: System Methods

Class	Id	Method	Description	Priv
0	0	sepos_proc_getpid()	Get the process pid	
0	1	sepos_proc_find_service()	Find a registered service by name	
0	1001	sepos_proc_limits()	Query process limit information	x
0	1002	sepos_proc_info()	Query process information	
0	1003	sepos_thread_info()	Query information for thread	
0	1004	sepos_thread_info_by_tid()	Query information for thread id	
0	1100	sepos_grant_capability()	-	x
0	2000	sepos_panic()	Panic the operating system	

SEPOS: Object Methods (1/2)

Class	Id	Method	Description	Priv
1	0	sepos_object_create()	Create an anonymous object	
1	1	sepos_object_create_phys()	Create an object from a physical region	x (*)
1	2	sepos_object_map()	Map an object in a task's address space	
1	3	sepos_object_unmap()	Unmap an object (not implemented)	
1	4	sepos_object_share()	Share an object with a task	
1	5	sepos_object_access()	Query the access control list of an object	
1	6	sepos_object_remap()	Remap the physical region of an object	x (*)
1	7	sepos_object_share2()	Share manifest with task	

SEPOS: Object Methods (2/2)

Class	Id	Method	Description	Priv
1	1001	sepos_object_object_info()	Query object information	x
1	1002	sepos_object_mapping_info()	Query mapping information	x
1	1003	sepos_object_proc_info()	Query process information	x
1	1004	sepos_object_acl_info()	Query access control list information	x

SEPOS: Thread Methods

Class	Id	Method	Description	Priv
2	0	sepos_thread_create()	Create a new thread	
2	1	sepos_thread_kill()	Kill a thread (not implemented)	
2	2	sepos_thread_set_name()	Set a service name for a thread	
2	3	sepos_thread_get_info()	Get thread information	