Make XNU ~~Great~~ little Again

Darwin evolution – hardware & software

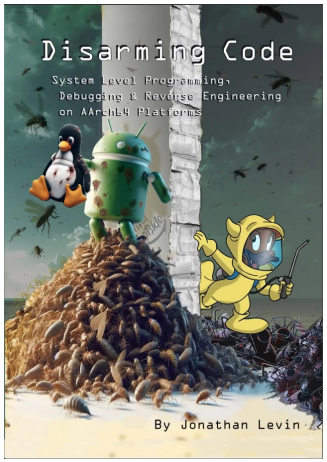
Objective-By-The-Sea v8

0xfa33415e!

/me

- Jonathan Levin
  - Founder, Technologeeks.com (providing expert internals/deep-dive trainings)
  - Co-Founder, DataFlow Forensics (turning internals expertise into APT detection)
  - Author, *OS Internals Trilogy, Android Internals (Vols I, II so far*) and DisARMing Code

* - Yes, I know, I owe the world Vols III & IV. Still on 'em



about:

- Apple is in a unique position controlling the full stack – hardware & software
  - Darwin, true to its name, is world's most advanced OS[1,2]
  - Apple Silicon has many custom/"bespoke" features
- "Premium" *OS Devices are also highly lucrative targets for APTs
- Last decade or so shows how Darwin is moving to hardware-based security
- XNU is slowly but surely shrinking, as security code bases are refactored

1 – If you want to argue about this, we can discuss this offline. If you want to pick a fight, tell me Android is better than *OS, I dare you

2 – For now.

Evolution of Jailbreaking – the 10,000 foot view

Year	iOS version	Mitigations*	Jailbreak technique
2008-2011	< 5	None	Get root, disable MACF sysctls
2011	5.0	MACF `sysctls` r/o in user mode	Get kernel r/w, patch
2012-2015	6.0-9.x	MACF `sysctls` ignored, KPP (WatchTower) debuts	patch in kernel (possibly racing KPP) CS bypass, amfidebilitate
2016	10.0	KTRR protects kernel text and __const	Patch kernel data, kernel code exec Or… just wait for Ian Beer
2018	12.0	A12: APRR/PPL protect (some) kernel data PAC largely kills kernel code exec	Patch kernel r/w data (zones) JB in User mode
2021	15.0	Read-Only (= write-once) kernel zones, CoreTrust	Physical (PTE) level attacks
2022	16.0	LockDown Mode, Developer Mode	CoreTrust, Triangulation (Dopamine)
2023	17.0	GXF (+SPTM/+TXM) on A15+	APTs in user mode Public JB eta S0n
2024	17.5**/18.0	Exclaves (on A16+).	
2025	19.0 (“26.0”)	”MIE” (read: FEAT_MTE4)	Public JB eta S0n0t happening

* - Mitigations not discussed – tighter sandboxing, AMFI improvements, Mach Msg/XPC filtering, mach_msg2, etc.

** - M4 iPad got exclaves in 17.5, an “E” release which often previews/tests the next year’s releases

Hardware Memory Protections

Apple Silicon Enhancements

KPP (WatchTower)

- Rudimentary form of Kernel Patch Protection for iPhone 5S-6S
  - Monitor code resides in EL3 (TrustZone)
  - Kernel requests `machine_lockdown` via SMC #2048
  - Monitor BLAKE-hashes kernel r/o pages
  - At interrupt time, monitor wakes up with budget to check pages
- Security gaps inherent and unfixable
  - Clearly an interim measure
  - Still the de-facto standard in Android[3]

3 – Still want to argue that Android is better than *OS? I dare you

KTRR

- A10 brings much stronger, hardware backed protection at MMU level
  - Apple Memory Cache Controller (AMCC) provides monitoring
  - Kernel requests `machine_lockdown` via special registers
    - q.v XNU-6153 sources which exposed the otherwise redacted sources*
  - Any attempt to `mprotect(2)` r/o pages immediately panics

Register	ARMv8 IMPLMENTATION DEFINED	Controls
KTRR_LOWER_EL1	S3_4_C15_C2_3	Lock from here
KTRR_UPPER_EL1	S3_4_C15_C2_4	Lock till here
KTRR_LOCK_EL1	S3_4_C15_C2_2	Toggle lock (write #1, once only)

* - And mysteriously appeared as a single tar file back then on what used to be https://opensource.apple.com/, allowing me to conclude MOXiI Vol.2. Thank you, AAPL!

KTRR challenges

- Much stronger
  - Patch/unpatch Window of KPP eliminated
  - Provides foundation of memory protection to this day, mostly intact*
- Although strong and well designed, not without shortcomings
  - Needs to be re-enabled on wake/resume of every CPU
  - Only allows one contiguous virtual memory region to be locked down
    - Inapplicable over kernel writable data - mutable stuff like processes, or entitlements/code signatures..

* - Aside from the ingenious KTRW by Brandon “the Breaker” Azad, then still with Project Zero, but later assimilated by AAPL

The Page Protection Layer (PPL)

- A12 (iPhone $X_{[R/S]}$) brings dynamic protection to kernel data – Page Protection Layer*
  - Page Protection Layer imposes masks on page-level access
  - Individual pages can now be toggled r/w only by a **trusted path**
    - Requires ARMv8.3 PAC (also introduced in A12) to actually be effective
    - PAC is a dramatic game changer (in kernel) effectively killing code exec
- XNU's already modular pmap rerouted to call PPL implementations

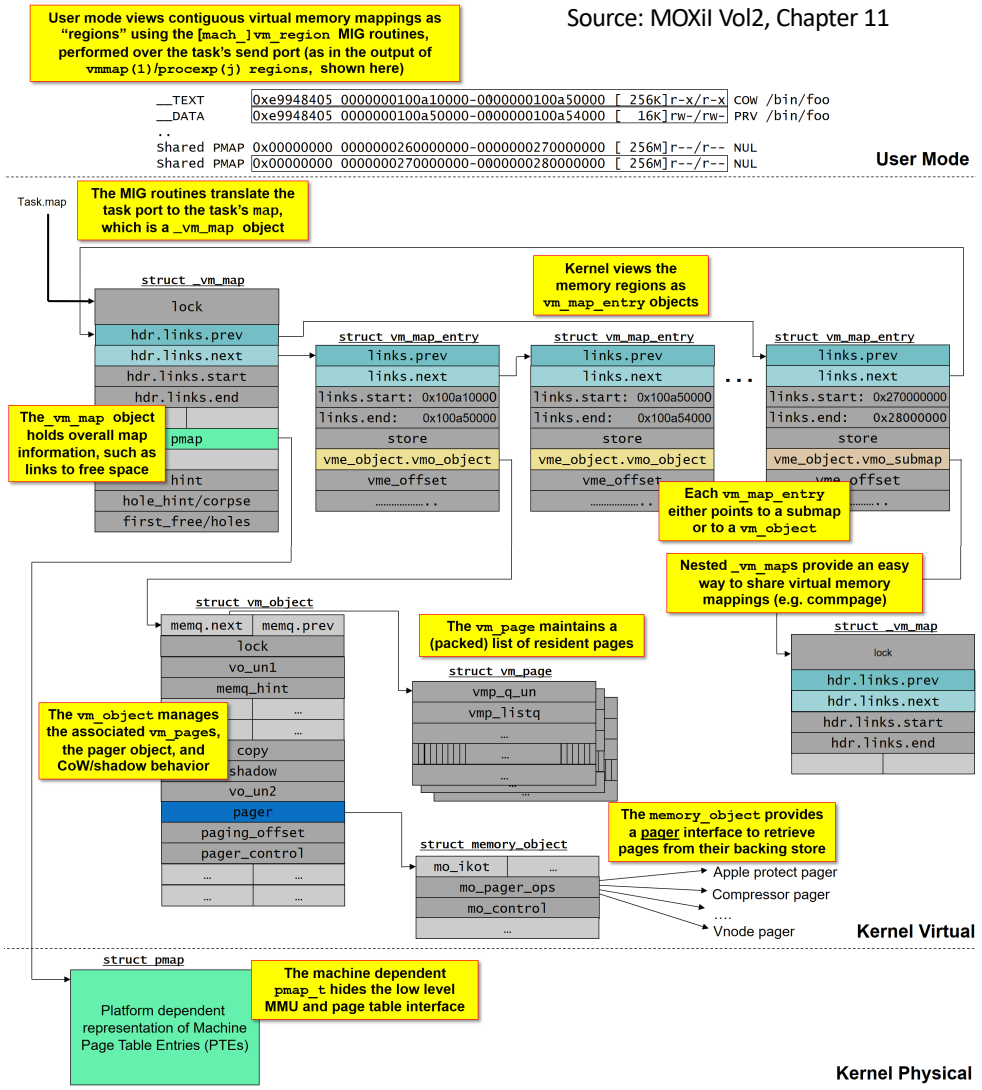
  http://newosxbook.com/articles/CasaDePPL.html

* - Or, from the way almost all *OS security hinges on it, the Panacea Protection Layer

* - Not so much in User Mode

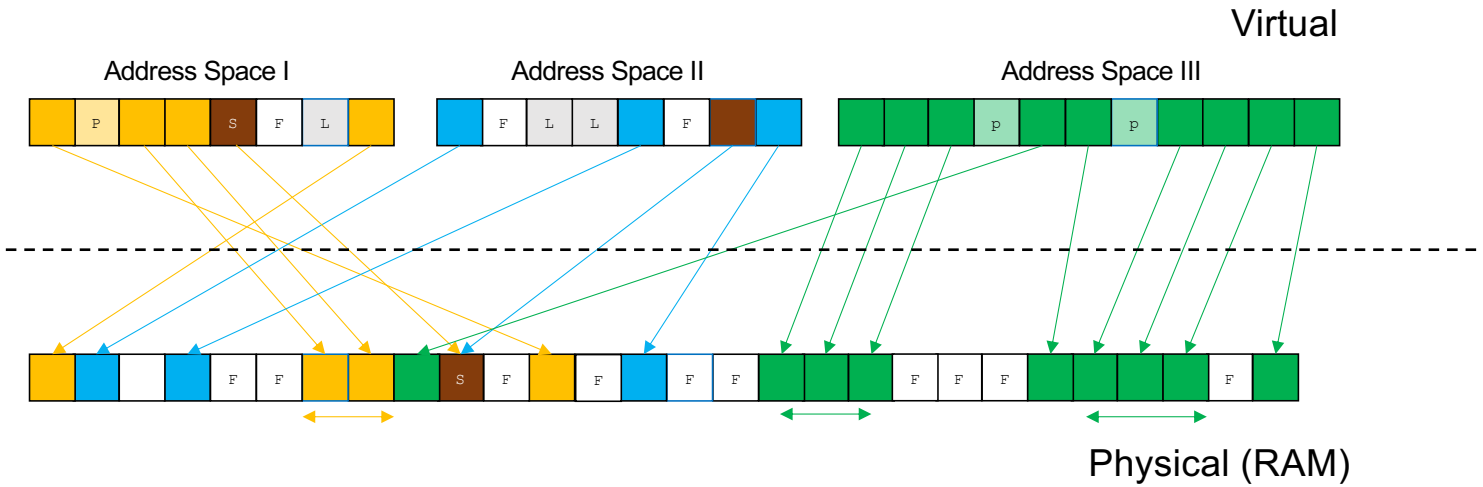
XNU vm_map/pmap

- XNU (Mach)'s vm_map handles VM
  - Linked list of vm_map_entry objects
    - Each is a mach_vm_region in user mode
    - Virtually contiguous, same permissions
- The pmap abstracts architecture paging
  - Entirely black boxed (for PPC/x86[_64]/ARM[v7/8])
  - Some help with "machine layer" (ml_*) APIs



* - No risk of RISC-V in the foreseeable future ☺

Reminder: Physical Memory and Paging



Legend:

- F — Free (unallocated)
- L — Lazy allocated (allocated, but not yet backed)
- S — Shared (between Address Space I and II)
- P P — Paged out (allocated, but not in RAM)
- ⟷ — Both Virtually & Physically Contiguous

Source: Disarming Code, Chapter 5

TL;DR: From moment MMU is on, any physical memory not in your page tables is inaccessible to you.

APRR

- Access Permission Restriction Registers(?) enable PPL on A12-A15
  - Two special registers:

Register	ARMv8 IMPLMENTATION DEFINED	Controls
ARM64_REG_APRR_EL1	S3_4_C15_C2_1	Kernel level access to pages (PPL)
ARM64_REG_APRR_EL0	S3_4_C15_C2_7	User mode access to pages (JIT)

  - Page permission bits in PTE (AP[1:2], PXN and XN) mask with register bits
  - Entry through a software trampoline (assumes no code exec in kernel)
- Excellent writeup by @s1guza (https://blog.siguza.net/APRR/)
- Inspires ARMv8.8/9 FEAT_S[12]P[IO]E (Permission Indirection/Overlays)

PPL as a substrate for hardened memory

- PPL can now be used to enable read-only zones
  - Zone element memory read still same
  - Zone element memory write requires ”trusted path” of PPL
- Solves KTRR limitations and protects mutable (but rarely modified) data
  - PROC/THREAD_RO, CRED, CS Blobs, Sandbox .. , AMFI entitlements
  - All but eliminates “Kernel RW Primitives” (reducing them to RO*)
  - Also guards “Developer Mode”, and 18.0+ “Restricted Execution Mode”
- Extended to user mode protections (JIT, later TPRO)

* - But note all other 600+ zones are still r/w

SPRR

- Shadow Permission Remap Registers(?) extend APRR functionality

Register	ARMv8 IMPLMENTATION DEFINED	Controls
SPRR_CONFIG_EL1	S3_6_C15_C1_0	Toggle SPRR
(Other register names/functions unknown, at least to me)		
aprr_shadow_jit_rw (EL1)	S3_6_C15_C1_5	User mode Page Permission Toggle
aprr_shadow_jit_rw (EL1)	S3_6_C15_C1_6	Possibly same, for EL1
aprr_shadow_jit_rw (EL2)	S3_6_C15_C1_7	Possibly same, for EL2

- First pointed out and well explained by Sven Peter of Asahi* (M1n1)
- `S3_6_C15_C1_5` used by `os_thread_self_restrict[tpro]*`
- Supported features communicated to userland via XNU's commpage

* - To AAPLytes, no doubt, a blasphemer (*"Linux?! UGH!"*) but to me, a giant whose shoulders I stand on today

AAPL-Si

MTE/MIE

- Latest* and greatest feature in Darwin 25
- https://security.apple.com/blog/memory-integrity-enforcement/

> Memory Integrity Enforcement (MIE) is the culmination of an unprecedented design and engineering effort, spanning half a decade, that combines the unique strengths of Apple silicon hardware with our advanced operating system security to provide industry-first, always-on memory safety protection across our devices — without compromising our best-in-class device performance. We believe Memory Integrity Enforcement represents the most significant upgrade to memory safety in the history of consumer operating systems.

- Implemented in kernel and in libsystem_malloc (xzm**)

* - Not that latest, Android 13-ish actually adopted it first. And it's only on the A19, and maybe whenever M5 comes out.

** - How about some sources already?

Enter: GXF

Guarded eXecution Feature

- GXF is a new AAPL-Si feature in A16+ devices
  - Their most advanced Silicon-Level protection yet
  - Not just more registers, but an entire compartmentalized execution state

Guarded eXecution Feature

Table 13/1-15: Apple Silicon's special GXF-related registers

Register encoding	Name	Purpose
s3_6_c15_c8_0	CurrentGL	Current Guarded Level (0/1/2)
s3_6_c15_c8_1	GENTER_ADDR	Address to jump to on GENTER
s3_6_c15_c8_5	PMCR1_GL2	Performance Monitor Control Register
s3_6_c15_c8_7	PMCR1_GL1	
s3_6_c15_c10_0	SP_GL1	Stack Pointer
s3_6_c15_c10_1	TPIDR_GL1	Thread Processor Identification Register
s3_6_c15_c9_2	VBAR_GL12	(Exception) Vector Base Address Register
s3_6_c15_c10_2	VBAR_GL1	
s3_6_c15_c9_3	SPSR_GL12	Saved Program Status Register
s3_6_c15_c10_3	SPSR_GL1	
s3_6_c15_c9_4	ASPSR_GL12	Saved Program Status Register, indicating GL/EL
s3_6_c15_c10_4	ASPSR_GL1	
s3_6_c15_c9_5	ESR_GL12	Exception Syndrome Register
s3_6_c15_c10_5	ESR_GL1	
s3_6_c15_c9_6	ELR_GL12	Exception Link Register
s3_6_c15_c10_6	ELR_GL1	
s3_6_c15_c9_7	FAR_GL12	Fault Address Register
s3_6_c15_c10_7	FAR_GL1	

(and probably many more registers than the narrow margins of this slide could fit)

Guarded eXecution Feature

- ”GL” is a TrustZone-inspired compartment Exception Level
  - Implements GL0, GL1, GL2 – akin to “Secure World”
  - Registers suffixed _GL*x* can only be accessed at that GL or higher
- Enables TXM (GL0), SPTM (GL2) and Exclaves (GL0)
- Mode traversal through a custom AAPL-Si instruction - GENTER
- GENTER: `0x0020142x` (where x=*#imm5*, i.e. 5-bit immediate)
- GEXIT: `0x00201400` (imm = 0)

GENTER?

what is Apple Silicon's GENTER?

It seems like there might be a typo or confusion with "GENTER" in your question. If you were referring to **Apple Silicon's "GPU"** (Graphics Processing Unit) or **"neural engine" (or NPU)**, I can explain that in more detail!

Apple Silicon refers to Apple's custom-designed ARM-based chips, like the M1, M1 Pro, M1 Max, M2, and so on. These chips integrate several components:

- **CPU** (Central Processing Unit): The general processor for running apps.
- **GPU** (Graphics Processing Unit): Handles graphics rendering for everything from casual apps to gaming.
- **Neural Engine (NPU)**: A specialized processor designed specifically for machine learning tasks, like image recognition or natural language processing.

If "GENTER" refers to something specific you had in mind, could you clarify? Or if you're asking about any other feature of Apple Silicon, I'd be happy to help!

Put this in more context?

You have no idea what you're talking about. GENTER is a custom instruction in Apple Silicon

Ah, now I understand! You're referring to **GENTER** as a **custom instruction** in Apple Silicon, specifically related to Apple's custom ARM-based architecture.

GENTER (General-Purpose *Entropic* Random Number Generator) is indeed a specialized instruction that is part of Apple Silicon's hardware. It's a custom instruction found in Apple's chips, like the M1, M1 Pro, M1 Max, and newer models. This instruction provides hardware-accelerated **random number generation** that is used for various cryptographic and security operations.

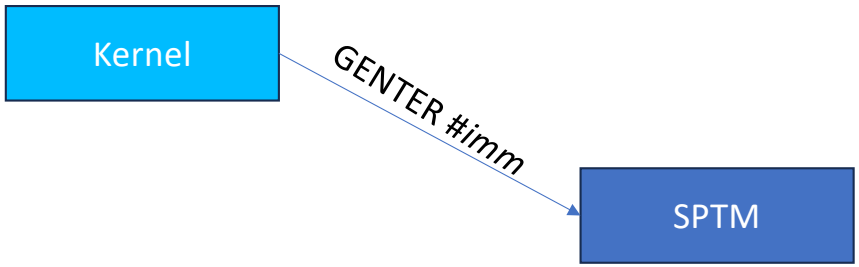
Key points about GENTER:

1. **Purpose**: It provides a **high-performance, hardware-backed random number generator** for cryptography, security, and other applications requiring randomness.
2. **Efficiency**: Unlike software-based RNGs (which can be slower and more power-hungry), GENTER is optimized directly in the hardware, ensuring that operations are quick and efficient.
3. **Security**: Hardware-based RNGs are often more secure because they can be less predictable and harder to manipulate compared to software RNGs, which might be vulnerable to attacks.

This instruction likely plays a crucial role in security protocols, encryption, and applications that require entropy for randomness, such as key generation, hashing, or tokenization processes.

Let me know if you'd like more details or have any further questions!

SPTM



Secure Page Table Monitor

- As name implies, manages/monitors page tables
- IM4P containing Mach-O executable, loaded by iBoot
  - Easily disassembles with `disarm(j), ipsw, etc.`
- Runs in GL2, traps:
  - GENTER from non GL (through ESR_GL1),
  - GL1 SVC/exceptions from TXM (through VBAR)
  - GL2 HVC/exceptions from SK (through VBAR)
- Provides pmap services for all the above + IOMMU/DARTs,and more
- Recent Paper by Stefin et Classen: https://arxiv.org/abs/2510.09272
- Surprisingly well documented by Kernel.framework headers!*

* - Likely due to an oversight of AAPL's, since these files are reducted from XNU-11xxx's own sources

SPTM Entry (via GENTER)

- XNU Enters SPTM through GENTER with immediate (in `ESR_GL1`) to `S3_6_C15_C8_1`
- Immediate gets stored in ESR_GL1, other arguments, X16 intact

```
morpheus@eM1nent (~/Downloads/Firmware/19/Firmware) %disarm  -r _do_genter sptm.t8150.release.im4p
_sets_up_genter
fffffff0270a38dc        d503245f        BTI     c               ;
fffffff0270a38e0        d50041bf        MSR     SPSel, #1       ;
fffffff0270a38e4        9100001f        MOVr    X31, X0         ; SP = X0 (0x0 - (null).. Rd Code 0)
fffffff0270a38e8        d50040bf        MSR     SPSel, #0       ;
fffffff0270a38ec        b0000062        ADRP    X2, #13         ; X2 = 0xfffffff0270b0000-
fffffff0270a38f0        91176042        ADD     X2, X2, #1496   ; X2 = 0xfffffff0270b05d8 -- !'_halts'
fffffff0270a38f4        d51ef842        MSR     AAPL_GENTER_ERROR_MAYBE, X2   ; S2_6_C15_C8_2...
fffffff0270a38f8        b0fffa2         ADRP    X2, #-11        ; X2 = 0xfffffff027098000-
fffffff0270a38fc        91149042        ADD     X2, X2, #1316   ; X2 = 0xfffffff027098524 -- !'_do_genter'
fffffff0270a3900        d51ef822        MSR     AAPL_GENTER_ADDR, X2  ; AAPL_GENTER_ADDR = 0xfffffff027098524
fffffff0270a3904        b0ffffc2        ADRP    X2, #-7         ; X2 = 0xfffffff02709c000-
fffffff0270a3908        91000042        ADD     X2, X2, #0      ; X2 = 0xfffffff02709c000 -- !'_vbar_for_GL1'
fffffff0270a390c        d51efa42        MSR     VBAR_GL1, X2    ; VBAR_GL1 =  0xfffffff02709c000
fffffff0270a3910        d5033fdf        ISB                     ;
..
fffffff0270a394c        d65f03c0        RET     ;
```

sptm.t8150.release.im4p from iPhone18,4_26.0_23A341

```
morpheus@eM1nent (~/Downloads/Firmware/19/Firmware) %disarm  -r _do_genter sptm.t8150.release.im4p
_do_genter:
fffffff027098524         d500409f        MSR     PAN, #0 ;
fffffff027098528         d503409f        MSR     TCO, #0 ;
fffffff02709852c         d53efaa8        MRS     X8, ESR_GL1     ; X8 = ESR_GL1
fffffff027098530         92401108        AND     X8, X8, #0x1f   ; X8 = 0x0 & 0x1f = 0x0
fffffff027098534         f100111f        CMPi    X8, #4  ;
fffffff027098538         54001da0        B.EQ    0xfffffff0270988ec      ; on GENTER 4 from XNU WTF!
fffffff02709853c         900002e9        ADRP    X9, #92 ; X9 = 0xfffffff0270f4000-
fffffff027098540         79404929        LDRH_i  W9, [X9, #18]   ; X9 = *(0xfffffff0270f4024) (9) = ???
fffffff027098544         36780189        TBZ     W9, #15 0xfffffff027098574      ; ??
fffffff027098548         f100011f        CMPi    X8, #0  ;
fffffff02709854c         540026a0        B.EQ    0xfffffff027098a20      ; on GENTER 0 from XNU
fffffff027098550         f100051f        CMPi    X8, #1  ;
fffffff027098554         540011c0        B.EQ    0xfffffff02709878c      ; on GENTER 1 resume from exception
fffffff027098558         f100091f        CMPi    X8, #2  ;
fffffff02709855c         540004e0        B.EQ    0xfffffff0270985f8      ; on GENTER 2 txm resume
fffffff027098560         f1000d1f        CMPi    X8, #3  ;
fffffff027098564         54000b40        B.EQ    0xfffffff0270986cc      ; on GENTER 3 sk resume
fffffff027098568         d29bd5a0        MOVZ    X0, #57005      ; X0 = 0xdead
hang_here:
fffffff02709856c         d503205f        WFE     ;
        func_0xfffffff02709856c(0xdead,ARG1,ARG2,ARG3,ARG4);
fffffff027098570         17ffffff        B       0xfffffff02709856c      ; hang here
```

```
/* SPTM dispatch logic GENTER immediate values */
#define SPTM_GENTER_DISPATCH_CALL          0
#define SPTM_GENTER_RESUME_FROM_EXCEPTION 1
#define SPTM_GENTER_TXM_RESUME             2
#define SPTM_GENTER_SK_RESUME              3
#define SPTM_GENTER_XNU_PANIC_BEGIN        4
```

sptm.t8150.release.im4p from iPhone18,4_26.0_23A341

Dispatch Tables

- SPTM registers dispatch tables for its domains
  - Domain then enters SPTM by packing domain and selector into X16

Dispatch Table	Type	Uses
0	Domain(1)	XNU
1	Domain(2)	TXM
2	Domain(3)	SK
3	IOMMU	t8110DART-XNU
4	IOMMU	t8110DART-SK
5	IOMMU	SART
6	IOMMU	NVMe
7	IOMMU	Universal(?) Address Translation (UAT)
8	IOMMU	Sh(ared?) Address Translation (SHART)
9	IOMMU	CPUTRACE (RESERVED in header)
10	Domain(4)	XNU Hibernation

Darwin 25

Dispatch Table	Type	Uses
11	IOMMU	Gen 3 t8110DART-XNU
12	IOMMU	Gen 3 t8110DART-SK

Dispatch Tables

- SPTM registers dispatch tables for its domains
  - Domain then enters SPTM by packing domain and selector into X16

```
/**
 * Used to represent the target of a function call via the dispatch logic. A
 * Dispatch target effectively identifies a specific called domain, dispatch
 * table ID, and endpoint ID.
 *
 * ```
 *  63      56 55           48 47      40 39                       32 31                           0
 * +----------+---------------+----------+---------------------------+-------------------------------+
 * | Reserved | sptm_domain_t | Reserved | sptm_dispatch_table_id_t | sptm_dispatch_endpoint_id_t |
 * +----------+---------------+----------+---------------------------+-------------------------------+
 * ```
 */
```

XNU/SPTM interface

```
morpheus@eM1nent (~/…/Firmware/19) %disarm -e filesets kernelcache.release.d23
kernelcache.release.d23.iOS19.iPhone18,4: This is an IM4P... with a BVX2 payload
295 LC_FILESET_ENTRY commands
All files are in /tmp/extracted. Kernel is in /tmp/extracted/kernel.rebuilt
Getting kernel from 0x8000
Kernel is in /tmp/extracted/kernel.rebuilt

morpheus@eM1nent (~/…/Firmware/19) %disarm -g movk,movk,movk,movk,b /tmp/extracted/kernel.rebuilt
fffffe0008adc2f4(0x8842f4): MOVK X16, #0, LSL #48  ; X16 := 0x0
fffffe0008adc2f8(0x8842f8): MOVK X16, #0, LSL #32  ; X16 := 0x0
fffffe0008adc2fc(0x8842fc): MOVK X16, #0, LSL #16  ; X16 := 0x0
fffffe0008adc300(0x884300): MOVK X16, #0 ; X16 := 0x0
fffffe0008adc304(0x884304): B 0xfffffe000b029190 ; _func_0xfffffe000b029190
---
...
fffffe0008adc4a4(0x8844a4): MOVK X16, #0, LSL #48  ; X16 := 0x0
fffffe0008adc4a8(0x8844a8): MOVK X16, #0, LSL #32  ; X16 := 0x0
fffffe0008adc4ac(0x8844ac): MOVK X16, #0, LSL #16  ; X16 := 0x0
fffffe0008adc4b0(0x8844b0): MOVK X16, #42 ; X16 := 0x2a
fffffe0008adc4b4(0x8844b4): B 0xfffffe000b029190 ; _func_0xfffffe000b029190
...
```

Note packing of domain (0) with SPTM selector

XNU/SPTM interface

```
morpheus@eM1nent (~/…/Firmware/19) %disarm -r  _func_0xfffffe000b029190   \
                                              /tmp/extracted/kernel.rebuilt
fffffe000b029190 d503237f PACIBSP  ;
fffffe000b029194 a9bf7bfd STP X29, X30, [X31, #-16]!   ; SP -= 16; *[SP] = [X29, X30]
fffffe000b029198 910003fd MOVr X29, X31                ; FP = SP
     _func_0xfffffe000825c438();                       ; pre_genter_callout
fffffe000b02919c 9748cca7 BL 0xfffffe000825c438        ;
fffffe000b0291a0 910003bf MOVr X31, X29                ; SP = FP
fffffe000b0291a4 a8c17bfd LDP X29, X30, [X31], #16     ; [X29, X30] = *[X31 + 16]; X31 += 2
fffffe000b0291a8 00201420 GENTER #0                    ;
fffffe000b0291ac a9bf7bfd STP X29, X30, [X31, #-16]!   ; SP -= 16; *[SP] = [X29, X30]
fffffe000b0291b0 910003fd MOVr X29, X31                ; FP = SP
    _func_0xfffffe000825c4a4(0);                       ; post_genter_callout
fffffe000b0291b4 9748ccbc BL 0xfffffe000825c4a4        ;
fffffe000b0291b8 910003bf MOVr X31, X29                ; SP = FP
fffffe000b0291bc a8c17bfd LDP X29, X30, [X31], #16     ; [X29, X30] = *[X31 + 16]; X31 += 2
```

XNU/SPTM interface

#	Purpose
0	Lockdown
1	Retype
2	Map page
3	Map Table
4	Unmap Table
…	
32	Hibernation finalize
33	IOFilter protected write
Darwin 25	
37	SPTM sysctl
..	…
42	SURT Free

```
morpheus@eM1nent (~/…/Firmware) %\disarm -r xnu_endpoints
sptm.t8150.release.im4p | head
xnu_endpoints:
fffffff02701cdd8:    0xfffffff0270b0b74   _sptm_lockdown_xnu
fffffff02701cde0: 00 00 00 00 00 00 00 00    |........|
fffffff02701cde8:    0xfffffff0270e2624   _sptm_retype_page
fffffff02701cdf0: 00 00 00 00 00 00 00 00    |........|
fffffff02701cdf8:    0xfffffff0270e2f88   _sptm_map_page
fffffff02701ce00: 00 00 00 00 00 00 00 00    |........|
fffffff02701ce08:    0xfffffff0270e4708   _sptm_map_table
fffffff02701ce10: 00 00 00 00 00 00 00 00    |........|
fffffff02701ce18:    0xfffffff0270e5188   _sptm_unmap_table
...
```

All ridiculously well documented in Darwin 24-25's

`Kernel.framework/Headers/platform/sptm/xnu_sptm.h`

Page Retyping

- A main function of SPTM is typing and retyping pages
- Some 64 such page types defined
- When servicing a pmap page request, type will be verified
- Type transitions are keyed in some state machine
  - Only specific transitions are considered valid.
  - Complete analysis of state transitions in Stefin & Klassen's paper

SPTM

```
morpheus@eM1nent (~/…/Firmware) %disarm  -r state_machine_states sptm.t8150.release.im4p
sptm.t8150.release.im4p: This is an IM4P... with a BVX2 payload... Uncompressed 1130528 bytes
Opened companion file sptm.t8150.release.im4p.ARM64.CAB28017-4800-31B0-8BD1-1524CBE33025
state_machine_states:
fffffff02701d0f8:    0xfffffff027012814   "STATE_SPTM_BOOTSTRAP"
fffffff02701d100:    0xfffffff027012829   "STATE_SK_BOOTSTRAP"
fffffff02701d108:    0xfffffff02701283c   "STATE_TXM_BOOTSTRAP"
fffffff02701d110:    0xfffffff027012850   "STATE_SPTM_CALLED_BY_TXM_BOOTSTRAP"
fffffff02701d118:    0xfffffff027012873   "STATE_SPTM_CALLED_BY_SK_BOOTSTRAP"
fffffff02701d120:    0xfffffff027012895   "STATE_XNU"
fffffff02701d128:    0xfffffff02701289f   "STATE_XNU_GUEST"
fffffff02701d130:    0xfffffff0270128af   "STATE_XNU_HIB"
fffffff02701d138:    0xfffffff0270128bd   "STATE_TXM_CALLED_BY_XNU"
fffffff02701d140:    0xfffffff0270128d5   "STATE_TXM_SAVE_CONTEXT"
fffffff02701d148:    0xfffffff0270128ec   "STATE_TXM_CALLED_BY_SPTM"
fffffff02701d150:    0xfffffff027012905   "STATE_SPTM_CALLED_BY_XNU"
fffffff02701d158:    0xfffffff02701291e   "STATE_SPTM_CALLED_BY_XNU_HIB"
fffffff02701d160:    0xfffffff02701293b   "STATE_XNU_HANDLE_SPTM_EXCEPTION"
fffffff02701d168:    0xfffffff02701295b   "STATE_SPTM_CALLED_BY_TXM"
fffffff02701d170:    0xfffffff027012974   "STATE_SPTM_CALLED_BY_SK"
fffffff02701d178:    0xfffffff02701298c   "STATE_SK_CALLED_BY_XNU"
fffffff02701d180:    0xfffffff0270129a3   "STATE_SK_CALLED_BY_SPTM_HIB"
fffffff02701d188:    0xfffffff0270129bf   "STATE_SK_EXCEPTION"
fffffff02701d190:    0xfffffff0270129d2   "STATE_XNU_PANIC"
fffffff02701d198:    0xfffffff0270129e2   "STATE_PANIC_SPTM_CALLED_BY_XNU"
fffffff02701d1a0:    0xfffffff027012a01   "STATE_DISPATCH_PANIC_SPIN"
fffffff02701d1a8:    0xfffffff027012a1b   "STATE_SPTM_REENTERED_BY_XNU"
```

type_id_to_string

```
morpheus@eM1nent (~/Downloads/Firmware/19/Firmware) %disarm -r _type_id_to_string sptm.t8150.release.im4p
sptm.t8150.release.im4p: This is an IM4P... with a BVX2 payload... Uncompressed 1130528 bytes
Opened companion file sptm.t8150.release.im4p.ARM64.CAB28017-4800-31B0-8BD1-1524CBE33025
fffffff0270ccbd8 d503245f BTI c ;
fffffff0270ccbdc 7101001f CMPi W0, #64                ;
fffffff0270ccbe0 540000a2 B.CS 0xfffffff0270ccbf4     ; type_id_to_string_panic
fffffff0270ccbe4 b0fffa68 ADRP X8, #-179              ; X8 = 0xfffffff027019000-
fffffff0270ccbe8 9115a108 ADD X8, X8, #1384          ; X8 = X8 + 0x568 = 0xfffffff027019568 -- !
fffffff0270ccbec f8605900 LDRr X0, [X8, W0, UXTW #2] ; (Arg0-relative - unavailable in static disassembly)
fffffff0270ccbf0 d65f03c0 RET ;
type_id_to_string_panic:
fffffff0270ccbf4        d503237f        PACIBSP ;
fffffff0270ccbf8        d100c3ff        SUBi    SP, SP, #48       ; SP = SP - 0x30
fffffff0270ccbfc        a9027bfd        STP     X29, X30, [X31, #32]    ; *[SP +32] = [X29, X30]
fffffff0270ccc00        910083fd        MOVr    X29, X31, #32     ; FP = SP + 32
fffffff0270ccc04        528007e8        MOVZ    W8, #63           ; X8 = 0x3f
fffffff0270ccc08        d0fffa09        ADRP    X9, #-190         ; X9 = 0xfffffff02700e000-
fffffff0270ccc0c        91055529        ADD     X9, X9, #341      ; X9 = 'type_id_to_string'
fffffff0270ccc10        a900a3e0        STP     X0, X8, [X31, #8] ; *[SP +8] = [X0, X8]
fffffff0270ccc14        f90003e9        STRi    X9, [X31]         ; *0x0 = 0xfffffff02700e155
fffffff0270ccc18        d0fffa00        ADRP    X0, #-190         ; X0 = 0xfffffff02700e000-
fffffff0270ccc1c        91047400        ADD     X0, X0, #285      ; X0 = '%s: %d is not a valid type ID …'
        _panics_sptm("%s: %d is not a valid type ID (valid type IDs are 0-%d)");
```

```
morpheus@eM1nent (~/Downloads/Firmware/19/Firmware) %disarm -a 0xfffffff027019568 sptm*p 2>/dev/null| head -64
fffffff027019568:    0xfffffff02700dcff   "SPTM_UNTYPED"
fffffff027019570:    0xfffffff02700dd0c   "SPTM_UNUSED”

fffffff027019578:    0xfffffff02700dd18   "SPTM_DEFAULT"

fffffff027019580:    0xfffffff02700dd25   "SPTM_RO"

fffffff027019588:    0xfffffff02700dd2d   "SPTM_CODE"

fffffff027019590:    0xfffffff02700dd37   "SPTM_TXM_CODE"

fffffff027019598:    0xfffffff02700dd45   "SPTM_XNU_CODE"
fffffff0270195a0:    0xfffffff02700dd53   "SPTM_XNU_CODE_DBG_RW"

fffffff0270195a8:    0xfffffff02700dd68   "SPTM_KERNEL_ROOT_TABLE"

fffffff0270195b0:    0xfffffff02700dd7f   "SPTM_PAGE_TABLE"

fffffff0270195b8:    0xfffffff02700dd8f   "SPTM_IOMMU_BOOTSTRAP"
```

```
/* SPTM-owned type IDs */
#define SPTM_UNTYPED                0U
#define SPTM_UNUSED                 1U
#define SPTM_DEFAULT                2U
#define SPTM_RO                     3U
#define SPTM_CODE                   4U
#define SPTM_TXM_CODE               5U
#define SPTM_XNU_CODE               6U
#define SPTM_XNU_CODE_DBG_RW        7U
#define SPTM_KERNEL_ROOT_TABLE      8U
#define SPTM_PAGE_TABLE             9U
#define SPTM_IOMMU_BOOTSTRAP        10U
```

SPTM

```
morpheus@eM1nent (~/Downloads/Firmware/19/Firmware) %disarm -a 0xfffffff027019568 sptm*p 2>/dev/null| head -64
…
fffffff0270195c0:    0xfffffff02700dda4   "XNU_DEFAULT"
fffffff0270195c8:    0xfffffff02700ddb0   "XNU_RO"                KTRR protected
fffffff0270195d0:    0xfffffff02700ddb7   "XNU_RO_DBG_RW"         Not in your build
fffffff0270195d8:    0xfffffff02700ddc5   "XNU_USER_EXEC"
fffffff0270195e0:    0xfffffff02700ddd3   "XNU_USER_DEBUG"          User mode pages
fffffff0270195e8:    0xfffffff02700dde2   "XNU_USER_JIT"
fffffff0270195f0:    0xfffffff02700ddef   "XNU_USER_ROOT_TABLE"    User mode root PTE (address space)
fffffff0270195f8:    0xfffffff02700de03   "XNU_SHARED_ROOT_TABLE"      DSC, Nested pmaps
fffffff027019600:    0xfffffff02700de19   "XNU_PAGE_TABLE"
fffffff027019608:    0xfffffff02700de28   "XNU_PAGE_TABLE_SHARED"      Kernel Page tables
fffffff027019610:    0xfffffff02700de3e   "XNU_PAGE_TABLE_ROZONE"
fffffff027019618:    0xfffffff02700de54   "XNU_PAGE_TABLE_COMMPAGE"
fffffff027019620:    0xfffffff02700de6c   "XNU_IOMMU"    Managed by IOMMU in SPTM
fffffff027019628:    0xfffffff02700de76   "XNU_ROZONE"     R/O Kernel Zones
fffffff027019630:    0xfffffff02700de81   "XNU_IO"
fffffff027019638:    0xfffffff02700de88   "XNU_PROTECTED_IO"
fffffff027019640:    0xfffffff02700de99   "XNU_COMMPAGE_RW"
fffffff027019648:    0xfffffff02700dea9   "XNU_COMMPAGE_RO"     XNU's Commpage
fffffff027019650:    0xfffffff02700deb9   "XNU_COMMPAGE_RX"
fffffff027019658:    0xfffffff02700dec9   "XNU_TAG_STORAGE"     MTE
fffffff027019660:    0xfffffff02700ded9   "XNU_STAGE2_ROOT_TABLE"  Stage 2 Page tables
fffffff027019668:    0xfffffff02700deef   "XNU_STAGE2_PAGE_TABLE"
fffffff027019670:    0xfffffff02700df05   "XNU_KERNEL_RESTRICTED"   User copyio forbidden
fffffff027019678:    0xfffffff02700df1b   "XNU_CPUTRACE_PA_BUFFER"
fffffff027019680:    0xfffffff02700df32   "XNU_CPUTRACE_VA_BUFFER"
fffffff027019688:    0xfffffff02700df49   "XNU_RESTRICTED_IO"
fffffff027019690:    0xfffffff02700df5b   "XNU_RESTRICTED_IO_TELEMETRY"
fffffff027019698:    0xfffffff02700df77   "XNU_SUBPAGE_USER_ROOT_TABLES"
```

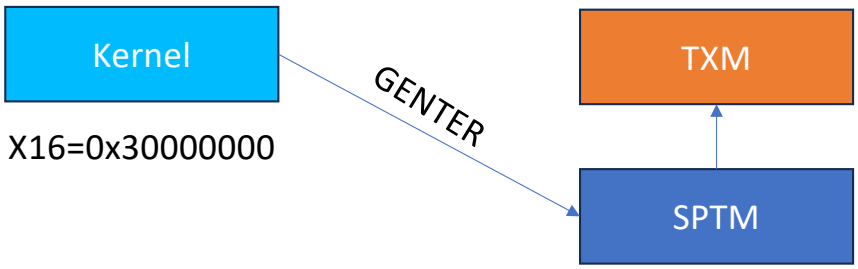
```
morpheus@eM1nent (~/Downloads/Firmware/19/Firmware) %disarm -a 0xfffffff027019568 sptm*p 2>/dev/null| head -64
…
fffffff0270196a0:    0xfffffff02700df94   "TXM_DEFAULT"
fffffff0270196a8:    0xfffffff02700dfa0   "TXM_RO"
fffffff0270196b0:    0xfffffff02700dfa7   "TXM_RW"
fffffff0270196b8:    0xfffffff02700dfae   "TXM_CPU_STACK"
fffffff0270196c0:    0xfffffff02700dfbc   "TXM_THREAD_STACK"
fffffff0270196c8:    0xfffffff02700dfcd   "TXM_ADDRESS_SPACE_TABLE"
fffffff0270196d0:    0xfffffff02700dfe5   "TXM_MALLOC_PAGE"
fffffff0270196d8:    0xfffffff02700dff5   "TXM_FREE_LIST"
fffffff0270196e0:    0xfffffff02700e003   "TXM_SLAB_TRUST_CACHE"
fffffff0270196e8:    0xfffffff02700e018   "TXM_SLAB_PROFILE"          TXM Slab (object) allocator
fffffff0270196f0:    0xfffffff02700e029   "TXM_SLAB_CODE_SIGNATURE"
fffffff0270196f8:    0xfffffff02700e041   "TXM_SLAB_CODE_REGION"
fffffff027019700:    0xfffffff02700e056   "TXM_SLAB_ADDRESS_SPACE"
fffffff027019708:    0xfffffff02700e06d   "TXM_BUCKET_1024"
fffffff027019710:    0xfffffff02700e07d   "TXM_BUCKET_2048"
fffffff027019718:    0xfffffff02700e08d   "TXM_BUCKET_4096"          TXM bucket (generic memory) allocator
fffffff027019720:    0xfffffff02700e09d   "TXM_BUCKET_8192"
fffffff027019728:    0xfffffff02700e0ad   "TXM_BULK_DATA"
fffffff027019730:    0xfffffff02700e0bb   "TXM_BULK_DATA_READ_ONLY"
fffffff027019738:    0xfffffff02700e0d3   "TXM_LOG"
fffffff027019740:    0xfffffff02700e0db   "TXM_SEP_SECURE_CHANNEL"  TXM <-> XNU communication

fffffff027019748:    0xfffffff02700e0f2   "SK_DEFAULT"
fffffff027019750:    0xfffffff02700e0fd   "SK_SHARED_RO"          Secure Kernel pages
fffffff027019758:    0xfffffff02700e10a   "SK_SHARED_RW"
fffffff027019760:    0xfffffff02700e117   "SK_IO"
```

libsptm

- libsptm is a common library for all SPTM-clients, currently v7(!)
- Was annoying to reverse, but Kernel.framework now leaks headers..
- SPTM provides xnu with _sptm_args, communicating state & bootstrap args
- libsptm_state held in sptm_args
- Kernel.framework headers tell the story

Trusted eXecution Monitor

Trust No One.



Trusted eXecution Monitor

- Offloads traditional AMFI responsibilities into Guarded Mode Code
- Implements a “Code Signing Monitor” (CSM) interface
  - AAPL compiles XNU with `CSM_PREFIX`
- `#if CODE_SIGNING_MONITOR`
  supplements `CONFIG_MACF`

bsd/sys/code_signing_internal.h

```
#if CONFIG_SPTM
/* TrustedExecutionMonitor */
#define CODE_SIGNING_MONITOR 1
#define CODE_SIGNING_MONITOR_PREFIX txm
#elif PMAP_CS_PPL_MONITOR
/* Page Protection Layer -- PMAP_CS */
#define CODE_SIGNING_MONITOR 1
#define CODE_SIGNING_MONITOR_PREFIX ppl
#else
/* No monitor -- XNU */
#define CODE_SIGNING_MONITOR 0
#define CODE_SIGNING_MONITOR_PREFIX xnu
#endif /* CONFIG_SPTM */
```

XNU <-> TXM Interface

- All calls conducted through txm_kernel_call, setting up:

```
typedef struct _txm_call {
            /* Input arguments */
/* 0x00 */ TXMKernelSelector_t selector;
/* 0x04 */ TXMReturnCode_t failure_code_silent;
/* 0x06 */ bool failure_fatal;
/* 0x07 */ bool failure_silent;
/* 0x08 */ bool skip_logs;
/* 0x0c */ uint32_t num_input_args;
/* 0x10 */ uint32_t num_output_args;
            /* Output arguments */
            TXMReturn_t txm_ret;
            uint64_t num_return_words;
            uint64_t return_words[kTXMStackReturnWords];
} txm_call_t;
```

- Parameters on stack, as per va_args

TXM

```
kern_return_t
txm_kernel_call(
        txm_call_t *parameters, ...) {
        TXMReturn_t txm_ret = (TXMReturn_t){.returnCode = kTXMReturnGeneric};
        kern_return_t ret = KERN_DENIED;
        va_list args;
        /* Start the variadic arguments list */
        va_start(args, parameters);
        do {
                txm_ret = txm_kernel_call_internal(parameters, args);

                if (txm_ret.returnCode == kTXMReturnOutOfMemory) {
                        if (parameters->selector == kTXMKernelSelectorAddFreeListPage) {
                                panic("received out-of-memory error when adding a free page to TXM");
                        }
                        txm_add_page();
                }
        } while (txm_ret.returnCode == kTXMReturnOutOfMemory);
        /* Clean up the variadic arguments list */
        va_end(args);
        /* Print all TXM logs from the log buffer */
        if (parameters->skip_logs == false) { txm_print_logs(); }


        /* Print the return code from TXM -- only prints for an error */
        if (parameters->failure_silent != true) {
                if (parameters->failure_code_silent != txm_ret.returnCode) {
                        txm_print_return(parameters->selector, txm_ret);
                }
        }
...
```

```
static void
get_trust_cache_info(void) {
        txm_call_t txm_call = {
                .selector = kTXMKernelSelectorGetTrustCacheInfo,
                .failure_fatal = true,
                .num_output_args = 4
        };
        txm_kernel_call(&txm_call);
        /*
         * The monitor returns the libTrustCache runtime it uses within the first
         * returned word. The kernel doesn't currently have a use-case for this, so
         * we don't use it. But we continue to return this value from the monitor
         * in case it ever comes in use later down the line.
         */
        txm_static_trust_caches = (uint32_t)txm_call.return_words[1];
        static_trust_cache_capabilities0 = (TCCapabilities_t)txm_call.return_words[2];
        static_trust_cache_capabilities1 = (TCCapabilities_t)txm_call.return_words[3];
}
void
trust_cache_runtime_init(void) {
        /* Image4 interface needs to be available */
        if (img4if == NULL) { panic("image4 interface not available"); }

        /* AMFI interface needs to be available */
        if (amfi == NULL) { panic("amfi interface not available"); }
        else if (amfi->TrustCache.version < 2) {
                panic("amfi interface is stale: %u", amfi->TrustCache.version);
        }
        /* Acquire trust cache information from the monitor */
        get_trust_cache_info();
}
```

XNU <-> TXM Interface

- All calls conducted through `txm_kernel_call`, setting up:

```
morpheus@eM1nent (~/…/Firmware/19) %disarm -r  _txm_enter /tmp/extracted/kernel.rebuilt
_txm_enter:
fffffe0008adb900 d503237f PACIBSP ;
fffffe0008adb904 2a0003f0 MOVr W16, W0 ; X16 = X0 (0x0 - (null).. Rd Code 0)
fffffe0008adb908 f2e00050 MOVK X16, #2, LSL #48 ; X16 := 0x2000000000000
fffffe0008adb90c f2c00010 MOVK X16, #0, LSL #32 ; X16 := 0x2000000000000
fffffe0008adb910 aa0103ea MOVr X10, X1 ; X10 = X1 (0x0 - (null).. Rd Code 0)
fffffe0008adb914 a9400540 LDP X0, X1, [X10] ; [X0, X1] = *[X10]
fffffe0008adb918 a9410d42 LDP X2, X3, [X10, #16] ; [X2, X3] = *[X10 + 16]
fffffe0008adb91c a9421544 LDP X4, X5, [X10, #32] ; [X4, X5] = *[X10 + 32]
fffffe0008adb920 a9431d46 LDP X6, X7, [X10, #48] ; [X6, X7] = *[X10 + 48]
fffffe0008adb924 00201420 GENTER #0 ;
fffffe0008adb928 d65f0fff RETAB ;
```

Note packing of domain (2) with value of 0, since actual selector is in stack structure

XNU <-> TXM Interface

```
// XNU/TXM communication
TXMKernelSelectorSetSharedRegionBaseAddress,
TXMKernelSelectorGetSecureChannelAddr,
TXMKernelSelectorGetLogInfo,

// Dev mode, Lockdown, Restricted Execution, etc
TXMKernelSelectorDeveloperModeToggle,       // 29
TXMKernelSelectorEnterLockdownMode,
TXMKernelSelectorEnableRestrictedMode,
TXMSelectorUpdateDeviceState,
TXMSelectorCompleteSecurityBootMode,

// Provisioning Profiles
TXMKernelSelector[Un]RegisterProvisioningProfile,
TXMKernelSelectorTrustProvisioningProfile,
TXMKernelSelector[Dis]AssociateProvisioningProfile,
TXMKernelSelectorAuthorizeCompilationServiceCDHash,
TXMKernelSelectorMatchCompilationServiceCDHash,
TXMKernelSelector[Get/Set]LocalSigningPublicKey,
TXMKernelSelectorAuthorizeLocalSigningCDHash,
```

XNU <-> TXM Interface

```
// Code Signing Subsystem
kTXMKernelSelectorCheckTrustCacheRuntimeForUUID,
TXMKernelSelectorGetCodeSigningInfo,
TXMKernelSelector[Un]RegisterCodeSignature,
TXMKernelSelectorValidateCodeSignature,
TXMKernelSelectorReconstituteCodeSignature,
TXMKernelSelector[Un]RegisterAddressSpace,   // registers a process address space with TXM
TXMKernelSelectorAssociateCodeSignature,     // Tie a code signature to previously assoc. Address Space

// JIT
TXMKernelSelectorAllowJITRegion,             // Enable previously associate JIT region
TXMKernelSelectorAssociate[JIT/Debug]Region,
TXMKernelSelectorAllowInvalidCode,           // 39 - vm_map_cs_wx_enable
TXMKernelSelectorAcquireSigningIdentifier,

//Entitlements
TXMKernelSelectorAssociateKernelEntitlements,
TXMKernelSelectorResolveKernelEntitlementsAddressSpace,
TXMKernelSelectorAccelerateEntitlements,


// Image 4 subsystem
TXMKernelSelectorImage4[Get/Set/Roll]Nonce,
TXMKernelSelectorImage4GetExports,
TXMKernelSelectorImage4SetReleaseType,
TXMKernelSelectorImage4SetBootNonceShadow
```

TXM<->SPTM Interface

- TXM calls SPTM through SVC #0 (from GL0 to GL1/2)
  - Calls subsystem #1: (0x100000001-0x100000004)
    - 0x100000001: lock-down/post fixups
    - 0x100000002: Register dispatch table
    - 0x100000003: page retyping
    - 0x100000004: Address Space related (unclear)
  - Call 0xfd00000000 to pass through back to caller (i.e XNU/AMFI)
  - Call 0xfe00000000 to panic
  - Call 0xff00000000 to save state

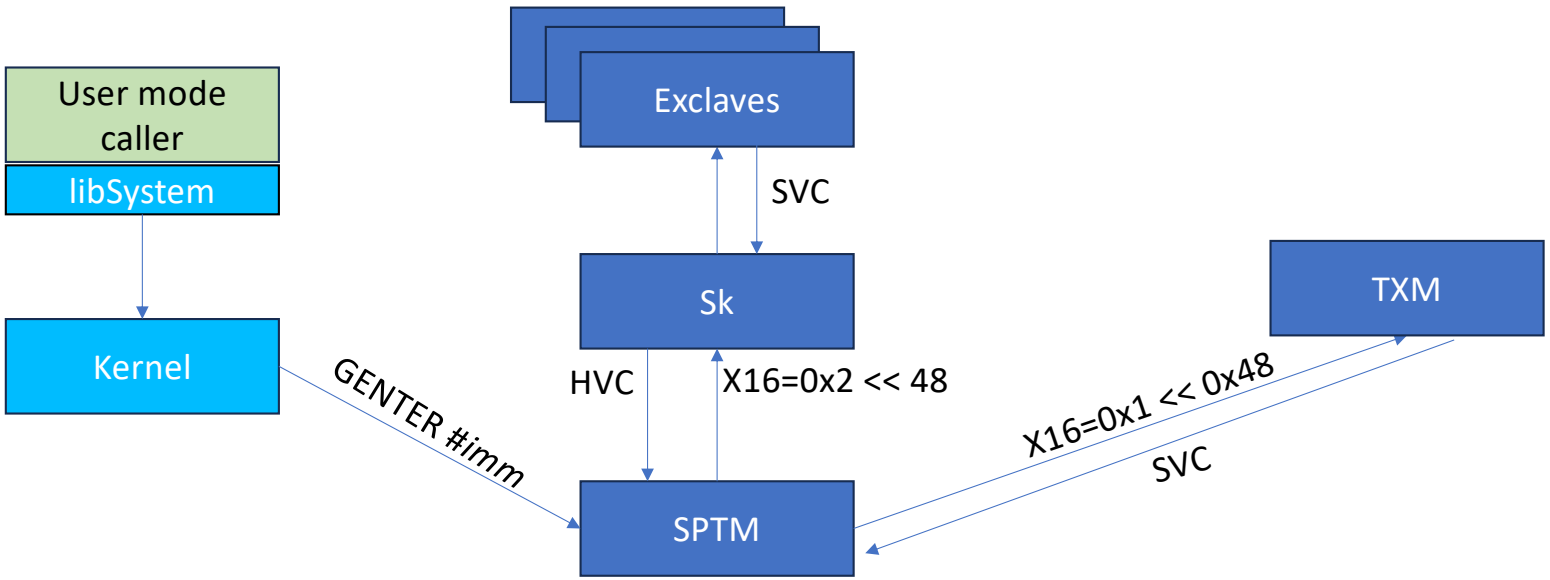
TXM<->SPTM Interface

```
morpheus@eM1nent (~/Downloads/Firmware/19/Firmware) %disarm -r txm_endpoints
sptm.t8150.release.im4p
txm_endpoints:
fffffff02701d250: 00 00 00 00 00 00 00 00    |........|
fffffff02701d258: 00 00 00 00 00 00 00 00    |........|
fffffff02701d260:    0xfffffff0270b4a6c   _sptm_txm_lockdown
fffffff02701d268: 00 00 00 00 00 00 00 00    |........|
fffffff02701d270:    0xfffffff0270db600   _sptm_register_dispatch_table
fffffff02701d278: 00 00 00 00 00 00 00 00    |........|
fffffff02701d280:    0xfffffff0270e2624   _sptm_retype_page
fffffff02701d288: 00 00 00 00 00 00 00 00    |........|
fffffff02701d290:    0xfffffff0270eb644   _sptm_address_space_related
```

And, in TXM:

```
_func_0xfffffff01704d90c:
fffffff01704d90c        d503245f        BTI     c       ;
fffffff01704d910        f2e00010        MOVK    X16, #0, LSL #48        ; X16 := 0x0
fffffff01704d914        f2c00030        MOVK    X16, #1, LSL #32        ; X16 := 0x100000000
fffffff01704d918        f2a00010        MOVK    X16, #0, LSL #16        ; X16 := 0x100000000
fffffff01704d91c        f2800070        MOVK    X16, #3 ; X16 := 0x100000003
        _sptm_enter(0);
fffffff01704d920        140059b8        B       0xfffffff017064000      ; _sptm_enter
...
```

Exclaves


User mode caller
libSystem
Kernel
GENTER #imm
Exclaves
SVC
Sk
HVC
X16=0x2 << 48
SPTM
TXM
X16=0x1 << 0x48
SVC

Some Nomenclature

- Enclave: Think: Vatican, San Marino, Lesotho.
  - Now think – SEP, as a detached, isolated, secure enclave processor

- Exclave: Think: Kalilingrad*

  Now think, Detached address spaces that are logically connected

- Conclave: Think: Pontifex Leo XIV's electors.

  Now think, segregated address spaces with TightBeam™ IPC messaging

* - If you're reading this after Oct 15th 2026, I can't guarantee it's still an exclave..

ExclaveKit IPSW

- A16+ IPSWs contain (yet-another) (AEAed) DMG
- Numbering will vary, but it's usually 3rd
- Get AEA key*, Decrypt, Mount

```
morpheus@eM1nent (/Volumes/Luck23A341.D23ExclaveOS/System/ExclaveKit) %ls
System usr
morpheus@eM1nent (/Volumes/Luck23A341.D23ExclaveOS/System/ExclaveKit) %ls usr/lib
/usr/lib/dyld
morpheus@eM1nent (/Volumes/Luck23A341.D23ExclaveOS/System/ExclaveKit) %ls usr/bin
Tightbeam_stub
morpheus@eM1nent (/Volumes/Luck23A341.D23ExclaveOS/System/ExclaveKit) %ls -F System/Library/
Frameworks/        PrivateFrameworks/ dyld/
morpheus@eM1nent (/Volumes/…/ExclaveKit) %disarm -L usr/lib/dyld| grep LC_BUILD_VER
      LC 11: LC_BUILD_VERSION      Build Version:           Platform: ExclaveOS 26.0.0 SDK: 26
```

* - Why, AAPL, WHY? Haven't we learned anything from the OTA saga?

ExclaveCore image

- A16+ IPSWs contain an “exclavecore_bundle” IM4P

```
morpheus@eM1nent (~/Downloads) %unzip -j iPhone18,4_26.0_23A341_Restore.ipsw "*exclave*"
Archive:  iPhone18,4_26.0_23A341_Restore.ipsw
  inflating: exclavecore_bundle.t8150.RELEASE.im4p
  inflating: exclavecore_bundle.t8150.RELEASE.restore.im4p

morpheus@eM1nent (~/Downloads) %hexdump -C exclavecore_bundle.t8150.RELEASE.im4p | head
00000000  30 84 01 dd 40 e4 16 04  49 4d 34 50 16 04 65 78  |0...@...IM4P..ex|
00000010  63 6c 16 01 31 04 84 01  dd 40 00 00 02 00 14 00  |cl..1....@......|
00000020  00 00 00 44 4e 55 42 00  00 03 00 db 8c 0e 4b 87  |...DNUB.......K.|
00000030  d3 39 5b 81 54 0b 12 e7  2a a9 bb 00 00 00 00 00  |.9[.T...*.......|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 01 00 00 00 00  |................|
00000060  00 00 00 00 00 00 00 00  00 00 00 00 40 dd 01 00  |............@...|
*
```

- TL;DR: IM4P is DER encoded, with metadata at end

ExclaveCore Image Components

- `imjtool(j)` can be used to extract components:

```
morpheus@eM1nent (~/Downloads) % imjtool exclavecore_bundle.t8150.RELEASE.im4p
exclavecore_bundle.t8150.RELEASE.im4p: Apple IM4P (DER) container Tag: excl
AAPL BUNDle
Component: kernel
Component: roottask
Component: ExclaveStackshotServer
Component: pmm_exclave
Component: sharedcache
```

- Using `-v` will provide details of component DER metadata
- Specifying `JDEBUG=1` will walk through component reassembly

ExclaveCore Image Components

kernel

- True microkernel(!): 224K binary, of which only 70K code
- Referred to as “SK” and/or “cL4”
  - Binary intentionally stripped, with error messages as just file:line
- Not directly related to SeL4, but very likely inspired by L4
- Runs in GL1, and provides “capabilities” to user space
- SVCs #0-5

ExclaveCore Image Components

Swifties

- `roottask`: “PID 1” – initial user mode launcher (loads rest)
- `pmm_exclave`
- `ExclaveStackshotServer`
- `sharedcache`
- `xnuproxy` (MIA since early 18.0… possibly moved to sharedcache)

- All statically linked with liblibc.

XNU <-> SK Interface

```
morpheus@eM1nent (~/…/Firmware/19) %disarm -r  _sk_enter /tmp/extracted/kernel.rebuilt
_sk_enter:
fffffe0008adb8ac        d503237f        PACIBSP ;
fffffe0008adb8b0        a9bf7bfd        STP     X29, X30, [X31, #-16]!  ; SP -= 16; *[SP] = [X29, X30]
fffffe0008adb8b4        910003fd        MOVr    X29, X31        ; FP = SP
        _func_0xfffffe0008adcb48(0);
fffffe0008adb8b8        940004a4        BL      0xfffffe0008adcb48      ;
fffffe0008adb8bc        910003bf        MOVr    X31, X29        ; SP = FP (0x0 - (null).. Rd Code 0)
fffffe0008adb8c0        a8c17bfd        LDP     X29, X30, [X31], #16    ; [X29, X30] = *[X31 + 16]; X31 += 2
fffffe0008adb8c4        2a0003f0        MOVr    W16, W0 ; X16 = X0 (0x0 - (null).. Rd Code 0)
fffffe0008adb8c8        f2e00070        MOVK    X16, #3, LSL #48        ; X16 := 0x3000000000000
fffffe0008adb8cc        f2c00010        MOVK    X16, #0, LSL #32        ; X16 := 0x3000000000000
fffffe0008adb8d0        aa0103ea        MOVr    X10, X1 ; X10 = X1 (0x0 - (null).. Rd Code 0)
fffffe0008adb8d4        a9400540        LDP     X0, X1, [X10]   ; [X0, X1] = *[X10]
fffffe0008adb8d8        a9410d42        LDP     X2, X3, [X10, #16]      ; [X2, X3] = *[X10 + 16]
fffffe0008adb8dc        a9421544        LDP     X4, X5, [X10, #32]      ; [X4, X5] = *[X10 + 32]
fffffe0008adb8e0        a9431d46        LDP     X6, X7, [X10, #48]      ; [X6, X7] = *[X10 + 48]
fffffe0008adb8e4        00201420        GENTER  #0      ;
fffffe0008adb8e8        a9bf7bfd        STP     X29, X30, [X31, #-16]!  ; SP -= 16; *[SP] = [X29, X30]
fffffe0008adb8ec        910003fd        MOVr    X29, X31        ; FP = SP
        _func_0xfffffe0008adcb50(0);
fffffe0008adb8f0        94000498        BL      0xfffffe0008adcb50      ;
fffffe0008adb8f4        910003bf        MOVr    X31, X29        ; SP = FP (0x0 - (null).. Rd Code 0)
fffffe0008adb8f8        a8c17bfd        LDP     X29, X30, [X31], #16    ; [X29, X30] = *[X31 + 16]; X31 += 2
fffffe0008adb8fc        d65f0fff        RETAB   ;
```

exclaves_ctl_trap

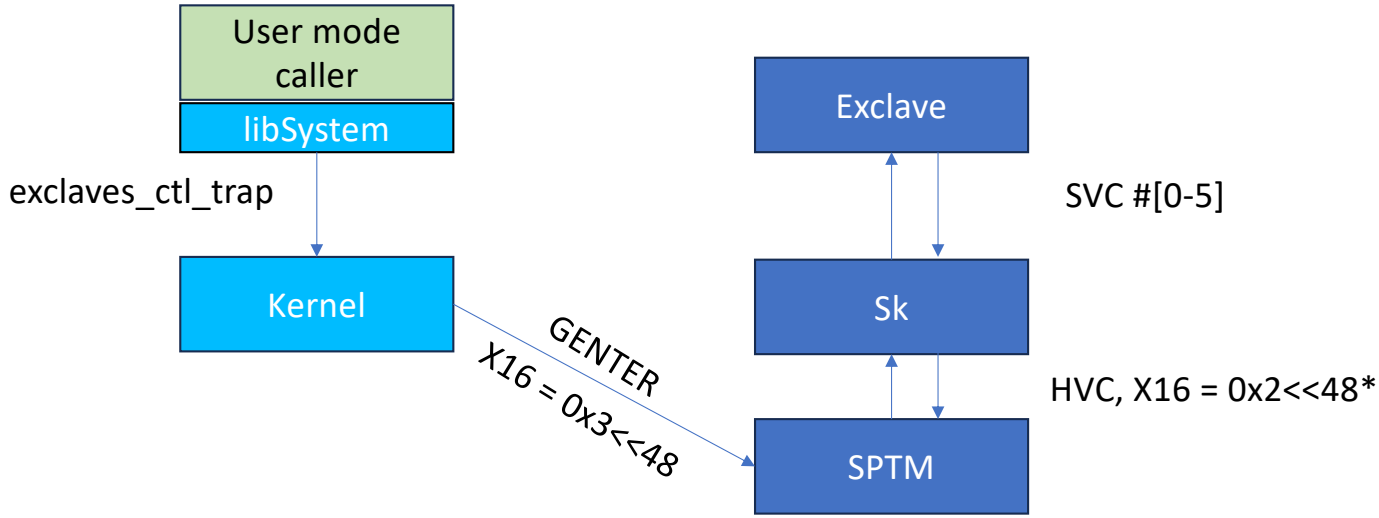
Op	EXCLAVES_CTL_OP_*	Wrapper
1	_ENDPOINT_CALL	exclaves_endpoint_call
		exclaves_[in/out]bound_buffer_*
2,3,4	_NAMED_BUFFER_[CREATE/COPYIN/COPYOUT]	exclaves_named_buffer_*
5	BOOT	exclaves_boot
8,9	AUDIO_BUFFER_CREATE/COPYOUT	exclaves_audio_buffer_*
10,11,12,13	SENSOR_CREATE/START/STOP/STATUS	exclaves_sensor_[create/start/stop/status]
6	**LAUNCH_CONCLAVE**	exclaves_launch_conclave
7	LOOKUP_SERVICES	exclaves_lookup_service
14	NOTIFICATION_RESOURCE_LOOKUP	exclaves_notification_create
		exclaves_[allocate/free/get]_ipc_buffer

Currently only for sensors, Camera/MIC indicators, but with unbounded potential!

exclaves_ctl_trap (#88)

`osfmk/kern/exclaves.c`

- New Mach Trap for Exclave access and/or control from user mode
- Wrapped by libSystem APIs



* - Also 0x4 <<48, for SK DART

That's all, folks!

Do I still have time?*

- Long overdue Blog post be published at https://df-f.com/blog
  - With `disarm(j)` companion files and matcher rules
  - License: Free, just please cite/give-credit-where-due/spread the word of DF-F
- Till then: This presentation @ https://Technologeeks.com/files/MXlA.pdf
- Questions? (if I still have time, ask now, else catch me later)
- And no, thank you, I'm still Done With Darwin.
  - This was a one-off for Patrick – Thanks, man. It was fun, Andy & you %$#@ing ROCK.

* – Probably not. Damn it. I knew it. Sorry, Andy. Pat, please don't shoot me..