Inside The Apple T2

Mikhail Davidov

Research Technical Leader, Duo Labs
mdavidov@duo.com @sirus

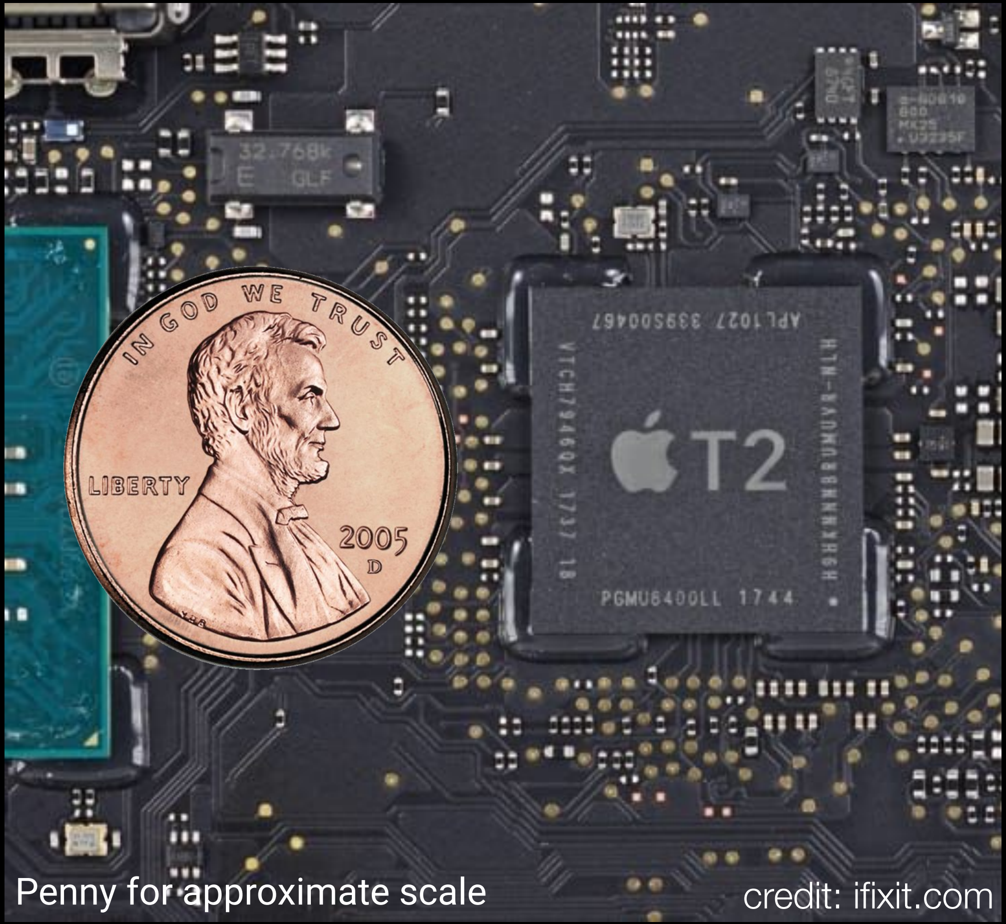
Jeremy Erickson

Senior Research Engineer, Duo Labs
jerickson@duo.com @jlericks

Duo Security is now part of Cisco.

Agenda

1) T2 Objectives
2) T2 Architecture
3) BridgeOS Static Analysis
4) Secure Boot: Past and Present
5) The Boot Process
6) Exposed T2 Services
7) Communication Channel
8) Decoding Message Layers
9) Decoding XPC
10) Listening in on T2 Services
11) Interacting with T2 Services



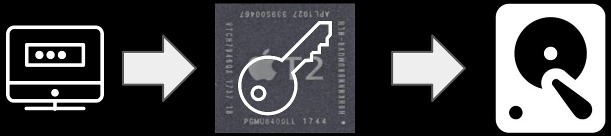
Penny for approximate scale

credit: ifixit.com

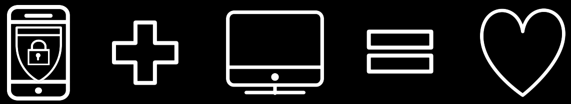
T2 Objectives

Enhance privacy controls for peripherals through physical data disconnects.

Better protect data at rest by mixing in key material stored in a secure element.



Make macOS boot as securely as iOS by closing UEFI security gaps.



Why investigate the T2?

The T2 chip has far-reaching impact across the security space and gives us a glimpse of where secure boot is headed.

Historically, there's been limited information available on the internal workings of Apple's hardware and software.

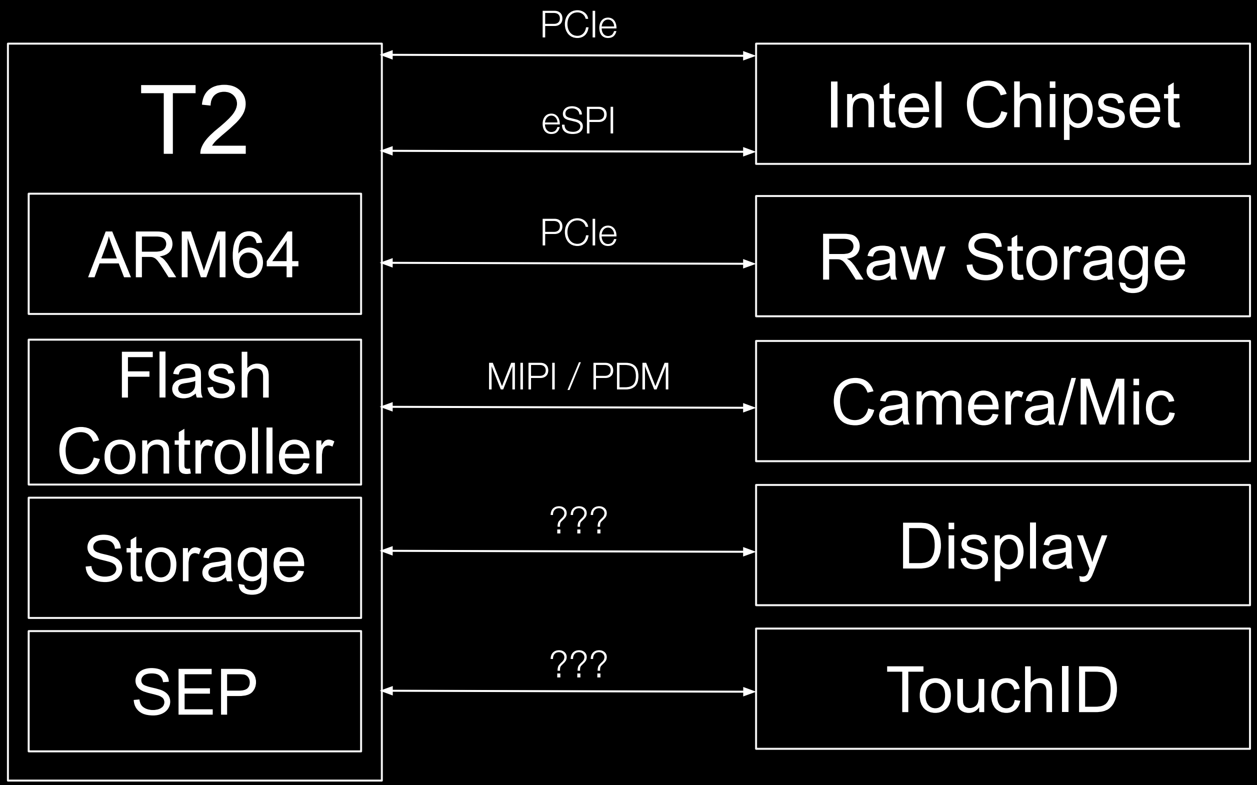
More eyes on any critical piece of technology will help uncover vulnerabilities.

T2 Architecture

Intel Embedded Controllers

- What the T2 is referred to as in the Intel world.
- Baseboard Management Controller (BMC) minus the remote management.

- Responsible for general orchestration tasks such as:
    - Power sequencing of components.
    - Thermal management
    - State transitions (S5 -> S0)
    - Peripheral interfacing


T2
ARM64
Flash Controller
Storage
SEP
PCIe
eSPI
PCIe
MIPI / PDM
???
???
Intel Chipset
Raw Storage
Camera/Mic
Display
TouchID

PCIe

eSPI

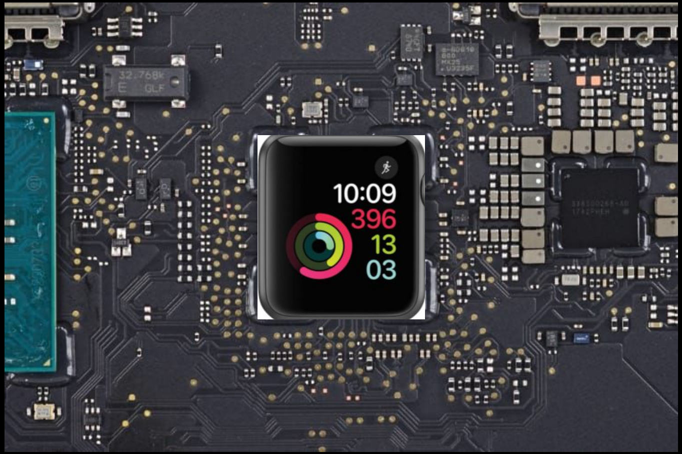
Intel Chipset

PCIe

Raw Storage

BridgeOS

Static Analysis



Examining Firmware

- BridgeOS “OTA” Updates obtainable through Apple’s software catalog.
  - Cached in /Library/Updates as BridgeOSUpdateCustomer.pkg

- Extractable with a combination of pbzx, ota, and joker:
  - http://newosxbook.com/articles/BridgeOS.html

- Full filesystem, kernelcache, and base UEFI image
- iBoot and SEP firmware still encrypted.

```
$ xar -xvf BridgeOSUpdateCustomer.pkg
$ cat Payload | pbzx | cpio -ivd
  …
  ./usr/standalone/firmware/bridgeOSCustomer.bundle/Contents/Resources/UpdateBundle.zip
```

Examining Firmware (Gold UEFI)

```
UpdateBundle.zip/
  boot/
    Firmware/
      MacEFI/
        - J132.RELEASE.im4p
        - J137.RELEASE.im4p
        - J140K.RELEASE.im4p
        - J174.RELEASE.im4p
        - J680.RELEASE.im4p
        - J780.RELEASE.im4p
```

```
$ img4tool -e -o mefi J137.RELEASE.im4p
$ file mefi
   mefi: Intel serial flash for PCH ROM
```

Examining Firmware (Gold UEFI)

```
UpdateBundle.zip/
  boot/
    Firmware/
      MacEFI/
        - J132.RELEASE.im4p   ⟸ ???
        - J137.RELEASE.im4p   ⟸ iMac Pro
        - J140K.RELEASE.im4p  ⟸ ???
        - J174.RELEASE.im4p   ⟸ ???
        - J680.RELEASE.im4p   ⟸ MacBook Pro
        - J780.RELEASE.im4p   ⟸ ???
```

```
$ img4tool -e -o mefi J137.RELEASE.im4p
$ file mefi
   mefi: Intel serial flash for PCH ROM
```

Examining Firmware (Gold UEFI)

UpdateBundle.zip/
boot/
Firmware/
MacEFI/

- J132.RELEASE.im4p ???
- J137.RELEASE.im4p iMac Pro
- J140K.RELEASE.im4p MacBook??
- J174.RELEASE.im4p Mac Mini
- J680.RELEASE.im4p MacBook Pro
- J780.RELEASE.im4p ???

```
$ img4tool -e -o mefi J137.RELEASE.im4p
$ file mefi
   mefi: Intel serial flash for PCH ROM
```

osint

PROJECT EXPERIENCE

Foxconn, China.

[redacted]

Made Products: MacBook (J140)/Mac Mini (J174)

Examining Firmware (Kernelcache)

```
UpdateBundle.zip/
  boot/
    - kernelcache.release.j132
    - kernelcache.release.j137
    - kernelcache.release.j140
    - kernelcache.release.j174
    - kernelcache.release.j680
```

```
$ joker -dec kernelcache.release.j137
$ file /tmp/kernel
  /tmp/kernel: Mach-O 64-bit executable arm64
```

Hex-rays + bazad/ida_kernelcache = IOKit <3

Examining Firmware (Filesystem)

```
UpdateBundle.zip/
  payloadv2/
    - payload.000
    - payload.001
    ...
```

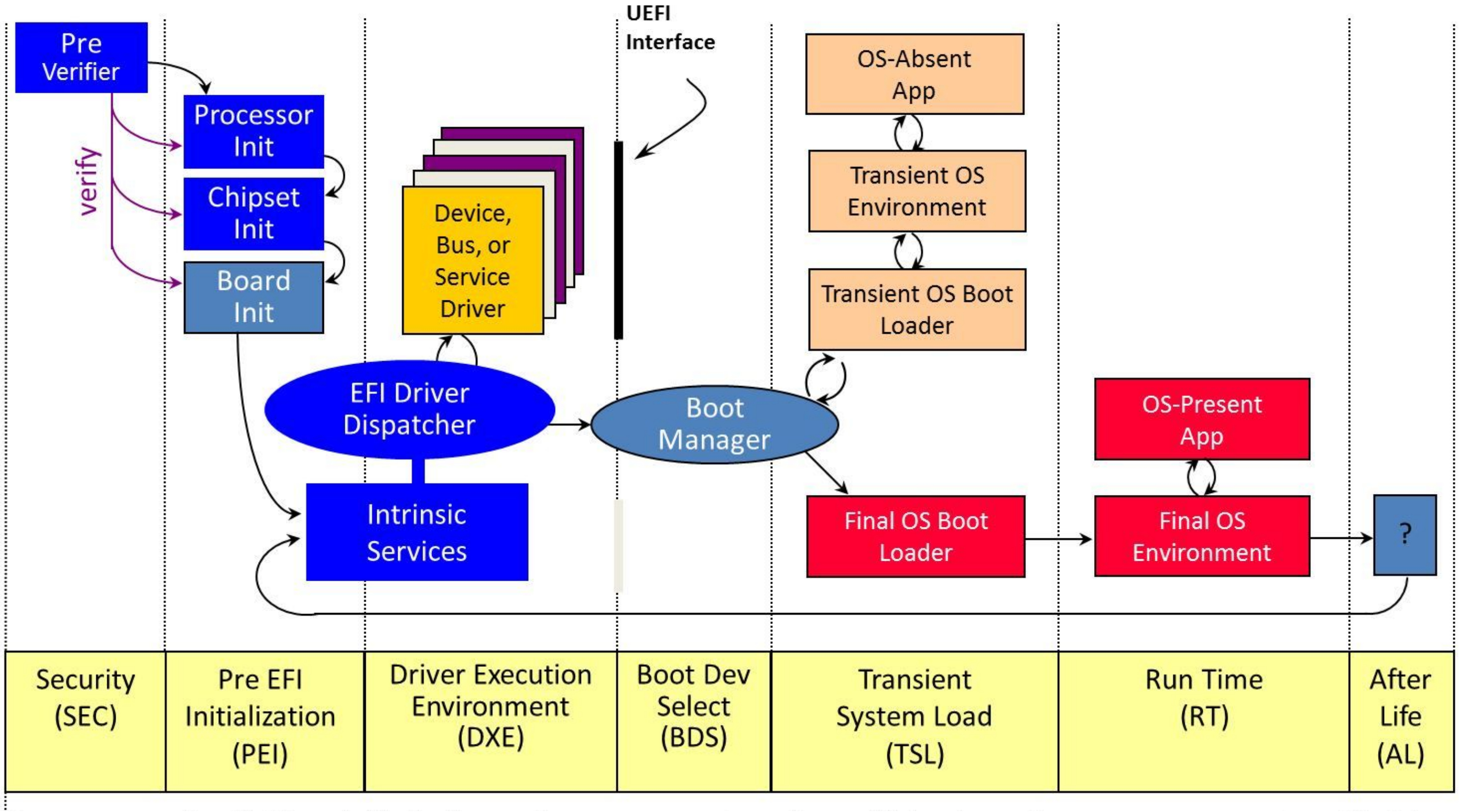
```
$ pbzx payload.000 > ext.000 && pbzx payload.001 > ext.001
$ mkdir ext && cd ext
$ ota -e ‘*’ ../ext.000 && ota -e ‘*’ ../ext.001 && ls -la
```

```
Library System  bin      etc      private sbin      tmp      usr
```

Secure Boot

Past and Present

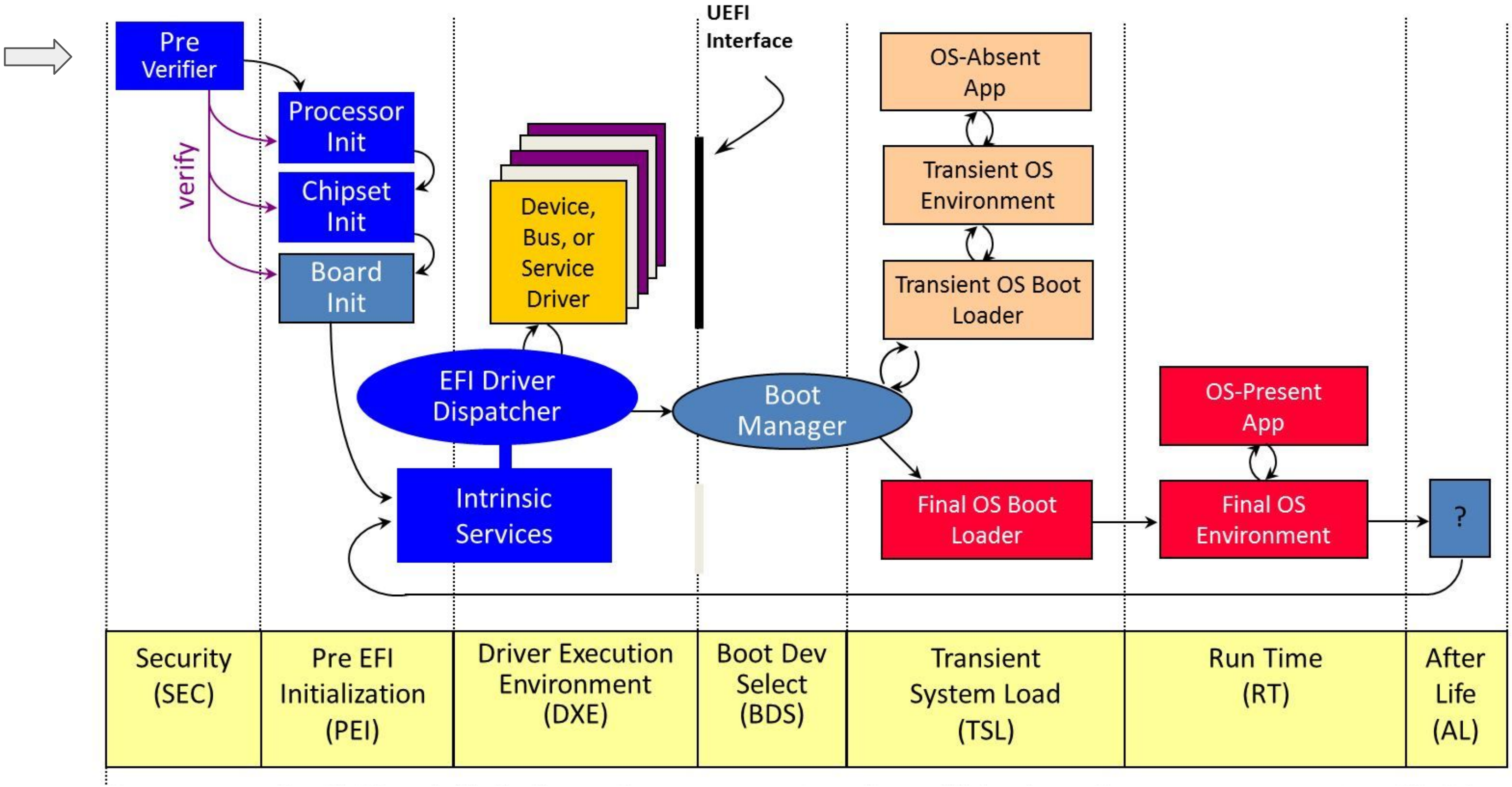
UEFI Platform Initialization (PI) Boot Phases



tianocore.github.io/master/images/PI_Boot_Phases.JPG

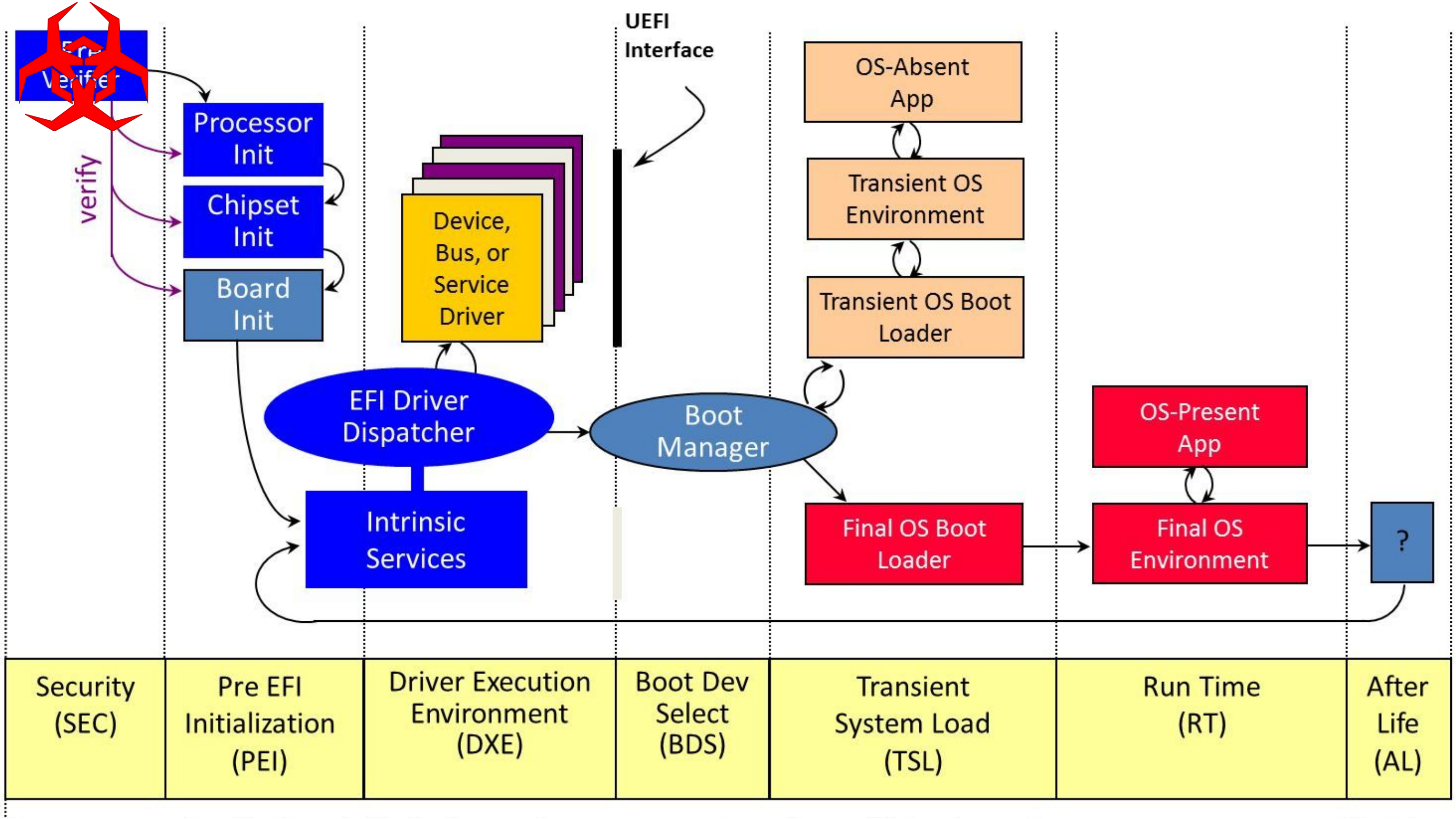
UEFI Platform Initialization (PI) Boot Phases

Reset Vector



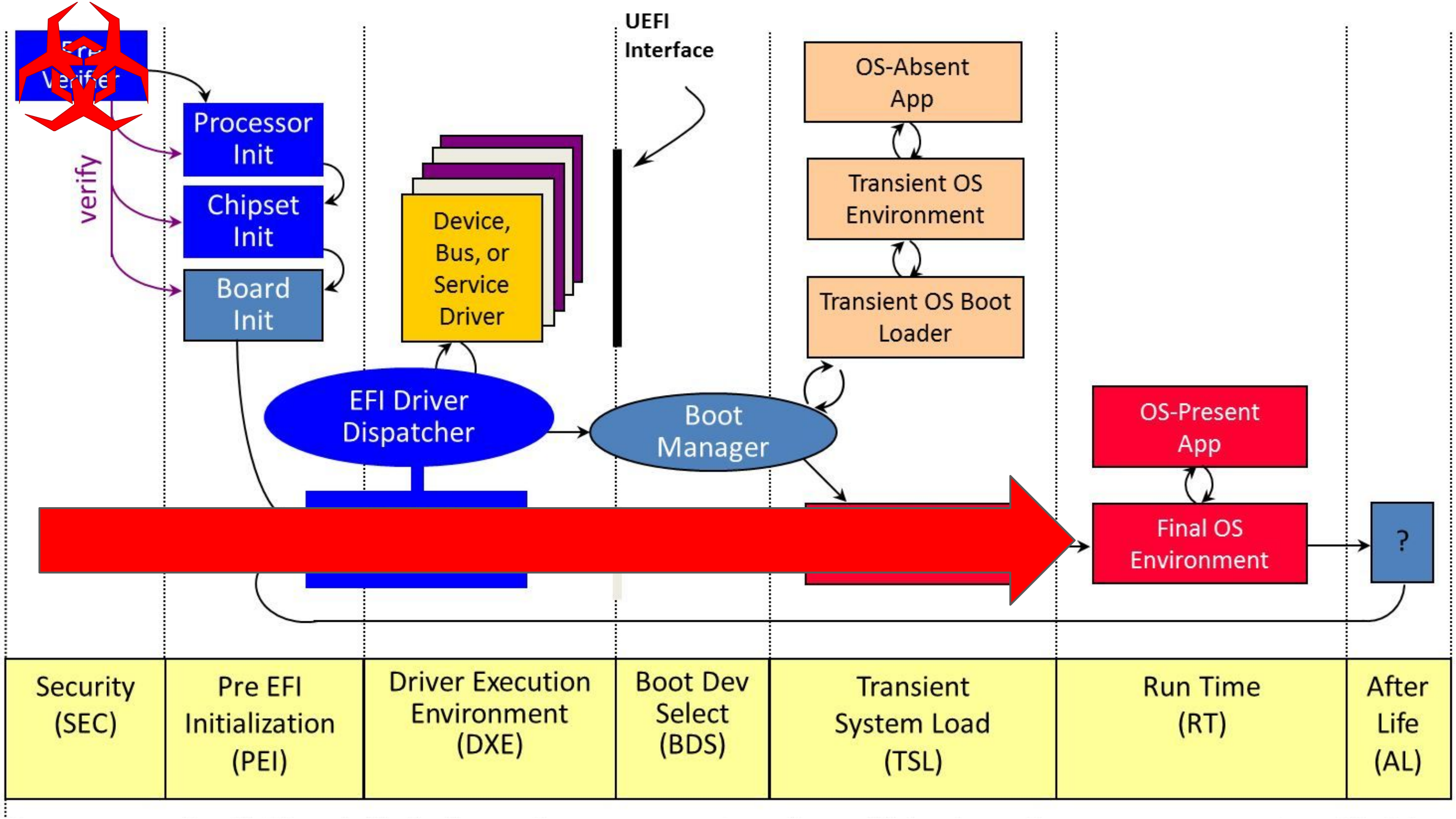
tianocore.github.io/master/images/PI_Boot_Phases.JPG

UEFI Platform Initialization (PI) Boot Phases



tianocore.github.io/master/images/PI_Boot_Phases.JPG

UEFI Platform Initialization (PI) Boot Phases



tianocore.github.io/master/images/PI_Boot_Phases.JPG

Intel Chipset

SPI

Flash Chip

NVARS

UEFI FW

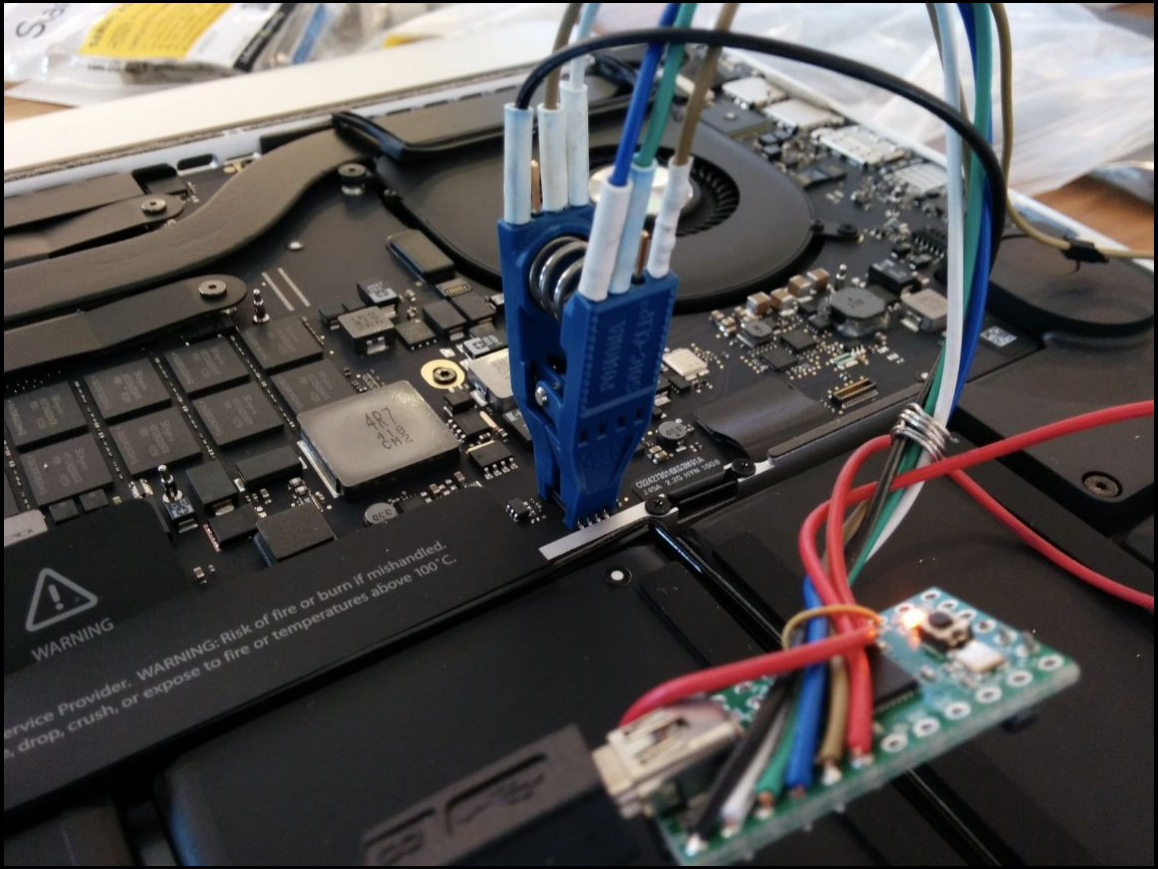


Image: @qrs

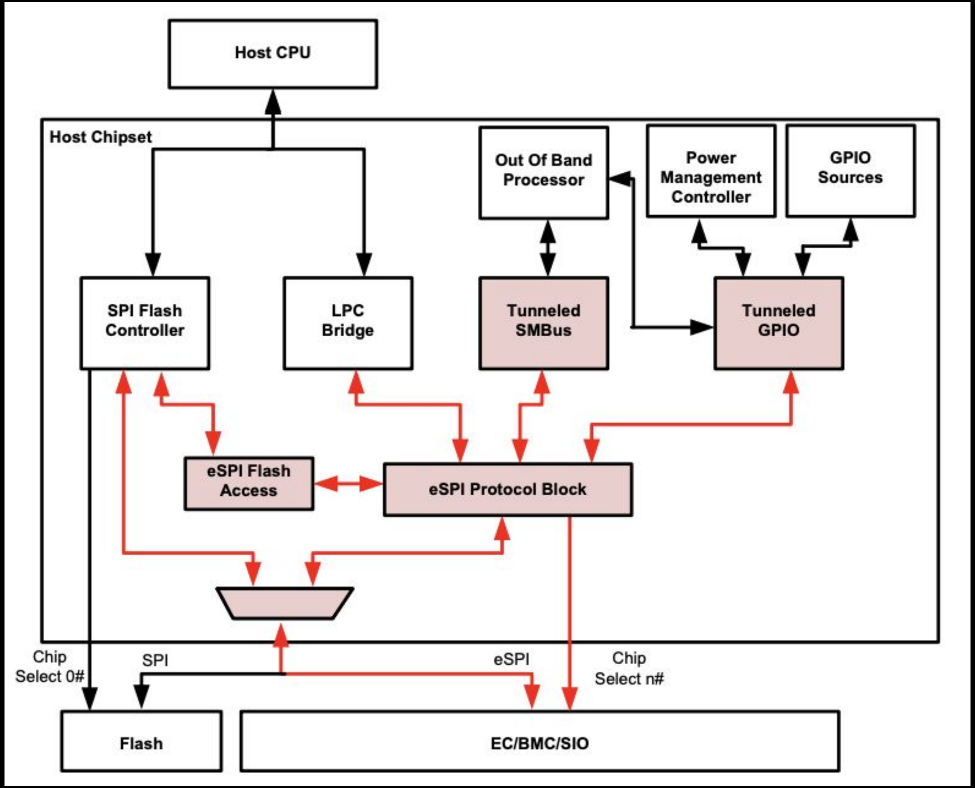
eSPI & Slave Attached Flash

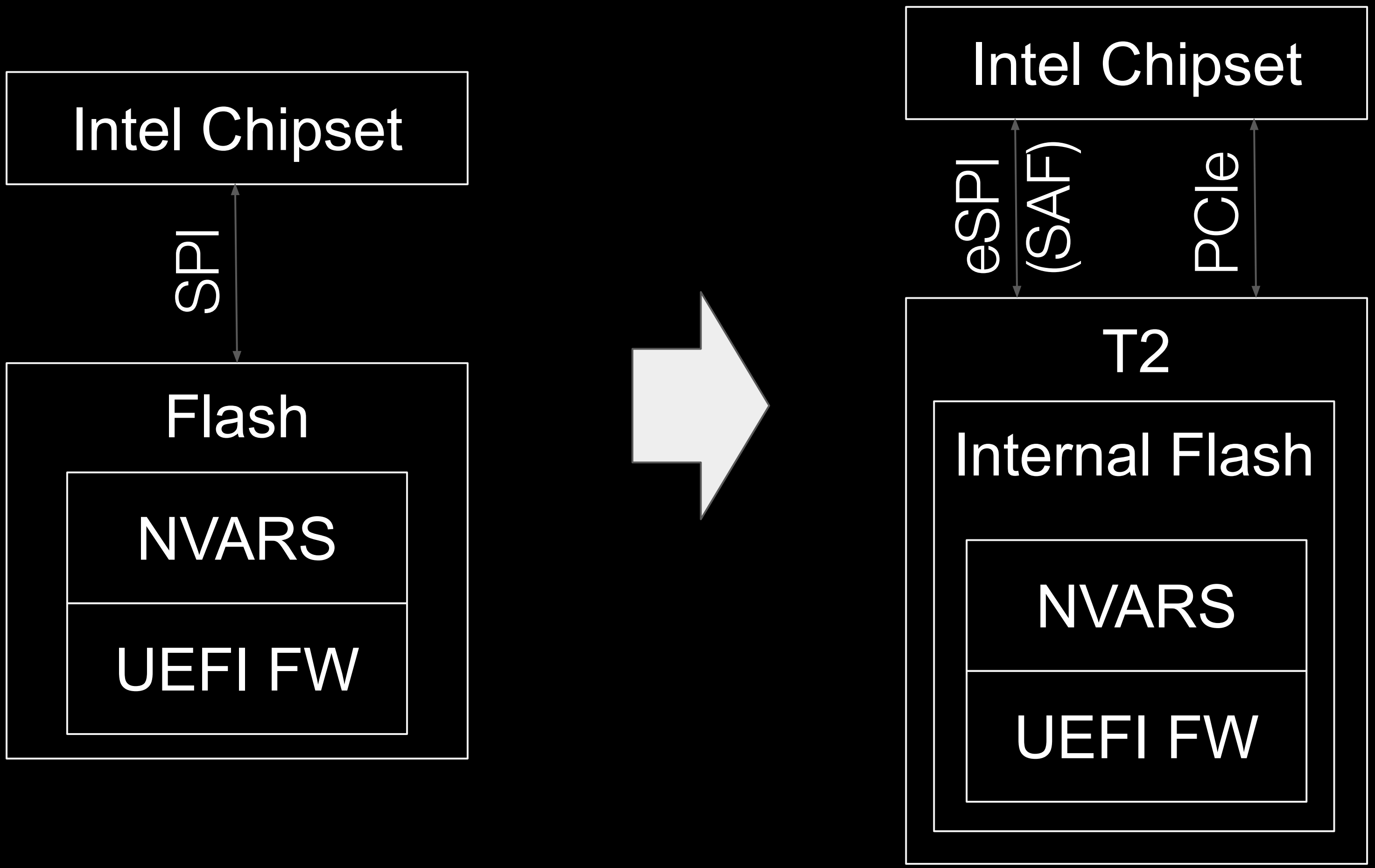
eSPI is the successor to the "Low Pin Count" (LPC) bus.

Recently extended for Xeon platforms with support for Slave Attached Flash (SAF)

Allows BMC/EC to manage all flash access operations.

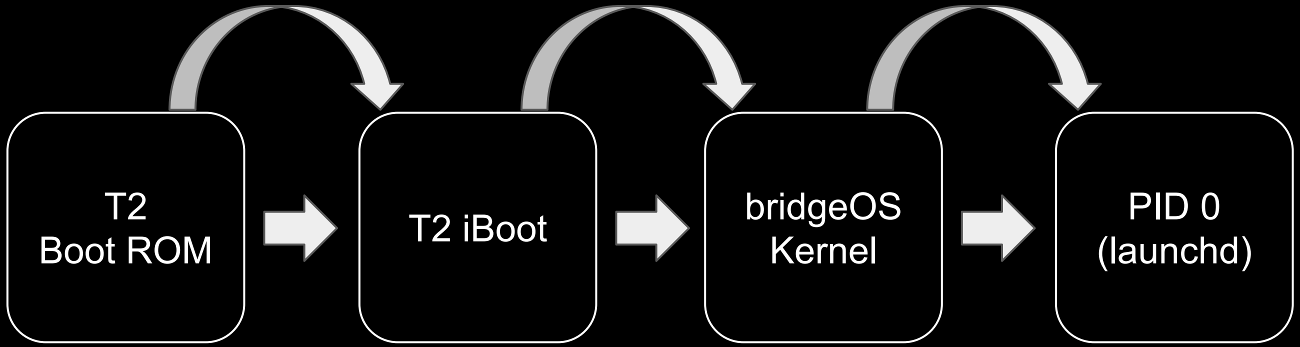
Allows BMC to remotely manage firmware.



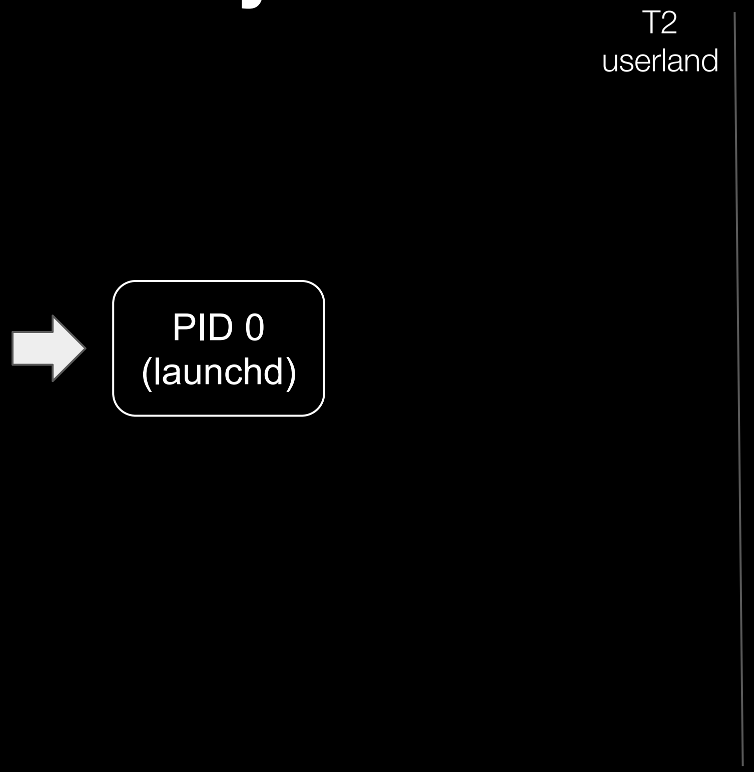

Intel Chipset
SPI
Flash
NVARS
UEFI FW
Intel Chipset
eSPI (SAF)
PCIe
T2
Internal Flash
NVARS
UEFI FW

The Boot Process

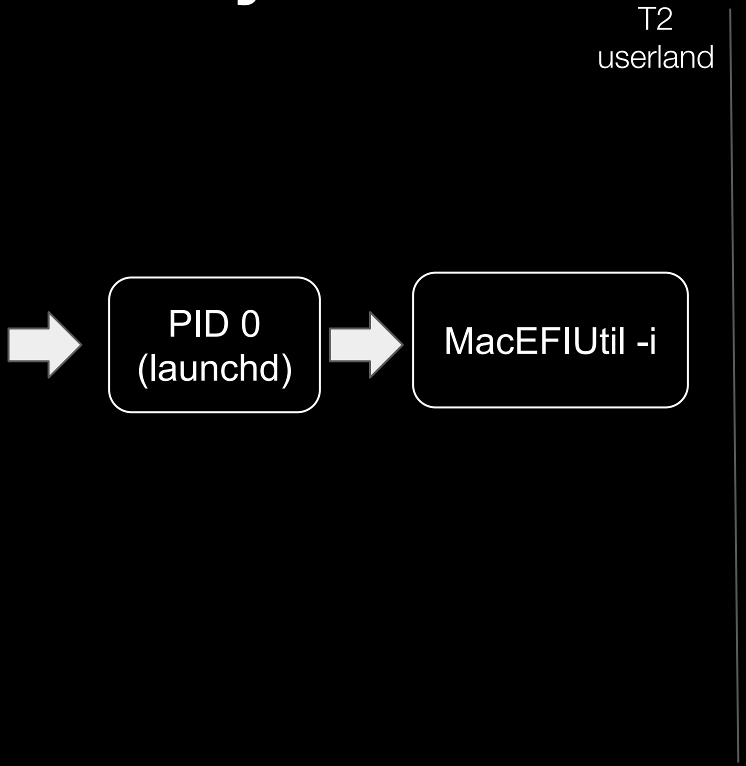
T2 Early Boot



T2 Early Boot



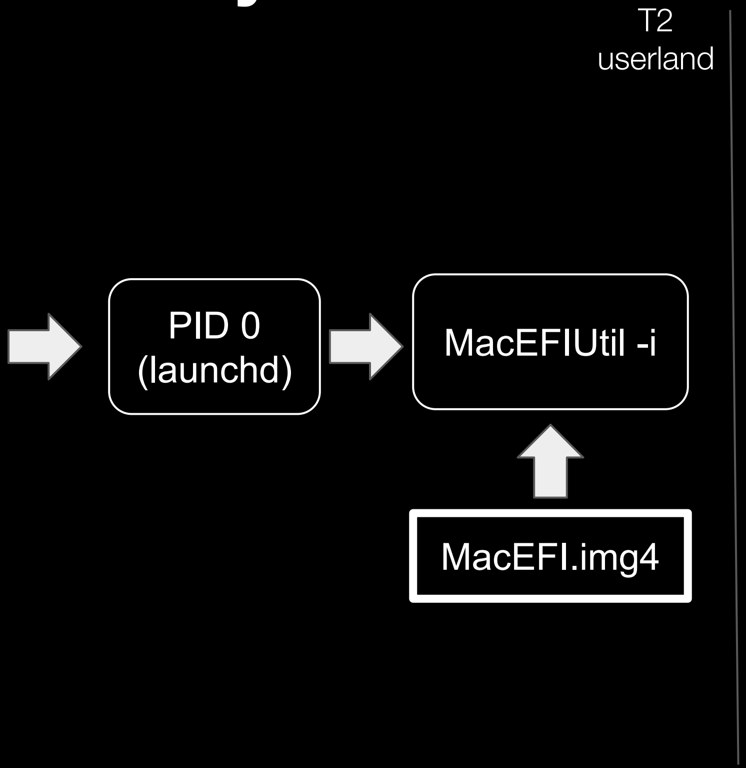
T2 Early Boot



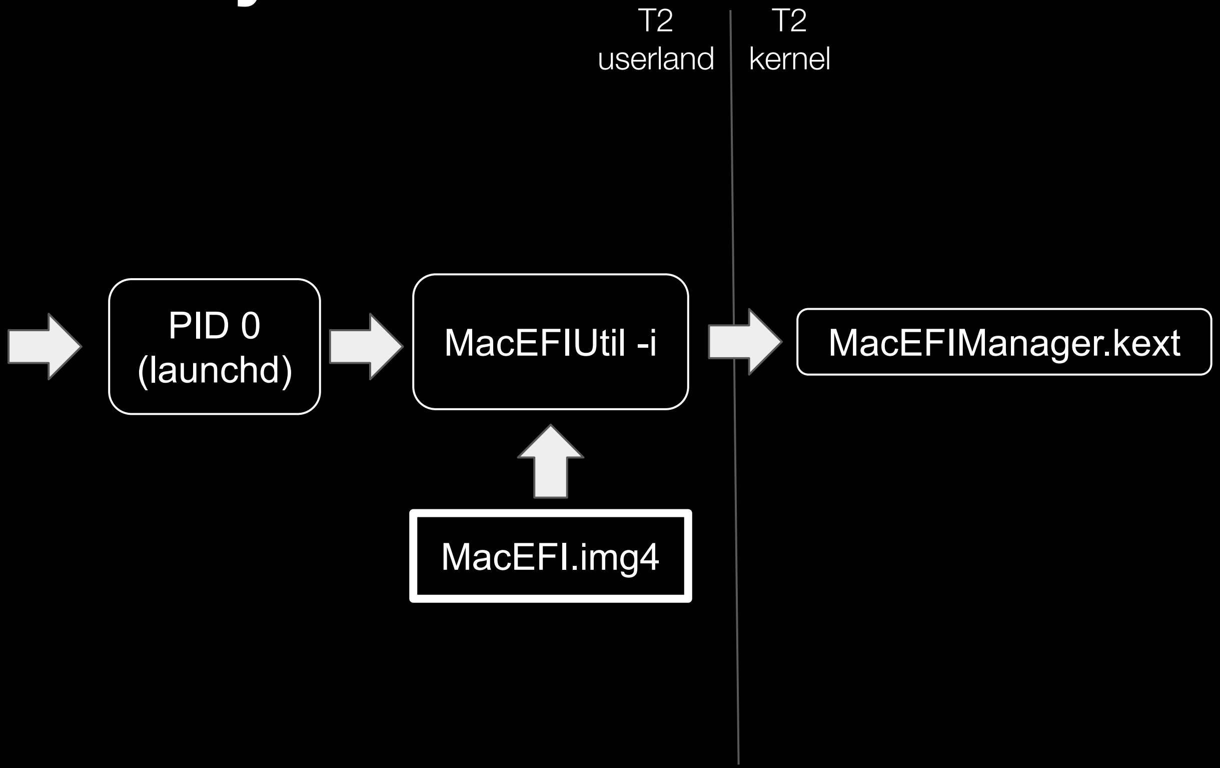
MacEFIUtil Functionality

- Start the UEFI firmware loading process from a signed image
- Read/write NVRAM variables
- Read/write Intel ME partitions:
  - IVBP - bring up cache
  - MFS - ME flash filesystem
  - FLOG - Flash log
  - UTOK - Debug unlock token
  - UEP - “Unified Emulation Partition”
  - SWBG - ???

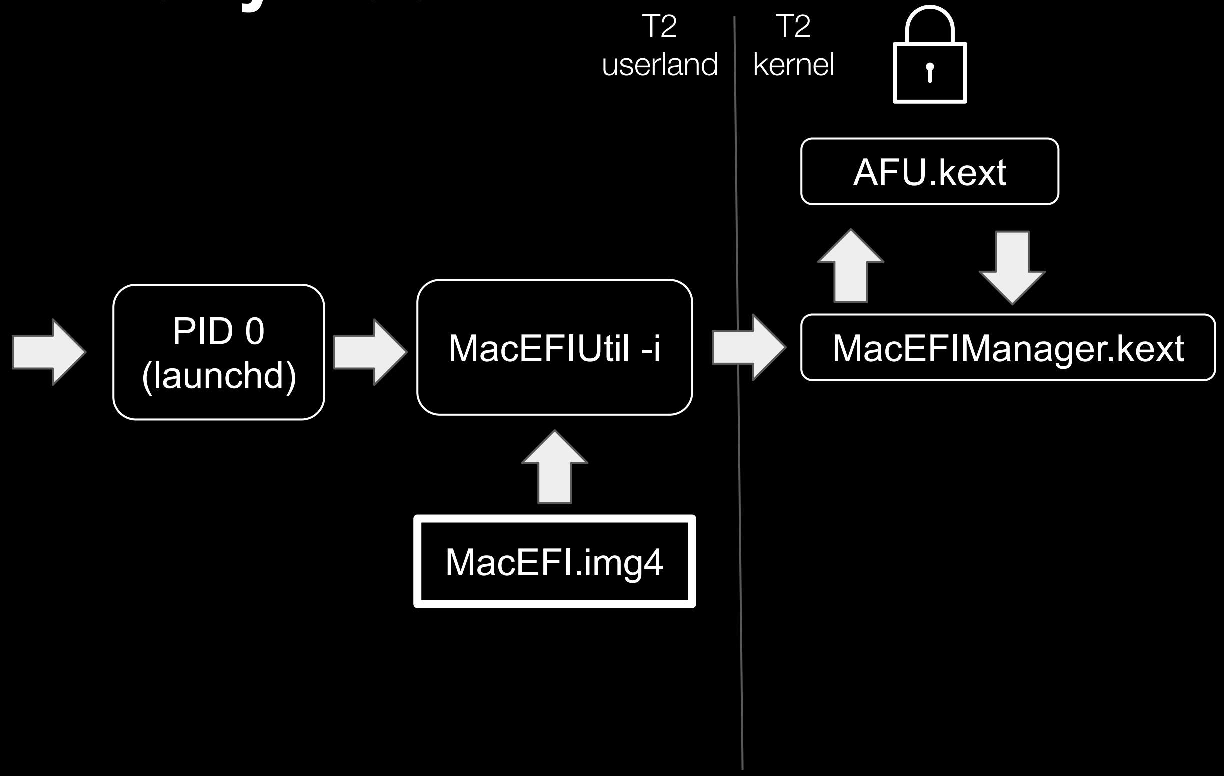
T2 Early Boot



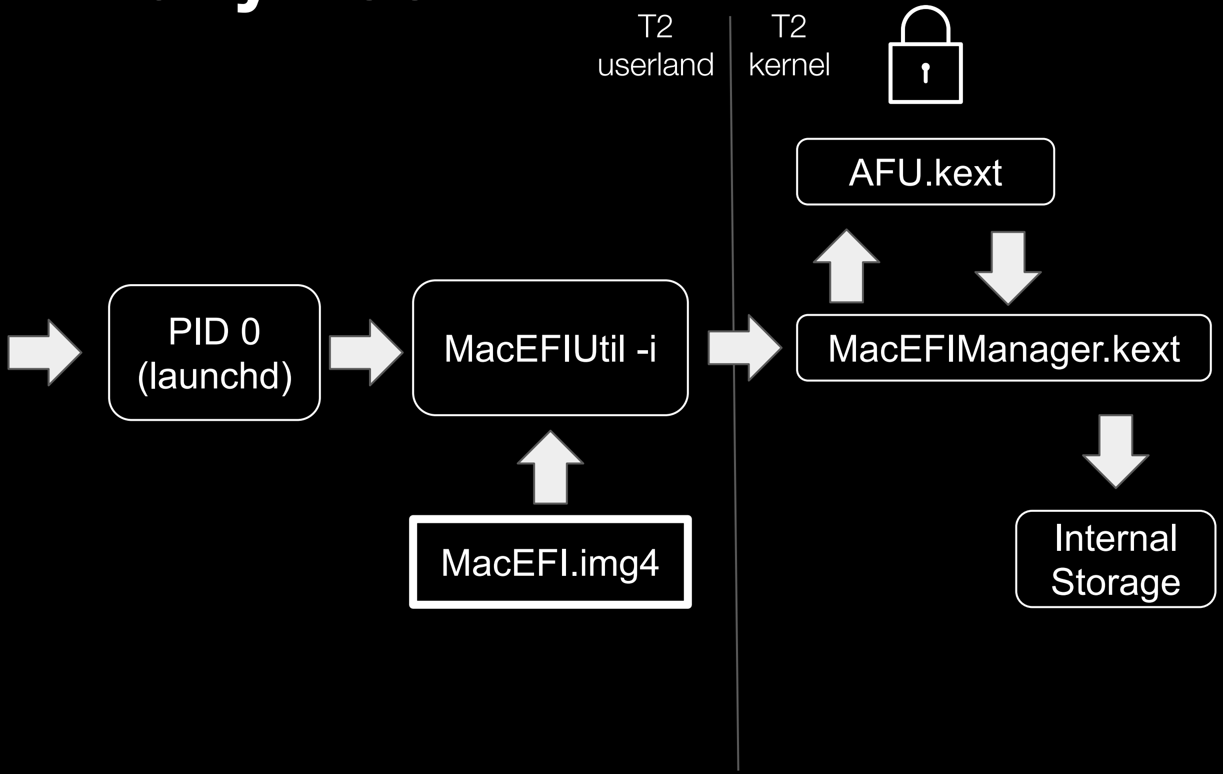
T2 Early Boot



T2 Early Boot

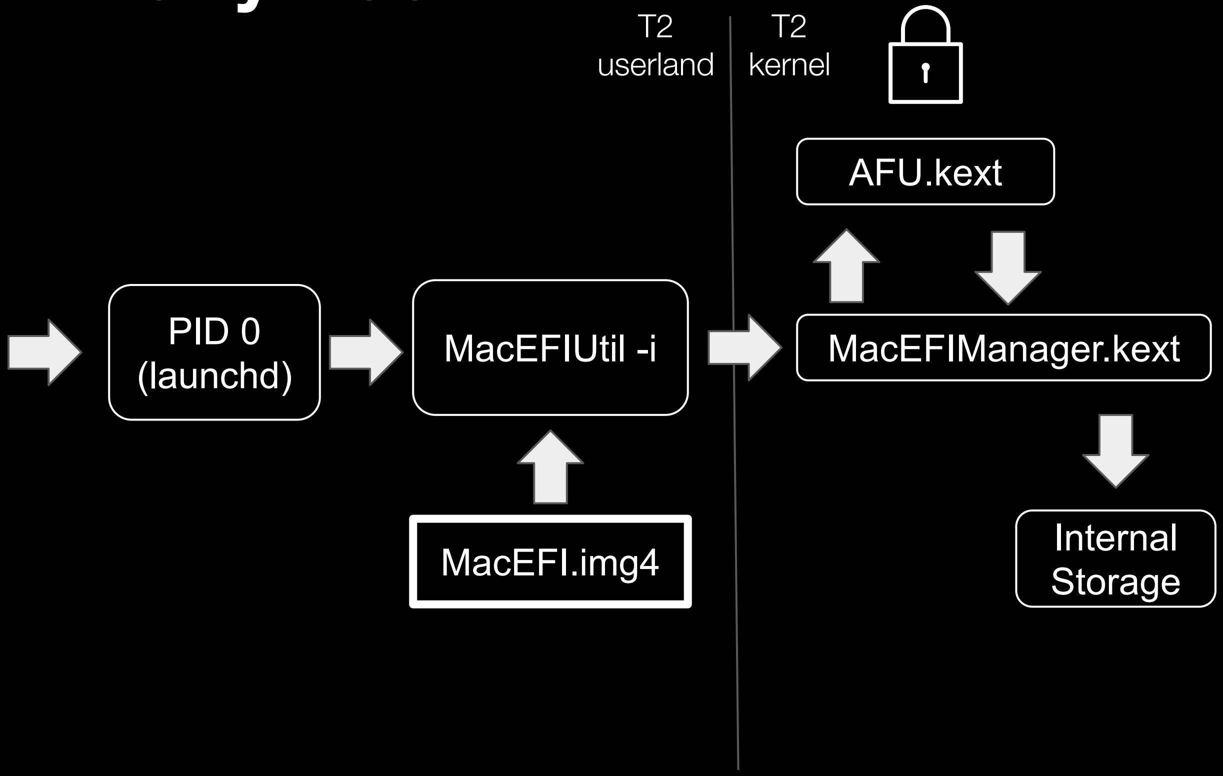


T2 Early Boot

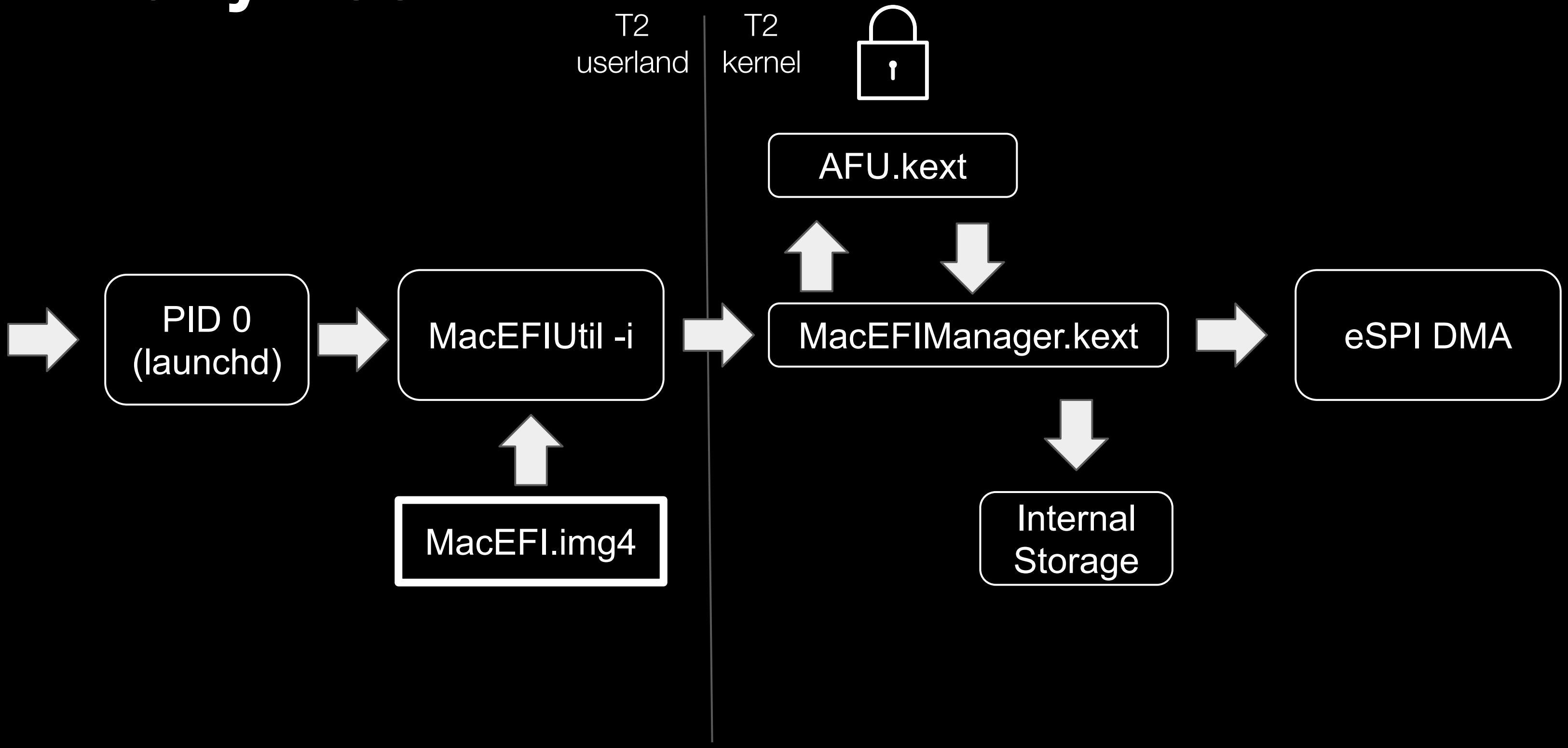


```
// Are we hardware fused to production mode?
if (Fuse_ApProductionStatus)
  isRomLocked = 1;

// Do we have an overriding boot argument?
PE_parse_boot_argn("macefi.locked", &isRomLocked, 1);

if ( isRomLocked )
  lockIndicatorValue = 0x4E4F223198E57BA1LL;
else
  lockIndicatorValue = 0x4E15E2F599858AC6LL;

// Write indicator into the UEFI image.
*(_QWORD *)(ESPIBaseAddress + UEFIPayloadSize - 128) = lockIndicatorValue;
```

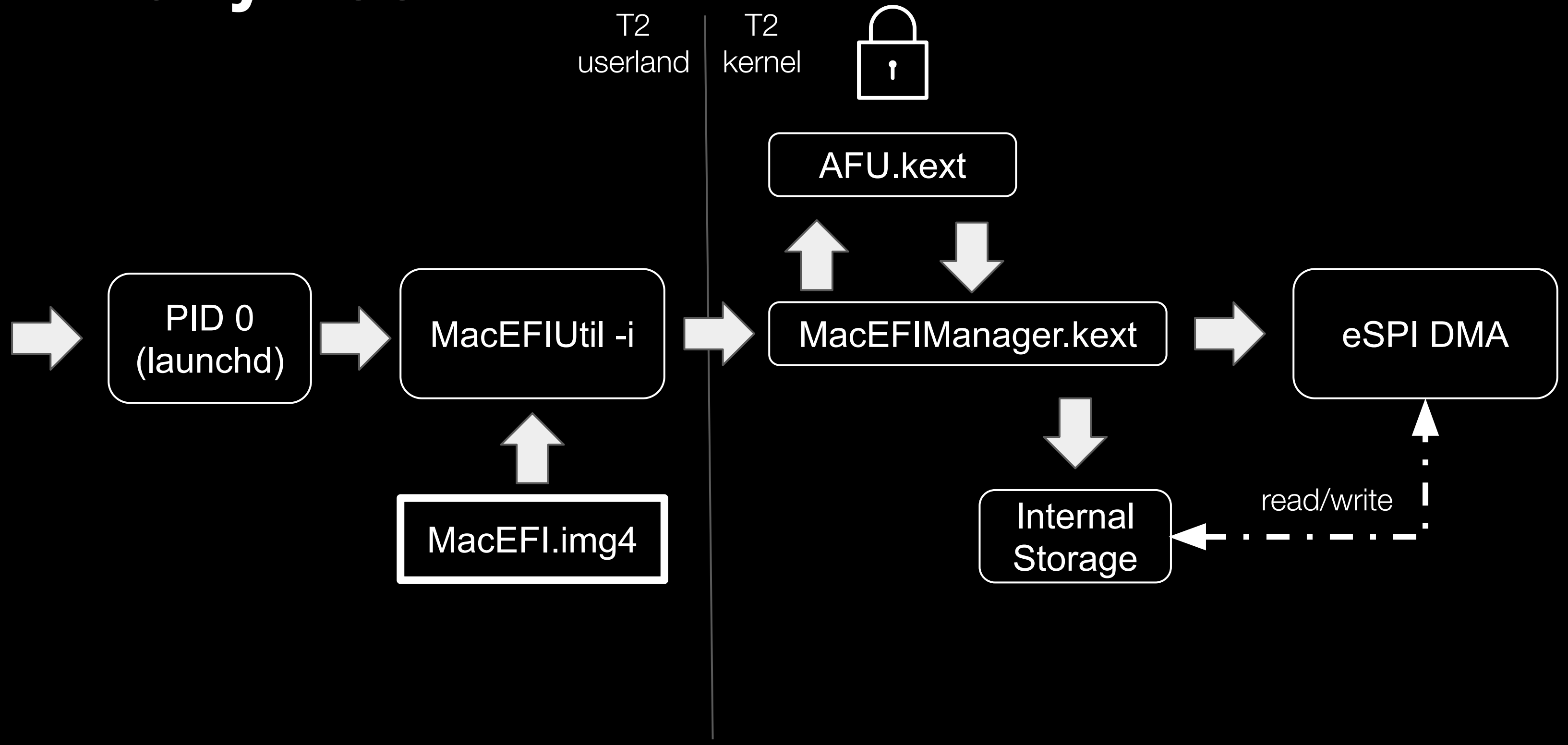
T2 Early Boot

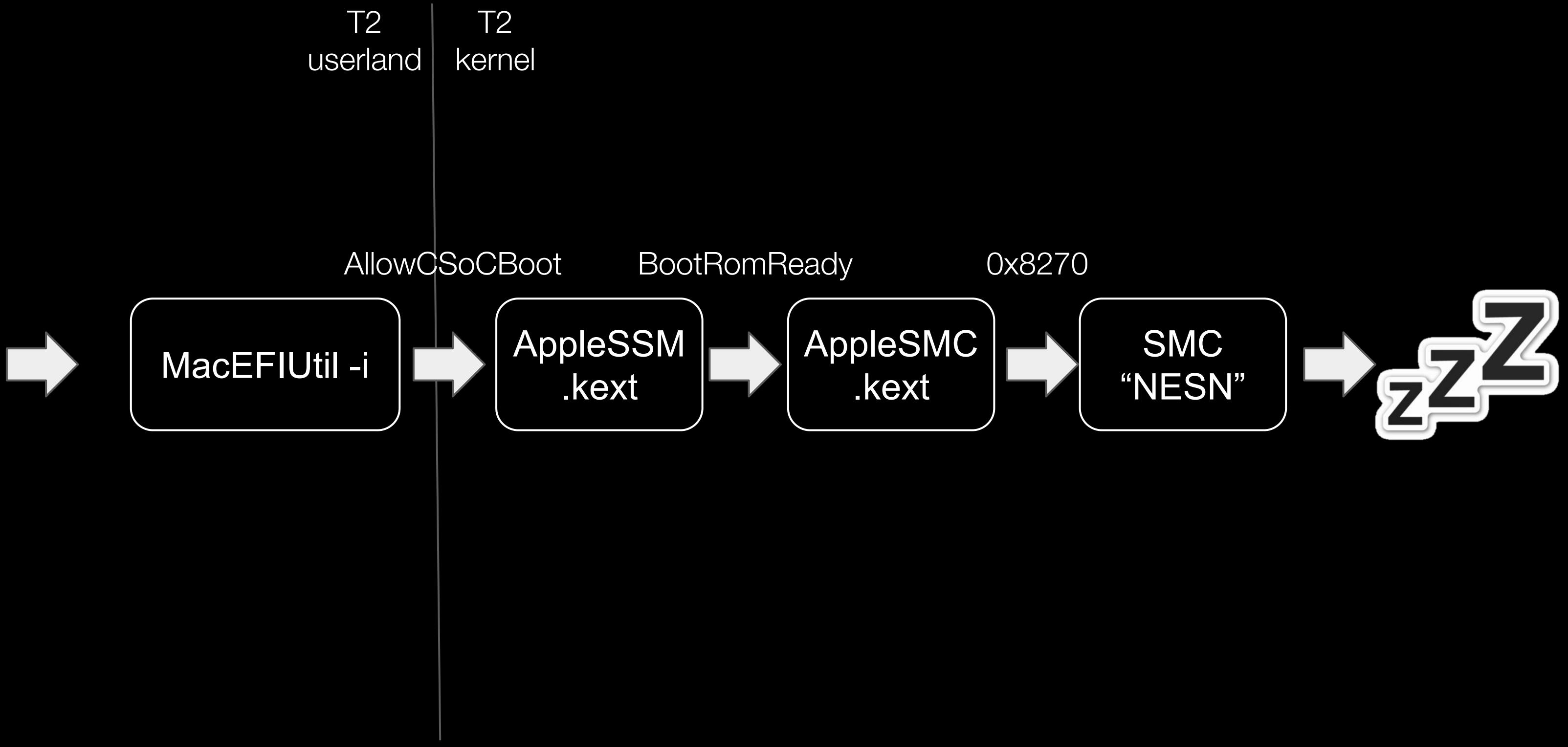


T2 Early Boot

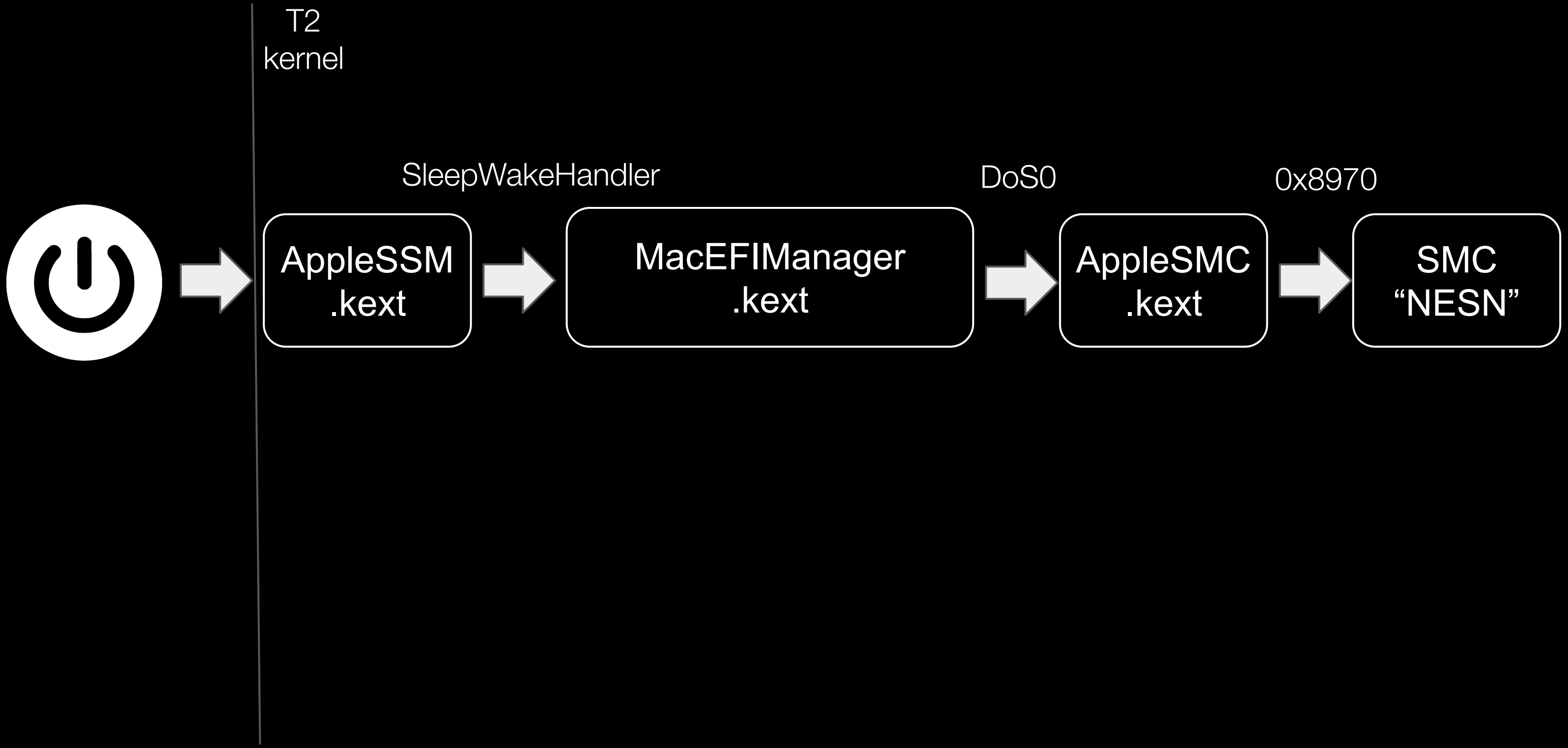


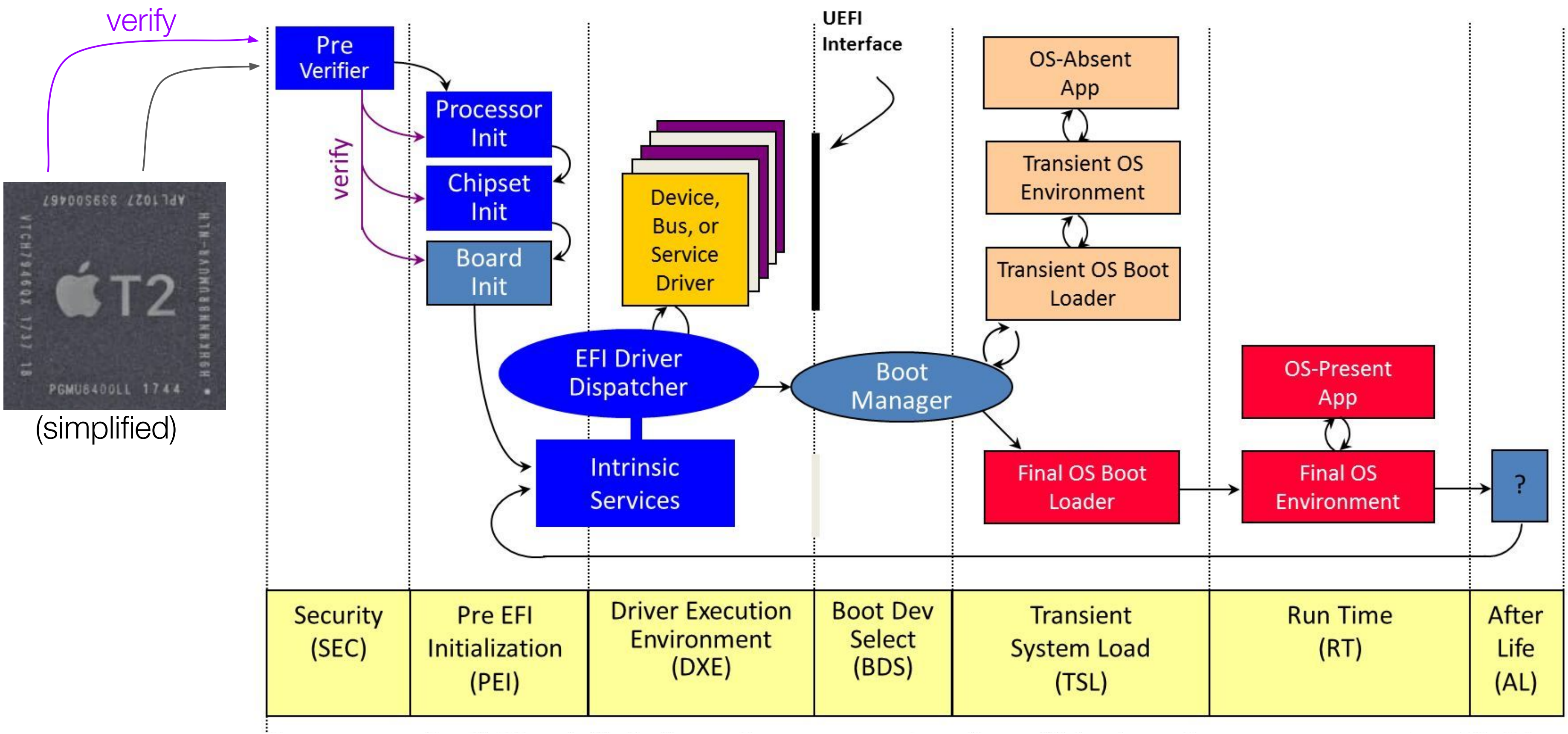
T2 Early Boot




T2
userland
T2
kernel
AllowCSoCBoot
BootRomReady
0x8270
MacEFIUtil -i
AppleSSM
.kext
AppleSMC
.kext
SMC
“NESN”

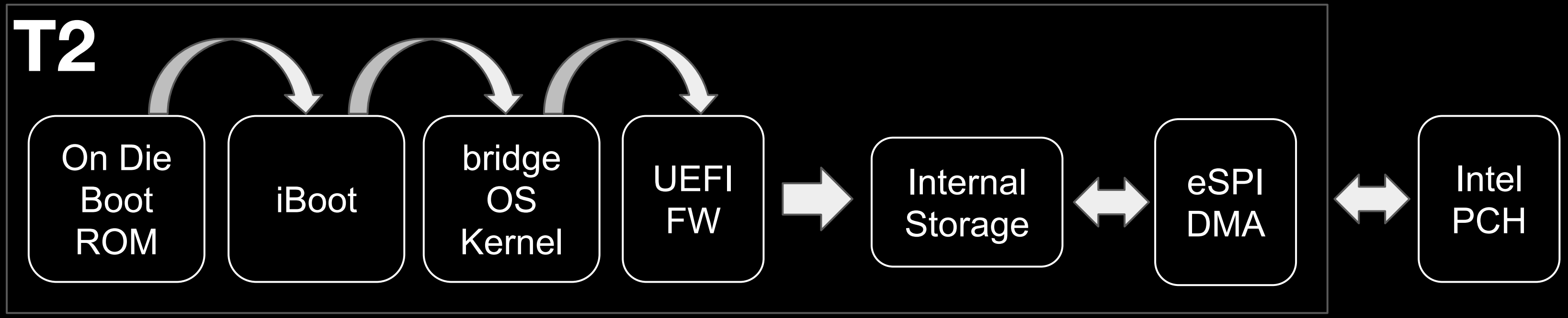
Getting to S0



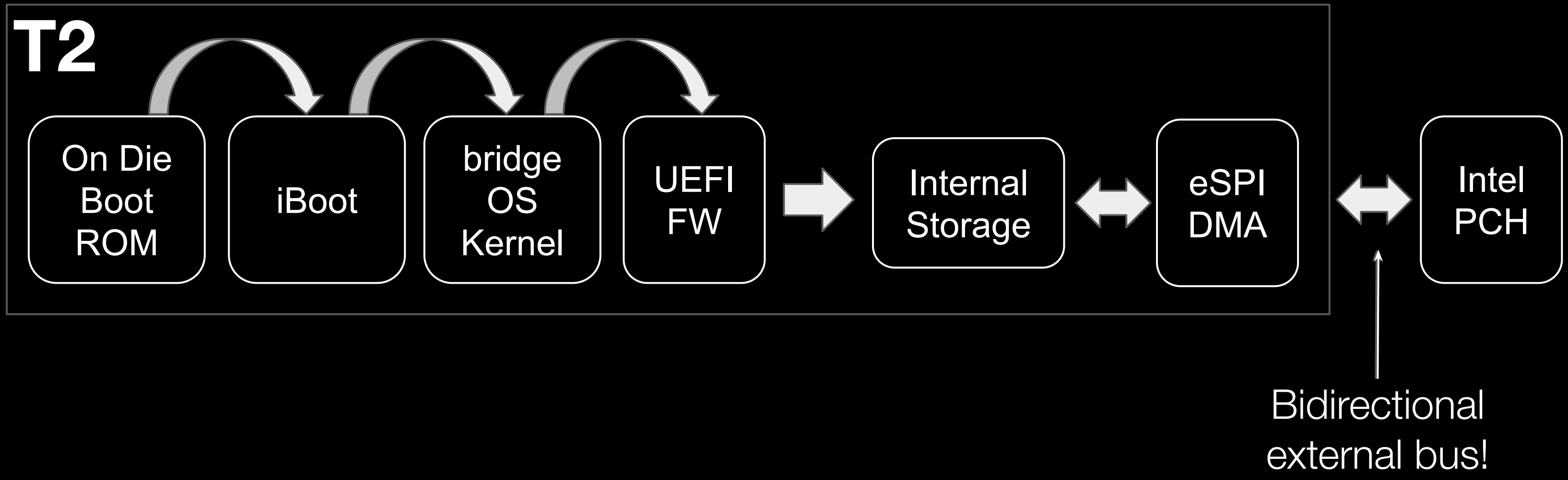


tianocore.github.io/master/images/PI_Boot_Phases.JPG

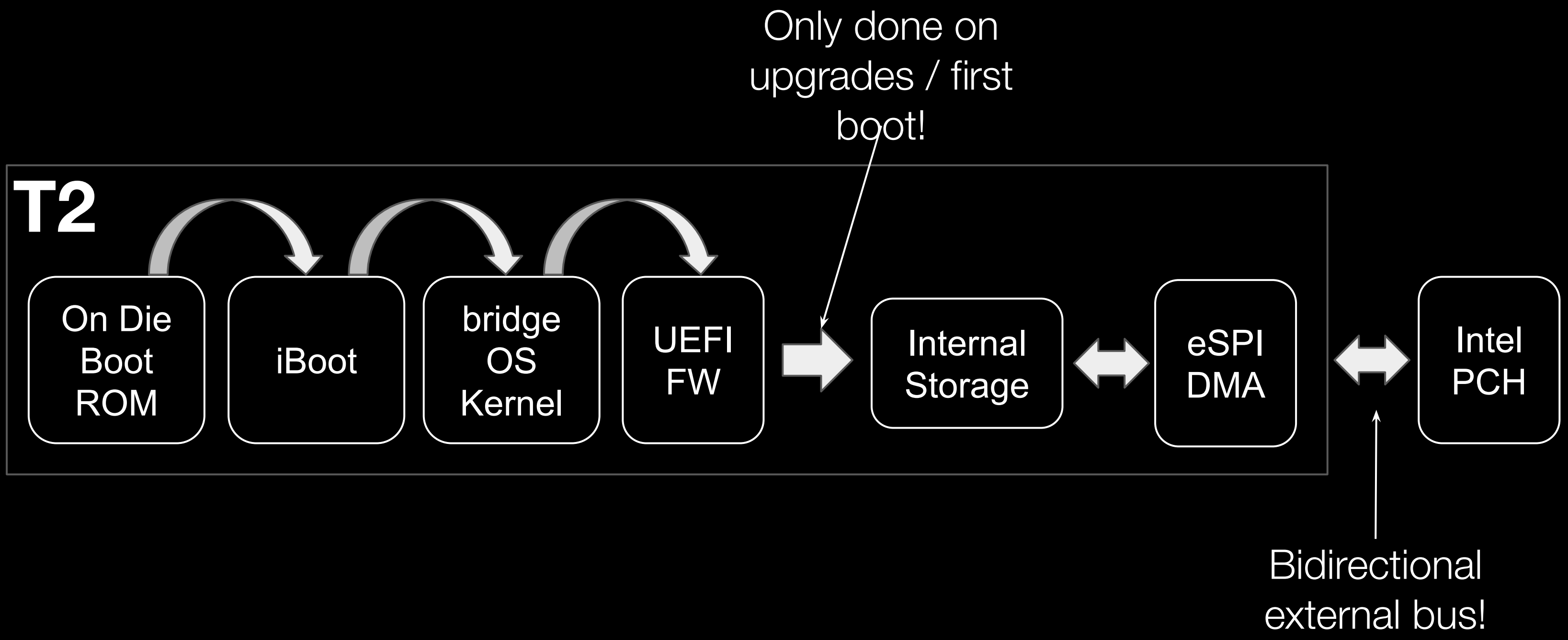
Attacking Secure Boot 💁‍♀️🏴‍☠️



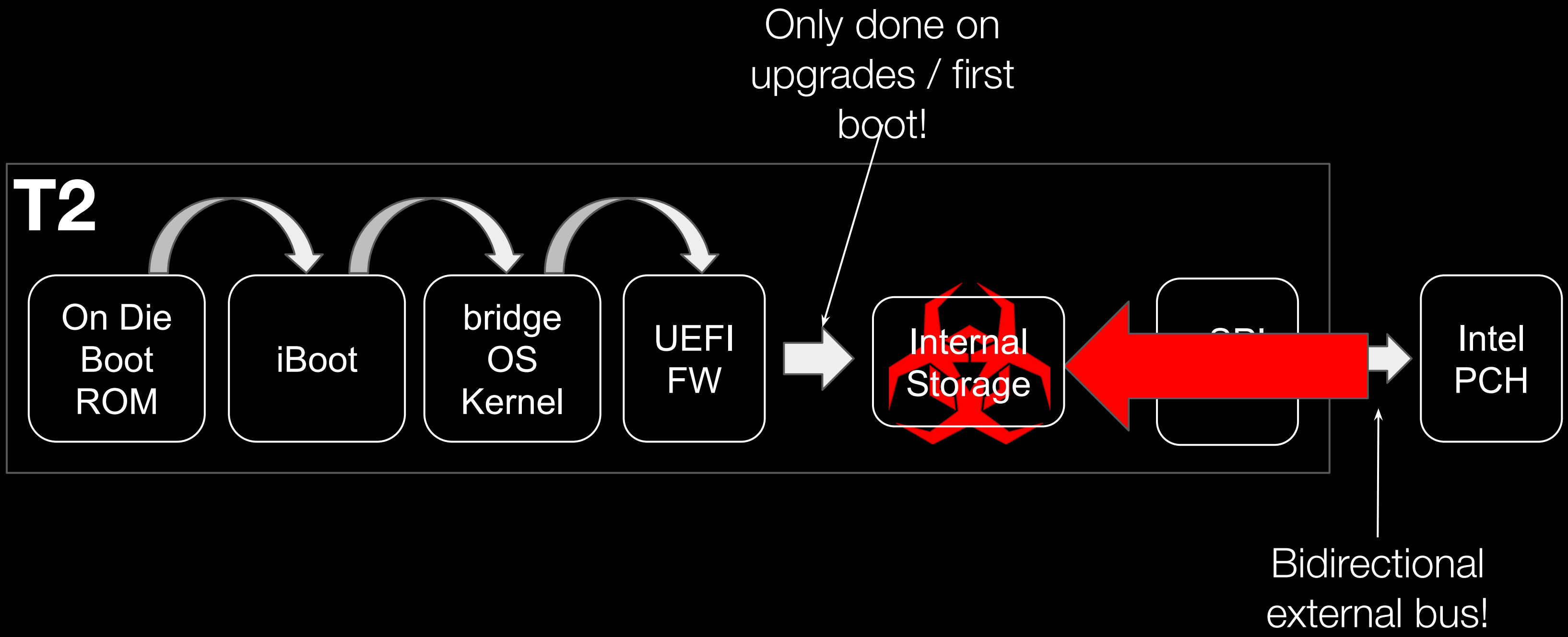
Attacking Secure Boot 



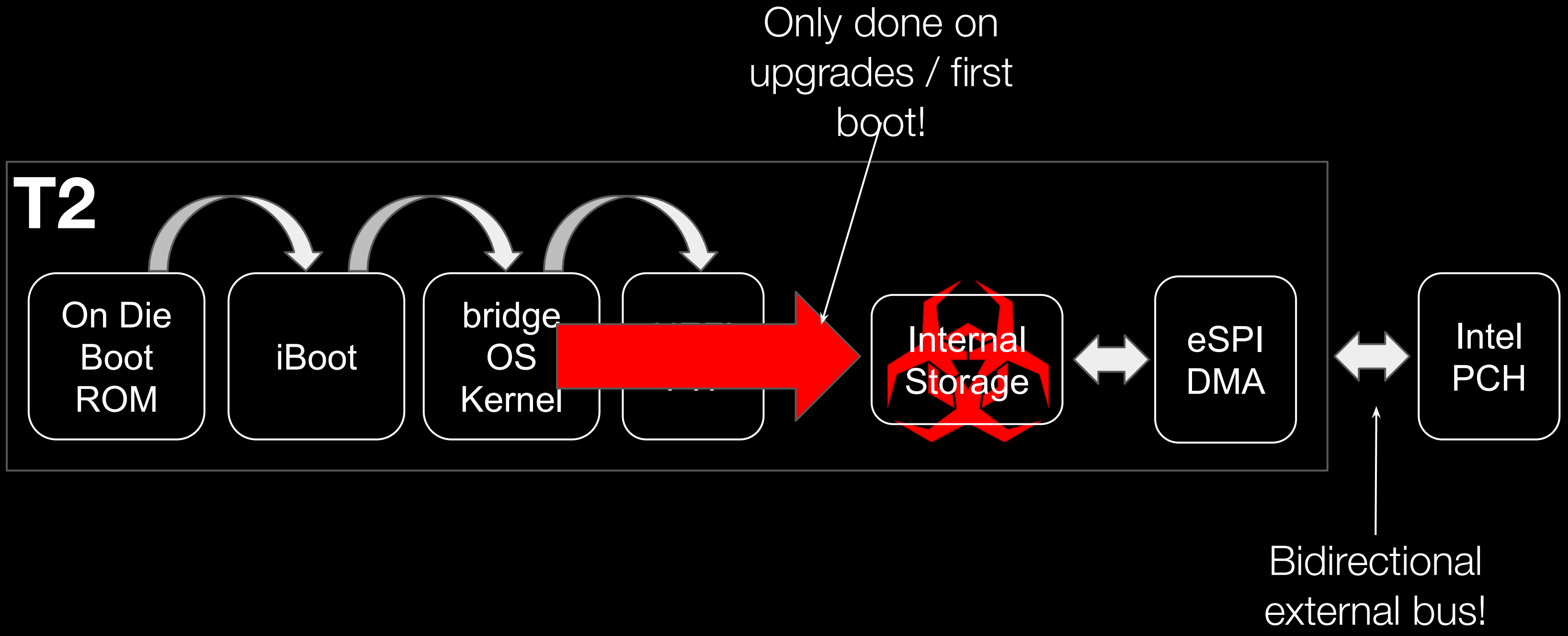
Attacking Secure Boot



Attacking Secure Boot 

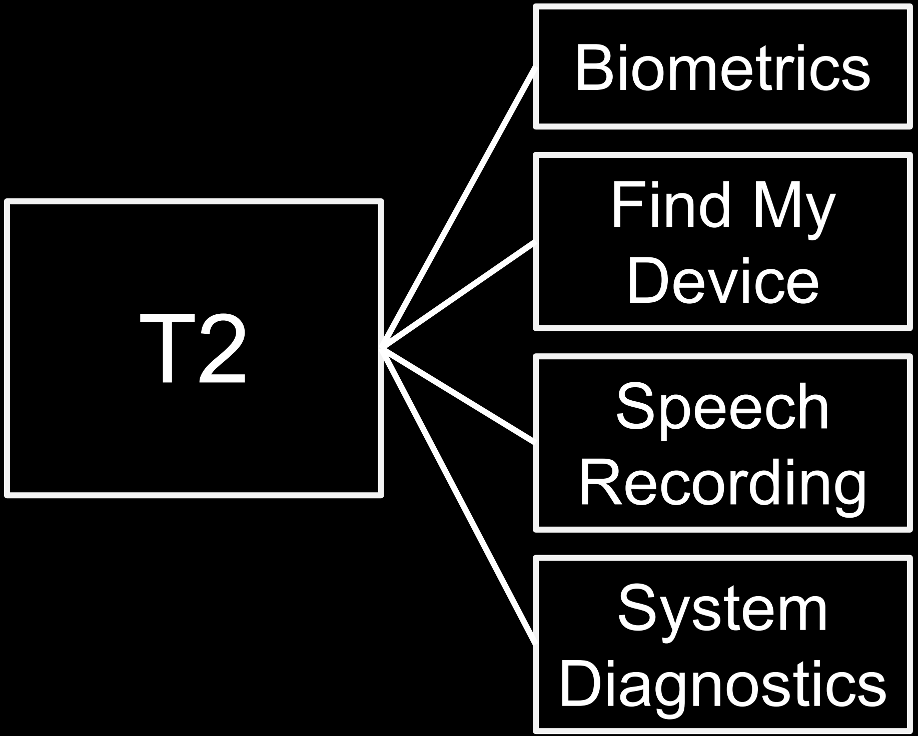


Attacking Secure Boot 



Exposed T2 Services

T2 Services



Once booted, the T2 runs a number of services on behalf of the host OS

Would it be possible to get remote code execution on the T2 via the host?

With a bridgeOS kernel exploit, it might be possible to overwrite the internal flash through software

What interface does the T2 expose to the host OS after boot?

Remotectl

```
$ remotectl
usage: remotectl list
usage: remotectl show (name|uuid)
usage: remotectl get-property ...
usage: remotectl dumpstate
usage: remotectl browse
usage: remotectl echo ...
usage: remotectl eos-echo
usage: remotectl netcat ...
usage: remotectl relay ...
usage: remotectl loopback ...
usage: remotectl convert-bridge-version
usage: remotectl heartbeat ...
usage: remotectl trampoline ...
```

Remotectl

```
$ remotectl
usage: remotectl list
usage: remotectl show (name|uuid)
usage: remotectl get-property ...
usage: remotectl dumpstate
usage: remotectl browse
usage: remotectl echo ...
usage: remotectl eos-echo
usage: remotectl netcat ...
usage: remotectl relay ...
usage: remotectl loopback ...
usage: remotectl convert-bridge-version
usage: remotectl heartbeat ...
usage: remotectl trampoline ...
```

```
$ remotectl list
2AC47A5D-E9EF   localbridge    iBridge ...
```

Remotectl

```
$ remotectl
usage: remotectl list
usage: remotectl show (name|uuid)
usage: remotectl get-property ...
usage: remotectl dumpstate
usage: remotectl browse
usage: remotectl echo ...
usage: remotectl eos-echo
usage: remotectl netcat ...
usage: remotectl relay ...
usage: remotectl loopback ...
usage: remotectl convert-bridge-version
usage: remotectl heartbeat ...
usage: remotectl trampoline ...
```

```
$ remotectl list
2AC47A5D-E9EF   localbridge    iBridge ...

$ remotectl show localbridge
Services:
    com.apple.CSCRemoteSupportd
    com.apple.sysdiagnose.remote
    com.apple.corespeech.xpc.remote.record
    com.apple.xpc.remote.multiboot
    com.apple.eos.LASecureIO
    com.apple.osanalytics.logTransfer
    com.apple.eos.BiometricKit
    com.apple.aveservice
    com.apple.powerchime.remote
    com.apple.bridgeOSUpdated
    com.apple.private.avvc.xpc.remote
    ...
```

Remotectl

```
$ remotectl
usage: remotectl list
usage: remotectl show (name|uuid)
usage: remotectl get-property ...
usage: remotectl dumpstate
usage: remotectl browse
usage: remotectl echo ...
usage: remotectl eos-echo
usage: remotectl netcat ...
usage: remotectl relay ...
usage: remotectl loopback ...
usage: remotectl convert-bridge-version
usage: remotectl heartbeat ...
usage: remotectl trampoline ...
```

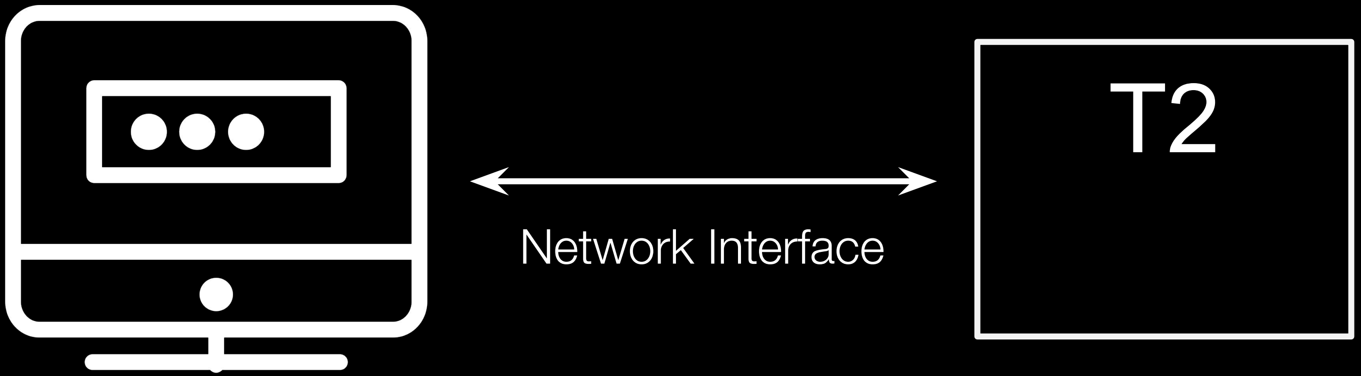
```
$ remotectl list
2AC47A5D-E9EF   localbridge    iBridge ...

$ remotectl show localbridge
Services:
    com.apple.CSCRemoteSupportd
    com.apple.sysdiagnose.remote
    com.apple.corespeech.xpc.remote.record
    com.apple.xpc.remote.multiboot
    com.apple.eos.LASecureIO
    com.apple.osanalytics.logTransfer
    com.apple.eos.BiometricKit
    com.apple.aveservice
    com.apple.powerchime.remote
    com.apple.bridgeOSUpdated
    com.apple.private.avvc.xpc.remote
    ...
```

Communication Channel

RemoteXPC

XPC is Apple’s IPC protocol, implemented by the RemoteXPC library

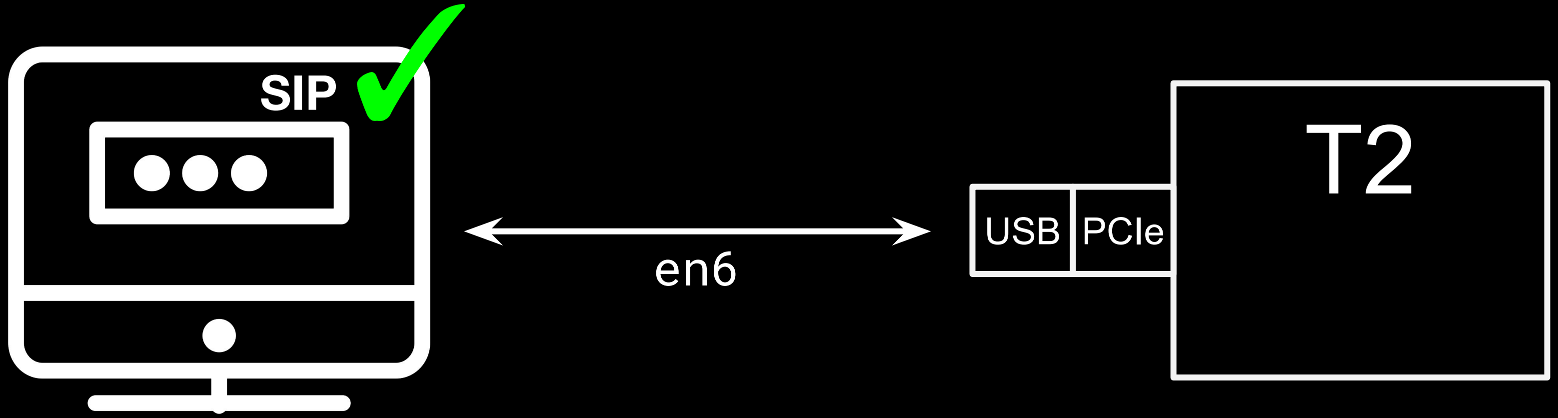
The T2 coprocessor uses RemoteXPC to communicate with the host macOS



Network Interface

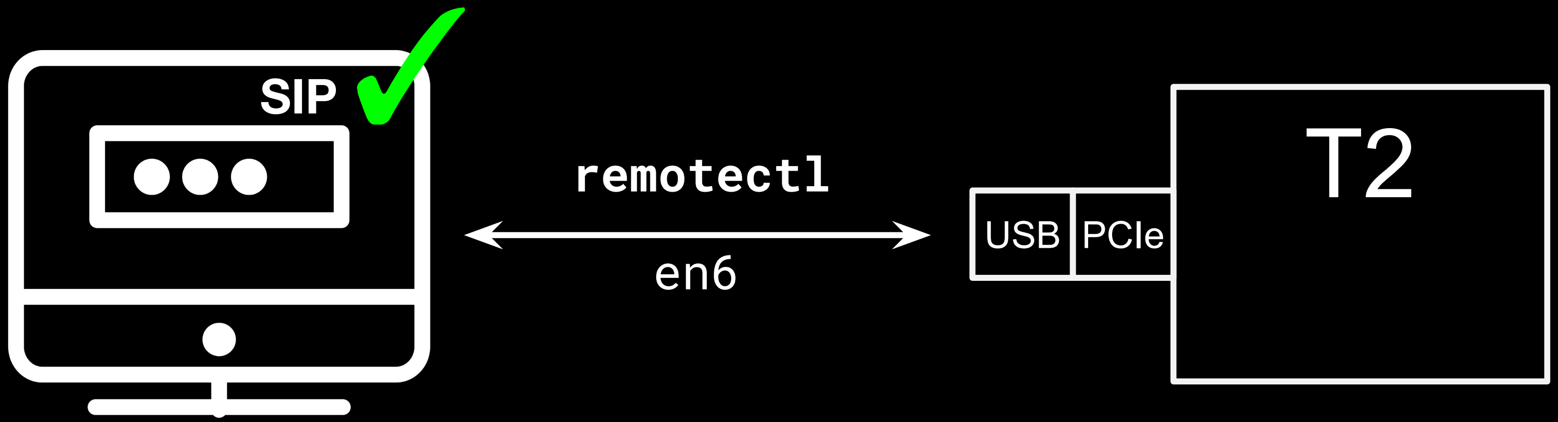
T2 is exposed as **en6**, a usb-attached network interface via the PCIe bus

Protected by SIP



Network Interface

Not necessary to have root or disable SIP to use **`remotectl relay`**

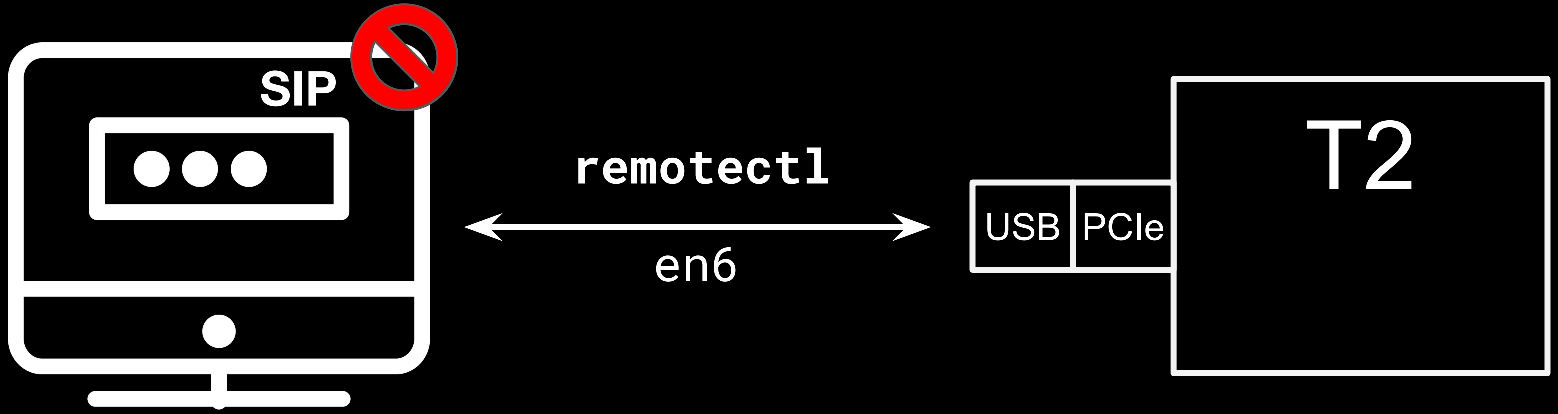


Network Interface

Was

Not necessary to have root or disable SIP to use **`remotectl relay`**

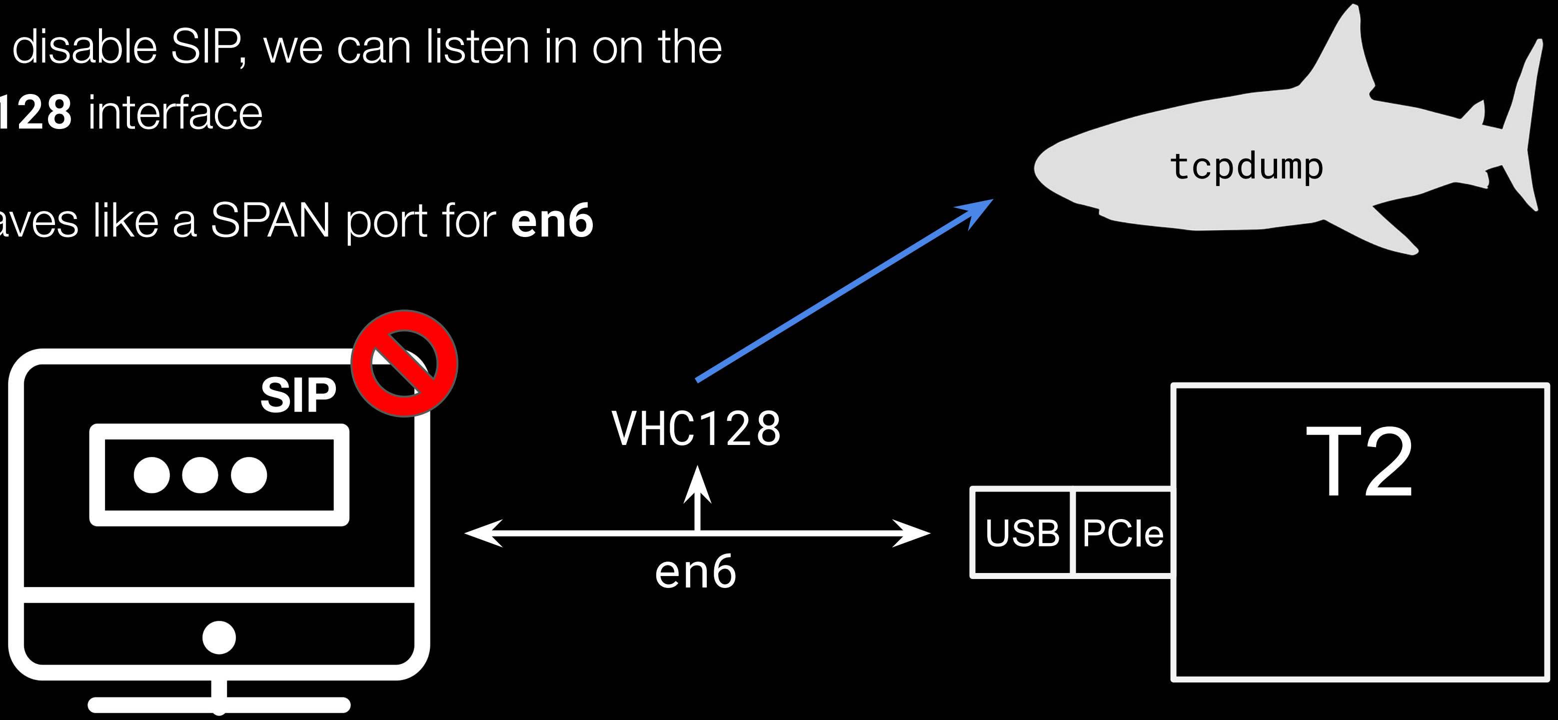
as of 10.14.3, `remotectl` needs a little “help” to work

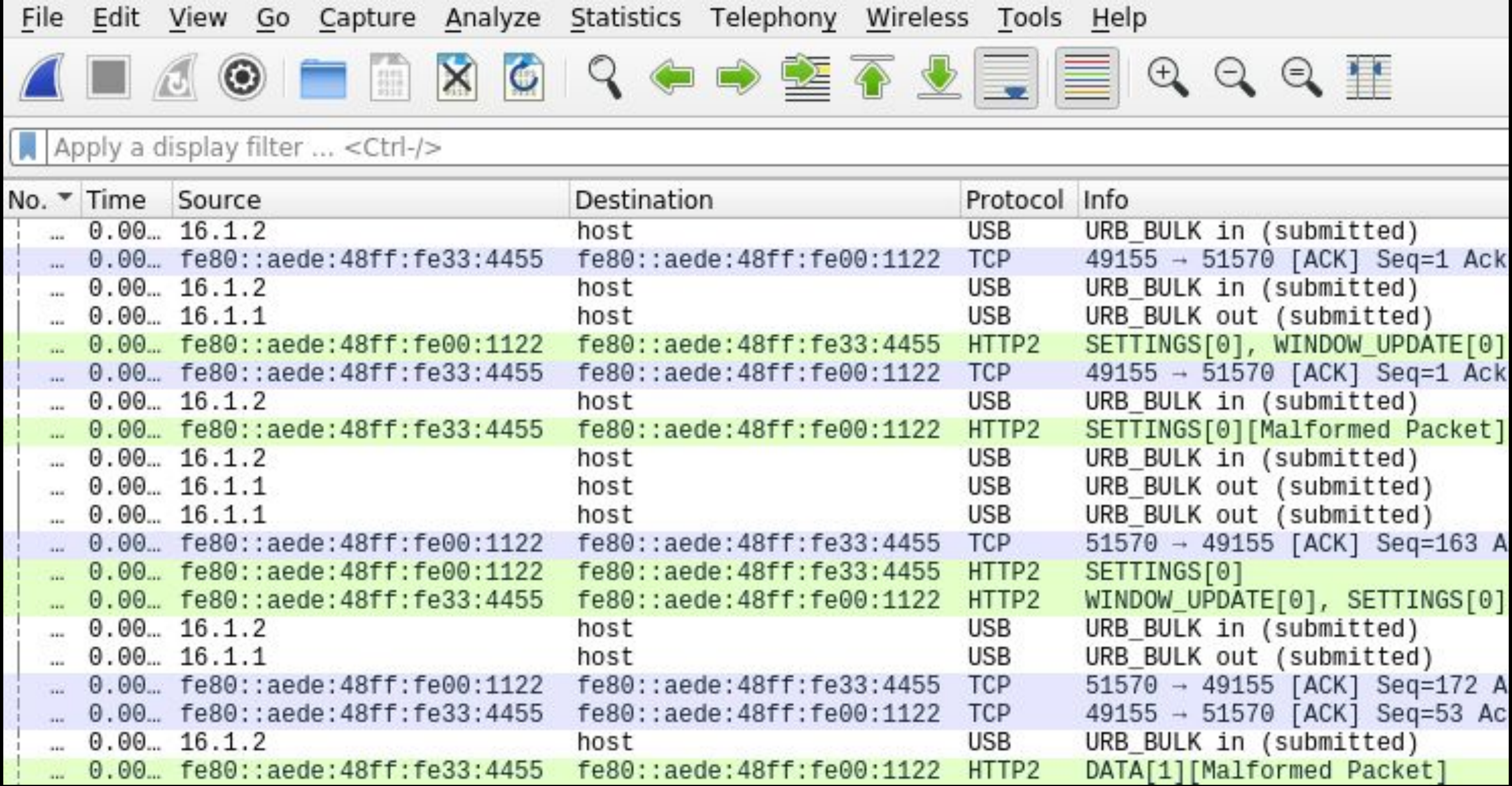


Network Interface

If we disable SIP, we can listen in on the **VHC128** interface

Behaves like a SPAN port for **en6**



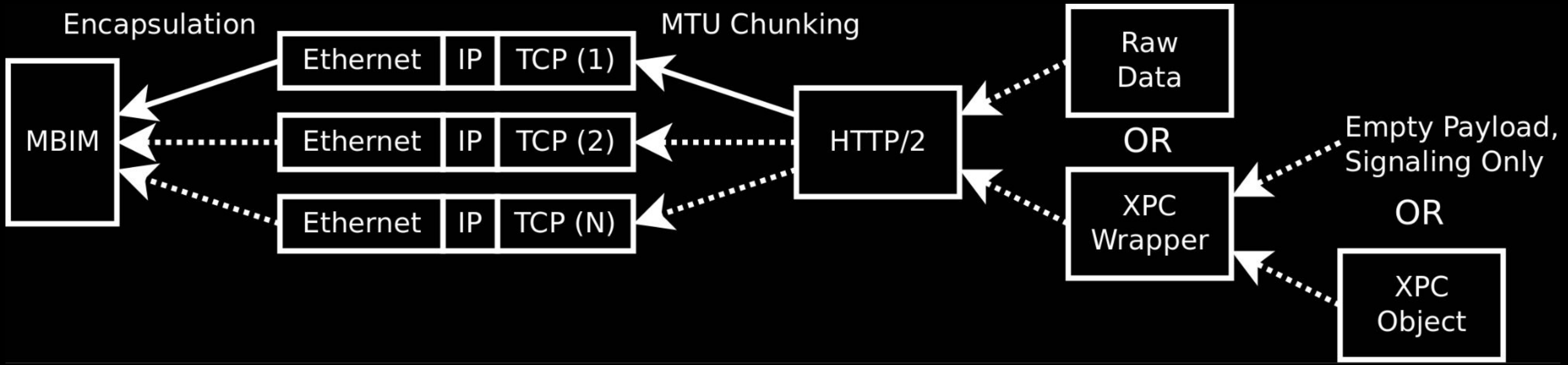

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

No.	Time	Source	Destination	Protocol	Info
…	0.00…	16.1.2	host	USB	URB_BULK in (submitted)
…	0.00…	fe80::aede:48ff:fe33:4455	fe80::aede:48ff:fe00:1122	TCP	49155 → 51570 [ACK] Seq=1 Ack
…	0.00…	16.1.2	host	USB	URB_BULK in (submitted)
…	0.00…	16.1.1	host	USB	URB_BULK out (submitted)
…	0.00…	fe80::aede:48ff:fe00:1122	fe80::aede:48ff:fe33:4455	HTTP2	SETTINGS[0], WINDOW_UPDATE[0]
…	0.00…	fe80::aede:48ff:fe33:4455	fe80::aede:48ff:fe00:1122	TCP	49155 → 51570 [ACK] Seq=1 Ack
…	0.00…	16.1.2	host	USB	URB_BULK in (submitted)
…	0.00…	fe80::aede:48ff:fe33:4455	fe80::aede:48ff:fe00:1122	HTTP2	SETTINGS[0][Malformed Packet]
…	0.00…	16.1.2	host	USB	URB_BULK in (submitted)
…	0.00…	16.1.1	host	USB	URB_BULK out (submitted)
…	0.00…	16.1.1	host	USB	URB_BULK out (submitted)
…	0.00…	fe80::aede:48ff:fe00:1122	fe80::aede:48ff:fe33:4455	TCP	51570 → 49155 [ACK] Seq=163 A
…	0.00…	fe80::aede:48ff:fe00:1122	fe80::aede:48ff:fe33:4455	HTTP2	SETTINGS[0]
…	0.00…	fe80::aede:48ff:fe33:4455	fe80::aede:48ff:fe00:1122	HTTP2	WINDOW_UPDATE[0], SETTINGS[0]
…	0.00…	16.1.2	host	USB	URB_BULK in (submitted)
…	0.00…	16.1.1	host	USB	URB_BULK out (submitted)
…	0.00…	fe80::aede:48ff:fe00:1122	fe80::aede:48ff:fe33:4455	TCP	51570 → 49155 [ACK] Seq=172 A
…	0.00…	fe80::aede:48ff:fe33:4455	fe80::aede:48ff:fe00:1122	TCP	49155 → 51570 [ACK] Seq=53 Ac
…	0.00…	16.1.2	host	USB	URB_BULK in (submitted)
…	0.00…	fe80::aede:48ff:fe33:4455	fe80::aede:48ff:fe00:1122	HTTP2	DATA[1][Malformed Packet]

```
▾ HyperText Transfer Protocol 2
  ▾ Stream: DATA, Stream ID: 1, Length 72 (partial entity body)
      Length: 72
      Type: DATA (0)
    ▸ Flags: 0x00
      0... .... .... .... .... .... .... .... = Reserved: 0x0
      .000 0000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
      [Pad Length: 0]
      Data: 920bb0290101000030000000000000000100000000000000...

0080  80 18 10 04 0c 15 00 00  01 01 08 0a 3d 97 6e ed   ········ ····=·n·
0090  3f a0 26 d9 00 00 48 00  00 00 00 00 01 92 0b b0   ?·&···H· ········
00a0  29 01 01 00 00 30 00 00  00 00 00 00 00 01 00 00   )····0·· ········
00b0  00 00 00 00 00 42 37 13  42 05 00 00 00 00 f0 00   ·····B7· B·······
00c0  00 20 00 00 00 01 00 00  00 52 45 51 55 45 53 54   · ······ ·REQUEST
00d0  5f 54 59 50 45 00 00 00  00 00 40 00 00 01 00 00   _TYPE··· ··@·····
00e0  00 00 00 00 00                                     ·····
```

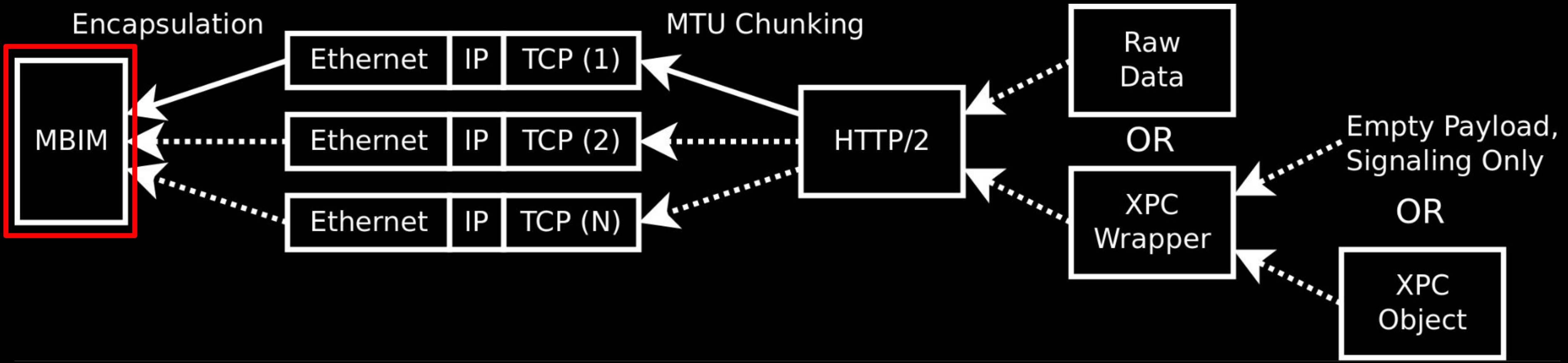
Decoding Message Layers

Layers



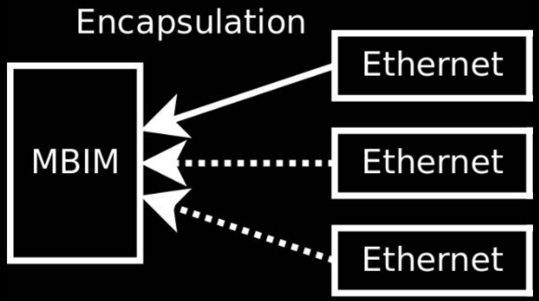
Encapsulation
MTU Chunking
MBIM
Ethernet
IP
TCP (1)
Ethernet
IP
TCP (2)
Ethernet
IP
TCP (N)
HTTP/2
Raw Data
OR
XPC Wrapper
Empty Payload, Signaling Only
OR
XPC Object

Layers


Encapsulation
MBIM
Ethernet
IP
TCP (1)
Ethernet
IP
TCP (2)
Ethernet
IP
TCP (N)
MTU Chunking
HTTP/2
Raw
Data
OR
XPC
Wrapper
Empty Payload,
Signaling Only
OR
XPC
Object

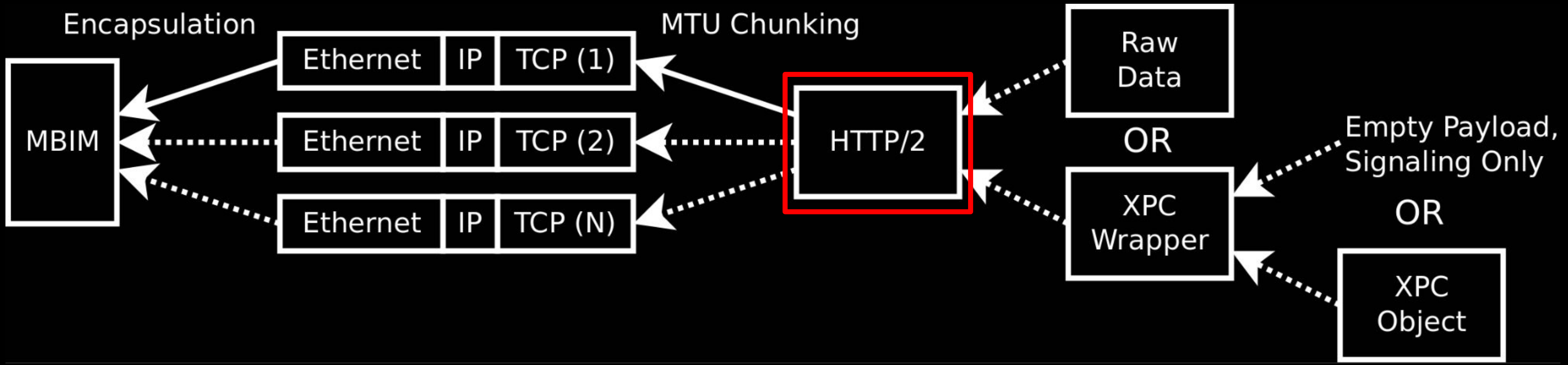
MBIM (USB)

Encapsulates one or more Ethernet frames for transit over USB-based interface




```
Frame 61: 10700 bytes on wire (85600 bits), 10700 bytes captured (85600 bits)
USB URB
Mobile Broadband Interface Model
  NCM Transfer Header
  NCM Datagram Pointer
  [Total Number Of Datagrams: 7]
Ethernet II, Src: Private_33:44:55 (ac:de:48:33:44:55), Dst: Private_00:11:22 (ac:de:48:00:11:22)
Internet Protocol Version 6, Src: fe80::aede:48ff:fe33:4455, Dst: fe80::aede:48ff:fe00:1122
Transmission Control Protocol, Src Port: 49164, Dst Port: 49154, Seq: 376, Ack: 43, Len: 1428
Ethernet II, Src: Private_33:44:55 (ac:de:48:33:44:55), Dst: Private_00:11:22 (ac:de:48:00:11:22)
Internet Protocol Version 6, Src: fe80::aede:48ff:fe33:4455, Dst: fe80::aede:48ff:fe00:1122
Transmission Control Protocol, Src Port: 49164, Dst Port: 49154, Seq: 1804, Ack: 43, Len: 1428
Ethernet II, Src: Private_33:44:55 (ac:de:48:33:44:55), Dst: Private_00:11:22 (ac:de:48:00:11:22)
Internet Protocol Version 6, Src: fe80::aede:48ff:fe33:4455, Dst: fe80::aede:48ff:fe00:1122
Transmission Control Protocol, Src Port: 49164, Dst Port: 49154, Seq: 3232, Ack: 43, Len: 1428
```

Layers


Encapsulation
MTU Chunking
MBIM
Ethernet
IP
TCP (1)
Ethernet
IP
TCP (2)
Ethernet
IP
TCP (N)
HTTP/2
Raw
Data
OR
XPC
Wrapper
Empty Payload,
Signaling Only
OR
XPC
Object

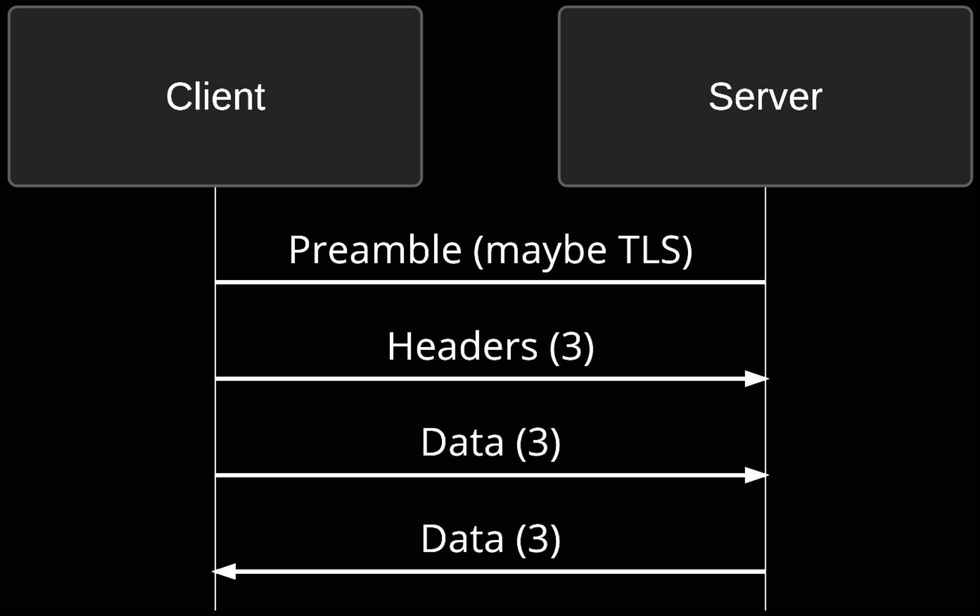
HTTP/2 Crash Course

One connection, multiple *streams*

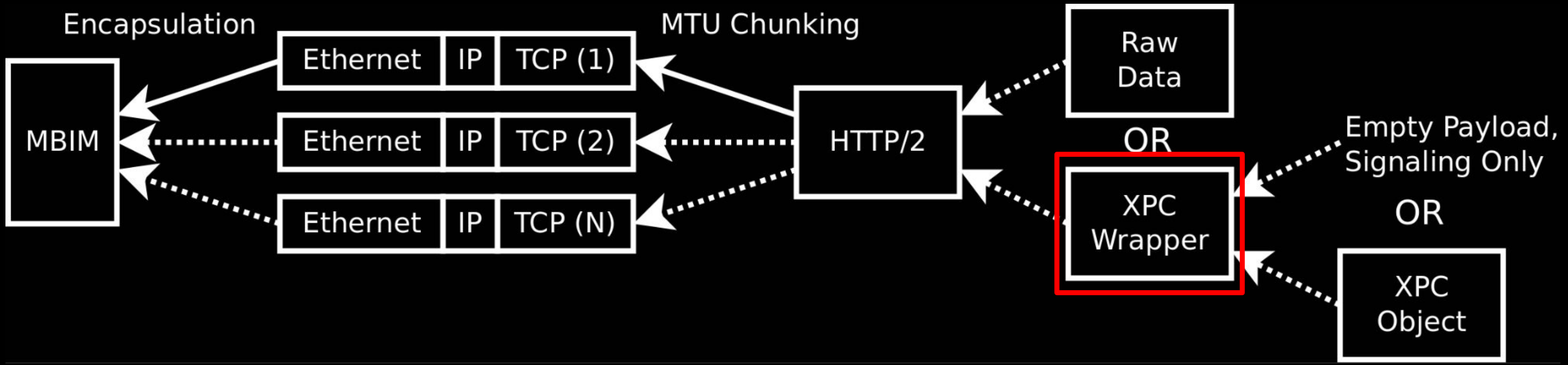
Streams are opened with a `HEADERS` frame

Once opened, `DATA` frames can be sent bidirectionally

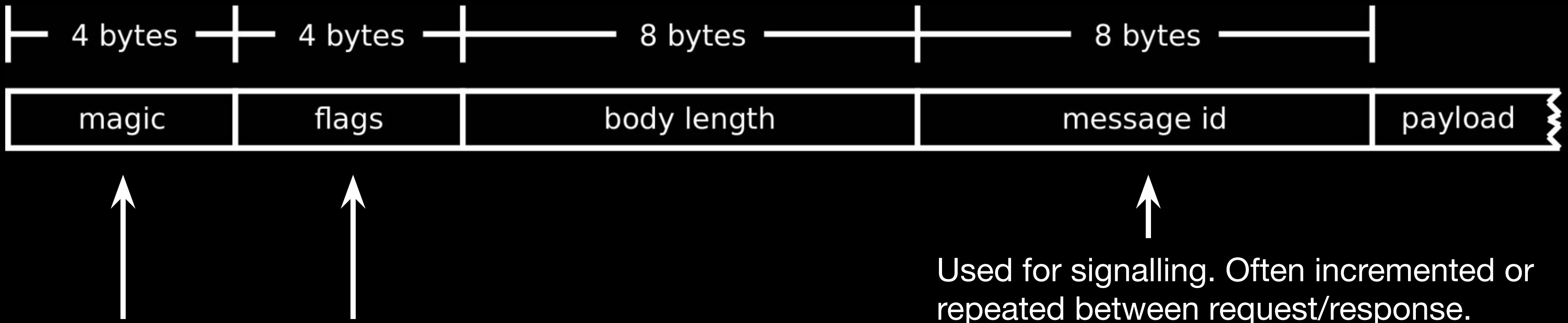
Apple uses this in a *non-standard* way as an encapsulation layer for XPC messaging



Layers


Encapsulation
MTU Chunking
MBIM
Ethernet
IP
TCP (1)
Ethernet
IP
TCP (2)
Ethernet
IP
TCP (N)
HTTP/2
Raw Data
OR
XPC Wrapper
Empty Payload, Signaling Only
OR
XPC Object

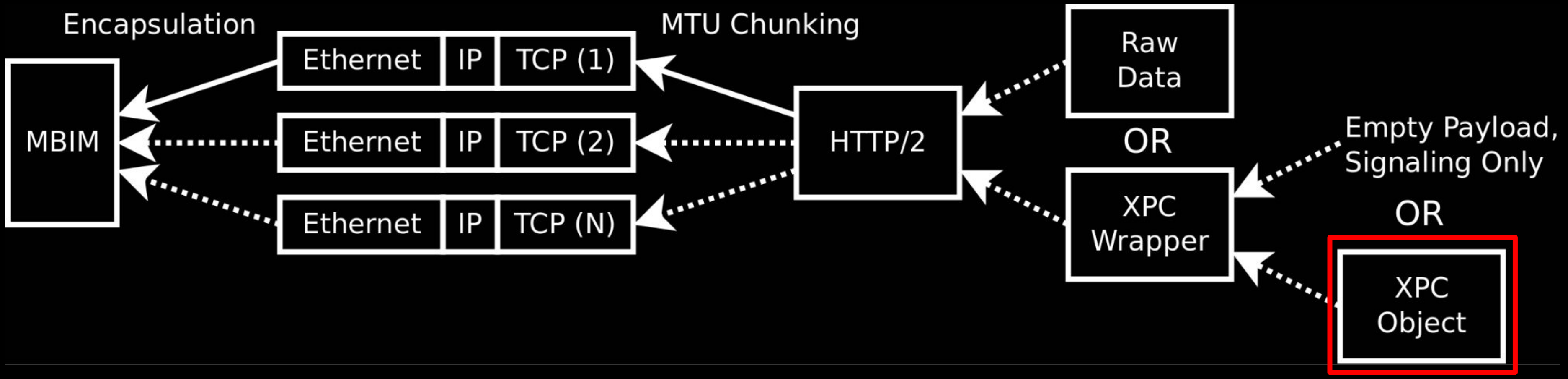
XPC Wrapper



Used for signalling. Often incremented or repeated between request/response.

```
0x29B00B92   Flag bits:
             00000000 00000000 00000000 00000001 - Always set
             00000000 00000000 00000001 00000000 - Data present
             00000000 00000001 00000000 00000000 - Heartbeat request
             00000000 00000010 00000000 00000000 - Heartbeat reply
             00000000 00010000 00000000 00000000 - Opening a new file_tx stream
             00000000 00100000 00000000 00000000 - Reply from file_tx stream
             00000000 01000000 00000000 00000000 - Sysdiagnose init handshake
```

Layers



Encapsulation
MTU Chunking
MBIM
Ethernet
IP
TCP (1)
Ethernet
IP
TCP (2)
Ethernet
IP
TCP (N)
HTTP/2
Raw Data
OR
XPC Wrapper
Empty Payload, Signaling Only
OR
XPC Object

Decoding XPC

Overview of XPC

```
xpc_connection_t conn = xpc_connection_create(...);

xpc_object_t message = xpc_dictionary_create(NULL, NULL, 0);

...

xpc_connection_send_message(conn, message);
```

Overview of XPC

```
xpc_connection_t conn = xpc_connection_create(...);

xpc_object_t message = xpc_dictionary_create(NULL, NULL, 0);

xpc_dictionary_set_bool(message, "bool", true);

xpc_dictionary_set_int64(message, "int64", -1);

xpc_dictionary_set_uint64(message, "uint64", 0xdeadbeef);

xpc_connection_send_message(conn, message);
```

Overview of XPC

```
xpc_connection_t conn = xpc_connection_create(...);
xpc_object_t message = xpc_dictionary_create(NULL, NULL, 0);
xpc_dictionary_set_bool(message, "bool", true);
xpc_dictionary_set_int64(message, "int64", -1);
xpc_dictionary_set_uint64(message, "uint64", 0xdeadbeef);
xpc_connection_send_message(conn, message);
```

Overview of XPC

```
xpc_connection_t conn = xpc_connection_create(...);

xpc_object_t message = xpc_dictionary_create(NULL, NULL, 0);

xpc_dictionary_set_bool(message, "bool", true);

xpc_dictionary_set_int64(message, "int64", -1);

xpc_dictionary_set_uint64(message, "uint64", 0xdeadbeef);

xpc_connection_send_message(conn, message);
```

lldb

```
(lldb) x -c 0x120 0x0000000103800fbc
0x103800fbc: 43 50 58 40 05 00 00 00 00 f0 00 00 08 01 00 00  CPX@............
0x103800fcc: 0b 00 00 00 66 64 00 00 00 b0 00 00 63 6f 6e 6e  ....fd......conn
0x103800fdc: 65 63 74 69 6f 6e 00 00 00 20 01 00 73 74 72 69  ection... ..stri
0x103800fec: 6e 67 00 00 00 90 00 00 0b 00 00 00 74 65 73 74  ng..........test
0x103800ffc: 73 74 72 69 6e 67 00 00 64 6f 75 62 6c 65 00 00  string..double..
0x10380100c: 00 50 00 00 cd cc cc cc fc ff ef 40 64 61 74 61  .P.........@data
0x10380101c: 00 00 00 00 00 80 00 00 0a 00 00 00 74 68 69 73  ............this
0x10380102c: 69 73 64 61 74 61 00 00 75 69 6e 74 36 34 00 00  isdata..uint64..
0x10380103c: 00 40 00 00 ef be ad de 00 00 00 00 62 6f 6f 6c  .@..........bool
0x10380104c: 00 00 00 00 00 20 00 00 01 00 00 00 76 61 6c 75  ..... ......valu
0x10380105c: 65 00 00 00 00 f0 00 00 28 00 00 00 01 00 00 00  e.......(.......
0x10380106c: 73 74 72 69 6e 67 5f 69 6e 5f 76 61 6c 75 65 00  string_in_value.
0x10380107c: 00 90 00 00 0c 00 00 00 76 61 6c 75 65 73 74 72  ........valuestr
0x10380108c: 69 6e 67 00 69 6e 74 36 34 00 00 00 00 30 00 00  ing.int64....0..
0x10380109c: ff ff ff ff ff ff ff ff 75 75 69 64 00 00 00 00  ........uuid....
0x1038010ac: 00 a0 00 00 31 32 33 34 35 36 37 38 2d 61 62 63  ....12345678-abc
0x1038010bc: 64 2d 31 32 64 61 74 65 00 00 00 00 00 70 00 00  d-12date.....p..
0x1038010cc: 00 18 9c 46 ae 9e 5c 15 00 00 00 00 00 00 00 00  ...F..\.........
```

XPC Header

XPC Types

XPC objects are always prefixed with a 4-byte **`type`** field

```
Types:
XPC_NULL                      = 0x00001000          XPC_ARRAY                     = 0x0000e000
XPC_BOOL                      = 0x00002000          XPC_DICTIONARY                = 0x0000f000
XPC_INT64                     = 0x00003000          XPC_ERROR                     = 0x00010000
XPC_UINT64                    = 0x00004000          XPC_CONNECTION                = 0x00011000
XPC_DOUBLE                    = 0x00005000          XPC_ENDPOINT                  = 0x00012000
XPC_POINTER                   = 0x00006000          XPC_SERIALIZER                = 0x00013000
XPC_DATE                      = 0x00007000          XPC_PIPE                      = 0x00014000
XPC_DATA                      = 0x00008000          XPC_MACH_RECV                 = 0x00015000
XPC_STRING                    = 0x00009000          XPC_BUNDLE                    = 0x00016000
XPC_UUID                      = 0x0000a000          XPC_SERVICE                   = 0x00017000
XPC_FD                        = 0x0000b000          XPC_SERVICE_INSTANCE          = 0x00018000
XPC_SHMEM                     = 0x0000c000          XPC_ACTIVITY                  = 0x00019000
XPC_MACH_SEND                 = 0x0000d000          XPC_FILE_TRANSFER             = 0x0001a000
```

XPC Types

XPC objects are always prefixed with a 4-byte **`type`** field

```
Types:
XPC_NULL          = 0x00001000     XPC_ARRAY                = 0x0000e000
XPC_BOOL          = 0x00002000     XPC_DICTIONARY           = 0x0000f000
XPC_INT64         = 0x00003000     XPC_ERROR                = 0x00010000
XPC_UINT64        = 0x00004000     XPC_CONNECTION           = 0x00011000
XPC_DOUBLE        = 0x00005000     XPC_ENDPOINT             = 0x00012000
XPC_POINTER       = 0x00006000     XPC_SERIALIZER           = 0x00013000
XPC_DATE          = 0x00007000     XPC_PIPE                 = 0x00014000
XPC_DATA          = 0x00008000     XPC_MACH_RECV            = 0x00015000
XPC_STRING        = 0x00009000     XPC_BUNDLE               = 0x00016000
XPC_UUID          = 0x0000a000     XPC_SERVICE              = 0x00017000
XPC_FD            = 0x0000b000     XPC_SERVICE_INSTANCE     = 0x00018000
XPC_SHMEM         = 0x0000c000     XPC_ACTIVITY             = 0x00019000
XPC_MACH_SEND     = 0x0000d000     XPC_FILE_TRANSFER        = 0x0001a000
```

XPC Fixed-size objects: `uint64`

4-byte type	known-length value

```
00 40 00 00 05 00 00 00 00 00 00 00
|___type__| |________value________|

   uint64              5
```

XPC Variable-length Objects: `string`

4-byte type	4-byte length	N-byte value

```
00 90 00 00 09 00 00 00 64 75 6f 6c 61 62 73 21 00 00 00 00
|___type__| |__length_| |d__u__o__l__a__b__s__!_\0_padding|
```

`string` **`9`** **`duolabs!\0`**

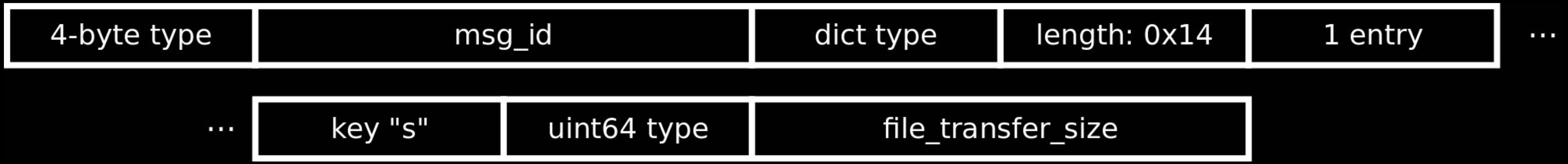
XPC Compound Objects: dictionary

4-byte type	length	num_entries	variable-len key	xpc_object	variable-len key	xpc_object

```
00 f0 00 00 28 00 00 00 02 00 00 00
|___type__| |__length_| |num_entry|
 dictionary      40             2
66 69 76 65 00 00 00 00 00 40 00 00 05 00 00 00 00 00 00 00
|f__i__v__e_\0_padding| |___type__| |________value________|
        “five”             uint64                5
73 69 78 00 00 40 00 00 06 00 00 00 00 00 00 00
|s__i_x_\0| |___type__| |________value________|
   “six”       uint64               6
```

{“five”: 5, “six”: 6}

Other XPC Objects: `file_transfer`



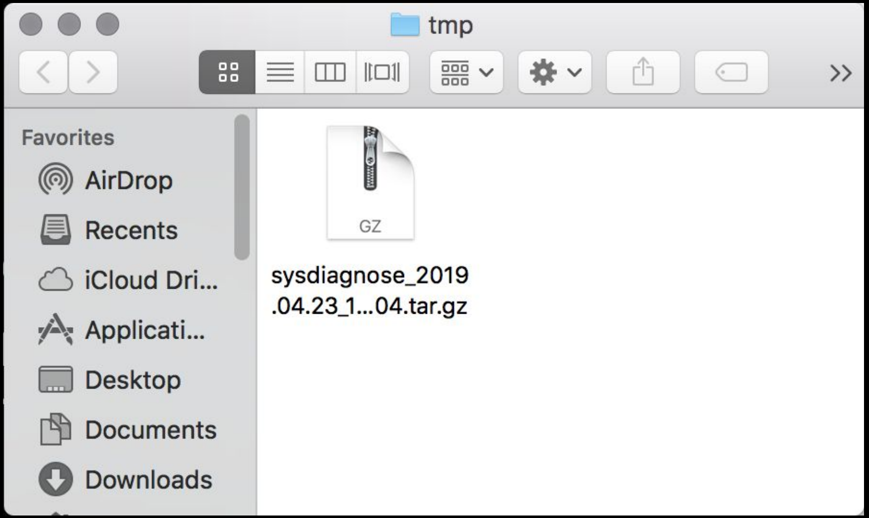
Other objects, such as the `file_transfer` object, may have more complex formats

Please refer to our whitepaper for more details

Listening in on T2 Services

Case Study: Sysdiagnose

System diagnostic reporting tool



`-c` flag retrieves diagnostic information from T2 chip

We can monitor the communications on the `VHC128` interface

Case Study: Sysdiagnose

```
$ sysdiagnose -c &
$ tcpdump -nni VHC128 -w dump.pcap
$ wireshark dump.pcap
```

```
▾ HyperText Transfer Protocol 2
  ▾ Stream: DATA, Stream ID: 1, Length 72 (partial entity body)
      Length: 72
      Type: DATA (0)
    ▸ Flags: 0x00
      0... .... .... .... .... .... .... .... = Reserved: 0x0
      .000 0000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
      [Pad Length: 0]
      Data: 920bb0290101000030000000000000000001000000000000000...

0080  80 18 10 04 0c 15 00 00  01 01 08 0a 3d 97 6e ed   ········ ····=·n·
0090  3f a0 26 d9 00 00 48 00  00 00 00 00 01 92 0b b0   ?·&···H· ········
00a0  29 01 01 00 00 30 00 00  00 00 00 00 00 01 00 00   )····0·· ········
00b0  00 00 00 00 00 42 37 13  42 05 00 00 00 00 f0 00   ·····B7· B·······
00c0  00 20 00 00 00 01 00 00  00 52 45 51 55 45 53 54   · ······ ·REQUEST
00d0  5f 54 59 50 45 00 00 00  00 00 40 00 00 01 00 00   _TYPE··· ··@·····
00e0  00 00 00 00 00                                     ·····
```

Case Study: Sysdiagnose

```
$ sysdiagnose -c &
```

~~$ tcpdump -nni VHC128 -w dump.pcap~~

~~$ wireshark dump.pcap~~

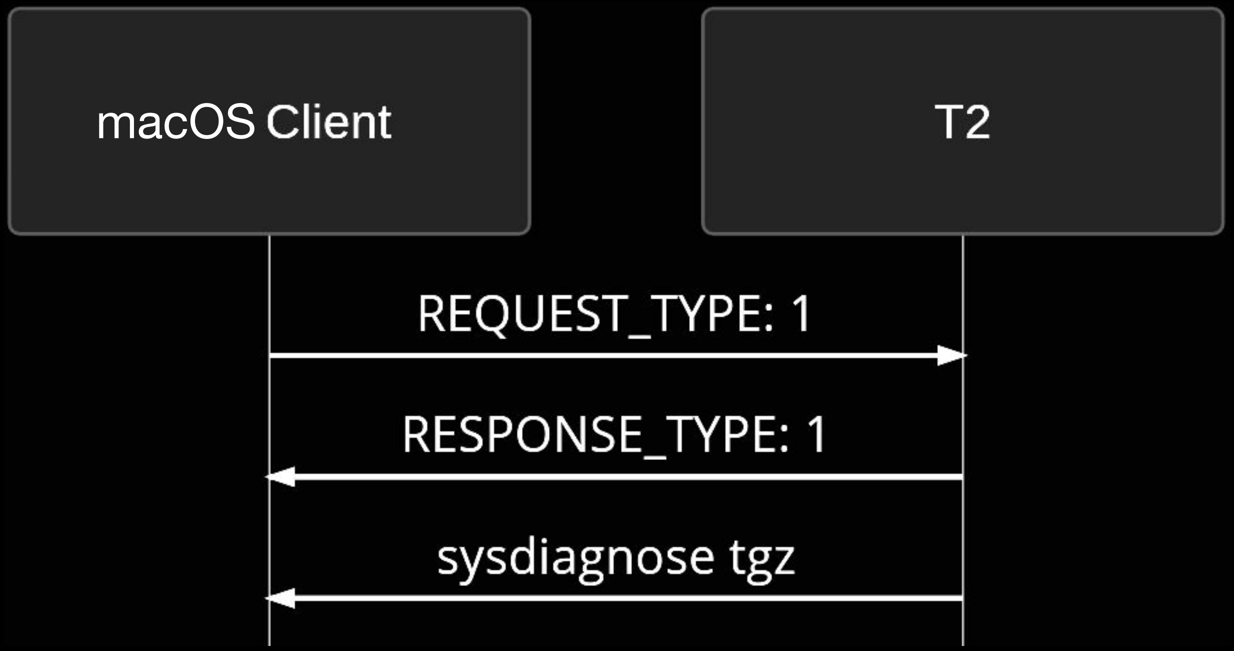
```
$ sniffer.py
```

Case Study: Sysdiagnose

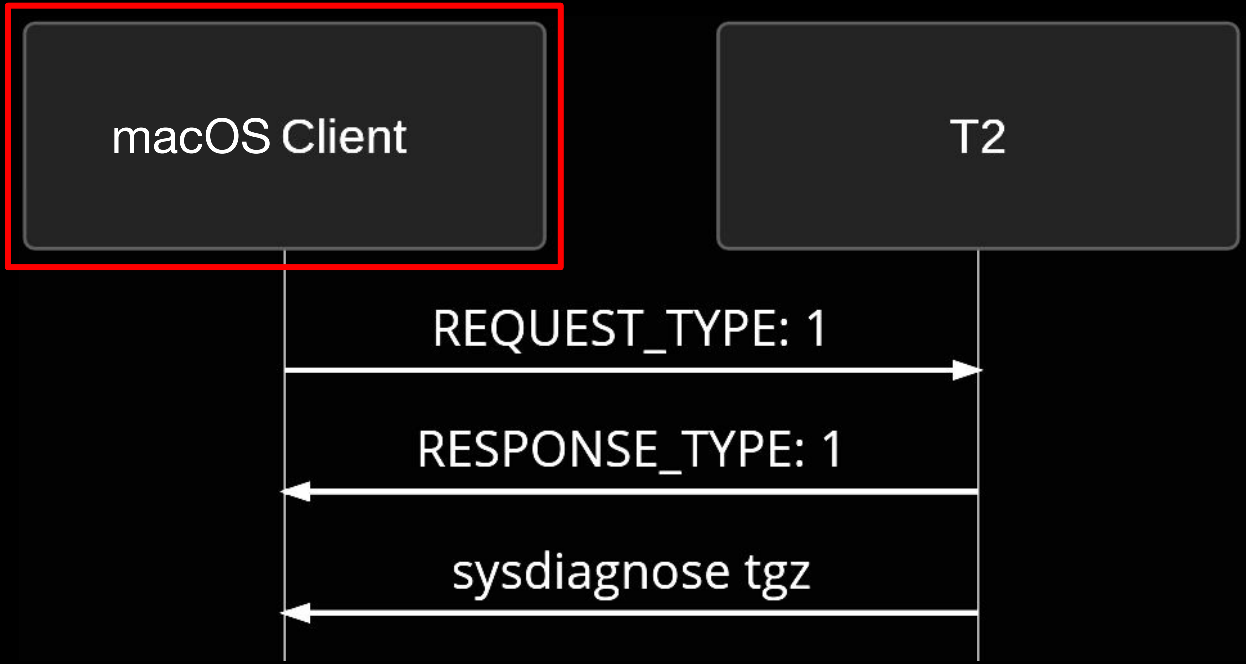
```
$ sniffer.py
...
imac opening stream 1 for communication on port 49155.
...
New HTTP/2 frame
New XPC Packet imac->t2 on HTTP/2 stream 1 TCP port 49155
XPC Wrapper: {
    Magic: 0x29b00b92
    Flags: 0b 00000000 00000000 00000001 00000001 (0x101)
    BodyLength: 0x30
    MessageId: 0x1
}
{
    "REQUEST_TYPE":
        uint64 0x0000000000000001: 1
}
```

{“REQUEST_TYPE”: 1}

Sysdiagnose Protocol (simplified)



Sysdiagnose Protocol (simplified)



Interacting with T2 Services

Connecting to Sysdiagnose Server (Before)

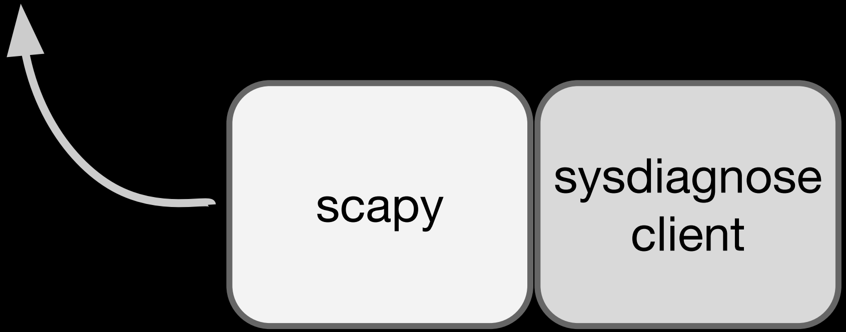
```
$ remotectl relay localbridge com.apple.sysdiagnose.remote
49923
```

Connecting to Sysdiagnose Server (Before)

```
$ remotectl relay localbridge com.apple.sysdiagnose.remote
49923

$ netstat -ant | grep 49923
tcp4       0      0  127.0.0.1.49923      *.*       LISTEN
```

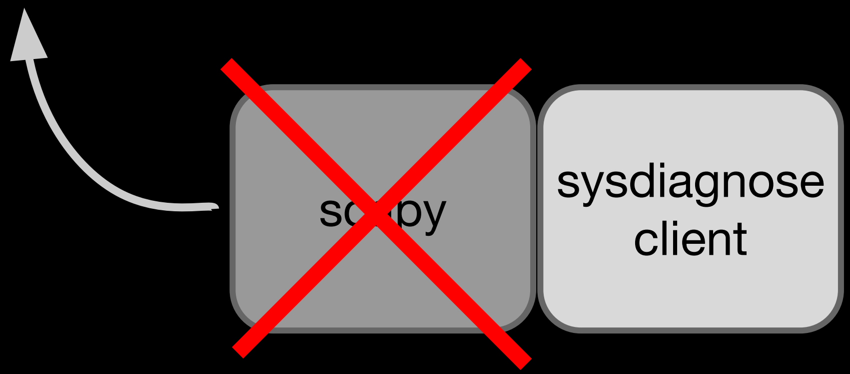
Connecting to Sysdiagnose Server (Before)

```
$ remotectl relay localbridge com.apple.sysdiagnose.remote
49923

$ netstat -ant | grep 49923
tcp4       0      0  127.0.0.1.49923      *.*         LISTEN
```



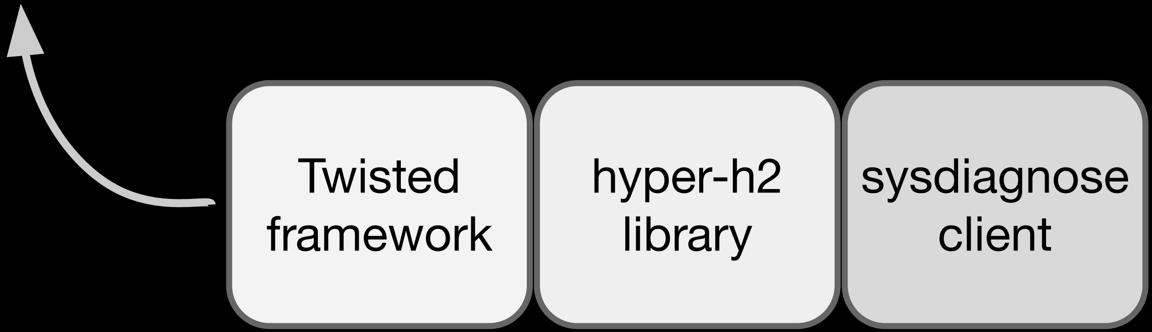
Connecting to Sysdiagnose Server (Before)

```
$ remotectl relay localbridge com.apple.sysdiagnose.remote
49923

$ netstat -ant | grep 49923
tcp4       0      0  127.0.0.1.49923      *.*        LISTEN
```



Connecting to Sysdiagnose Server (Before)

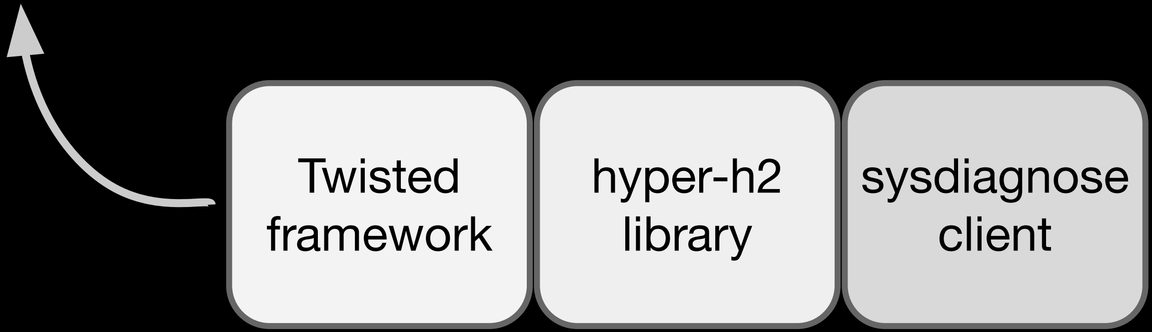
```
$ remotectl relay localbridge com.apple.sysdiagnose.remote
49923

$ netstat -ant | grep 49923
tcp4       0      0  127.0.0.1.49923      *.*        LISTEN
```



Connecting to Sysdiagnose Server (Before)

~~sudo~~

```
$ remotectl relay localbridge com.apple.sysdiagnose.remote
49923

$ netstat -ant | grep 49923
tcp4       0      0  127.0.0.1.49923        *.*          LISTEN
```



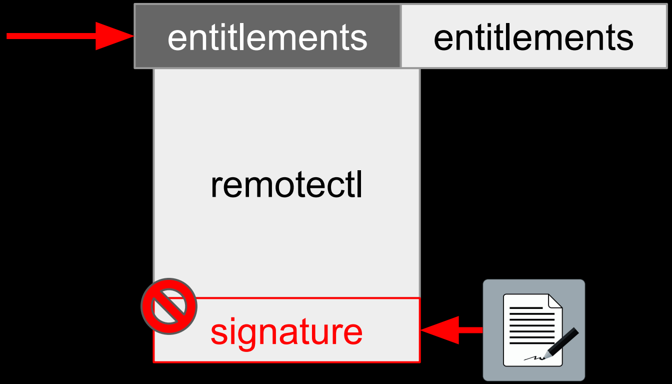
Connecting to Sysdiagnose Server (After)

```
# remotectl relay localbridge com.apple.sysdiagnose.remote
remotectl: Unable to connect to
localbridge/com.apple.sysdiagnose.remote: No such process
```

Make **`remotectl`** work again

`remotectl relay` gated by Entitlements

In 10.14.3+, **`remotectl relay`** appears to be gated by a new entitlement:
`com.apple.private.network.intcoproc.restricted`



Researchers can use **jtool** to insert this entitlement and self-sign a new **`remotectl`** binary

Disable SIP and **`amfid`** to allow **`remotectl`** binary to run

```
# csrutil disable # in recovery mode

# nvram boot-args="amfi_get_out_of_my_way=0x01" # reboot

# cp /usr/libexec/remotectl /tmp/
# cat << EOF > /tmp/entitlements.ent
... com.apple.private.network.intcoproc.restricted ...
EOF
# jtool --sign --ent /tmp/entitlements.ent --inplace /tmp/remotectl
```

Back to sysdiagnose client

Sysdiagnose Request and Response

```
$ sysdiagnose -c

...
{
  "REQUEST_TYPE":
    uint64 0x0000000000000001: 1
}
```

```
{
  "RESPONSE_TYPE":
    uint64 0x0000000000000001: 1
  "FILE_TX":
    MessageId: 0x5
    File transfer size:
        0x00000000005b49d7 5982679
  "FILE_NAME":
        "bridge_sysdiagnose_2019.01
        .18_16-57-46+0000_Bridge_OS
        _Bridge_16P375.tar.gz"
}
```

Sysdiagnose Options

```
$ sysdiagnose -cup

...
{
  "disableUIFeedback": True
  "shouldRunOSLogArchive": False
  "shouldRunLoggingTasks": False
  "shouldDisplayTarBall": False
  "shouldRunTimeSensitiveTasks": True
  "REQUEST_TYPE":
    uint64 0x0000000000000001: 1
}
```

Sysdiagnose Options

```
$ sysdiagnose -cup

...
{
  "disableUIFeedback": True
  "shouldRunOSLogArchive": False
  "shouldRunLoggingTasks": False
  "shouldDisplayTarBall": False
  "shouldRunTimeSensitiveTasks": True
  "REQUEST_TYPE":
    uint64 0x0000000000000001: 1
}
```

```
                        getMetrics bool
                      diagnosticID string
                     baseDirectory string
                          rootPath string
                       archiveName string
                embeddedDeviceType string
                     coSysdiagnose string
                     generatePlist bool
                         quickMode bool
              shouldDisplayTarBall bool
               shouldCreateTarBall bool
             shouldRunLoggingTasks bool
       shouldRunTimeSensitiveTasks bool
             shouldRunOSLogArchive bool
shouldRemoveTemporaryDirectory bool
             shouldGetFeedbackData bool
                  disableStreamTar bool
                 disableUIfeedback bool
                      setNoTimeOut bool
                      pidOrProcess string
                       capOverride NSData
                 warnProcWhitelist string
```

Sysdiagnose Options

```
$ sysdiagnose_client.py

...
{
  "REQUEST_TYPE":
    uint64 0x0000000000000001: 1
  "archiveName":
    "duolabs"
}
```

```
                  getMetrics bool
                diagnosticID string
               baseDirectory string
                    rootPath string
                 archiveName string
          embeddedDeviceType string
               coSysdiagnose string
               generatePlist bool
                   quickMode bool
        shouldDisplayTarBall bool
         shouldCreateTarBall bool
       shouldRunLoggingTasks bool
 shouldRunTimeSensitiveTasks bool
       shouldRunOSLogArchive bool
shouldRemoveTemporaryDirectory bool
       shouldGetFeedbackData bool
            disableStreamTar bool
           disableUIfeedback bool
                setNoTimeOut bool
                pidOrProcess string
                 capOverride NSData
           warnProcWhitelist string
```

Sysdiagnose Options

```
$ sysdiagnose_client.py

...
{
  "REQUEST_TYPE":
    uint64 0x0000000000000001: 1
  "archiveName":
    "duolabs"
}
```

```
{
  "RESPONSE_TYPE":
    uint64 0x0000000000000001: 1
  "MSG_TYPE":
    uint64 0x0000000000000002: 2
  "FILE_TX":
    MessageId: 0x58
    File transfer size:
        0x00000000004a22b6 4858550
  "FILE_NAME":
    "duolabs.tar.gz"
}
```

Further Exploration

We are unlikely to revisit this anytime soon

There are lots of other exposed services to be explored

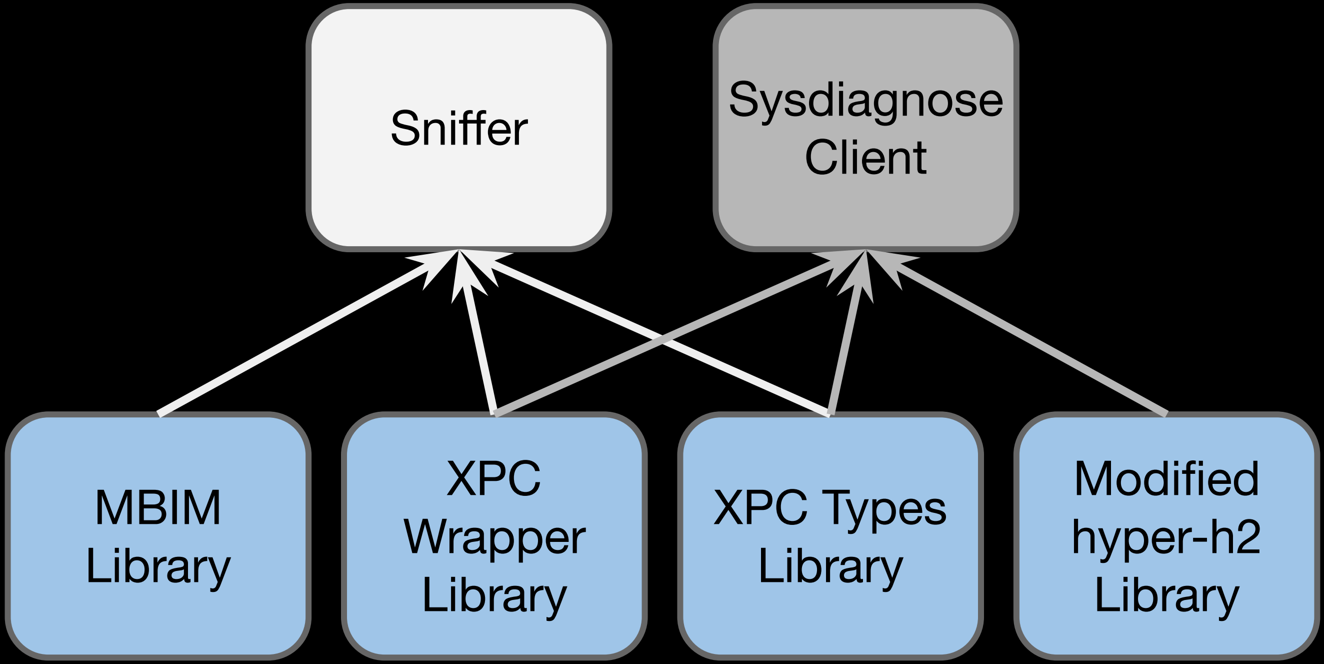
Fuzzing would be a great next step

The T2 chip is arguably the most advanced secure boot process -- validation of this approach to secure boot is valuable!

```
com.apple.CSCRemoteSupportd
com.apple.sysdiagnose.remote
com.apple.corespeech.xpc.remote.record
com.apple.xpc.remote.multiboot
com.apple.eos.LASecureIO
com.apple.osanalytics.logTransfer
com.apple.eos.BiometricKit
com.apple.aveservice
com.apple.powerchime.remote
com.apple.bridgeOSUpdated
com.apple.private.avvc.xpc.remote
com.apple.corecaptured.remoteservice
com.apple.icloud.findmydeviced.bridge
com.apple.mobileactivationd.bridge
com.apple.sysdiagnose.stackshot.remote
com.apple.multiverse.remote.bridgetime
com.apple.logd.remote-daemon
com.apple.corespeech.xpc.remote.control
```

Open Source Tooling

https://github.com/duo-labs/apple-t2-xpc/



Black Hat Sound Bytes

The T2 is a significant step forward towards bringing the same security properties of iOS to macOS.

The UEFI firmware images are still mutable by design and only validated on “first-boot” scenarios.

Hardware attacks appear to still be feasible, albeit through a new (eSPI) interface.

Mikhail Davidov
Research Technical Leader
@sirus
mdavidov@duo.com

Jeremy Erickson
Senior Research Engineer
@jlericks
jerickson@duo.com

Us: duo.com/labs

Papers: duo.sc/t2boot [duo.sc/t2xpc](duo.sc/t2xpc)

Backup Slides

Sysdiagnose Server Binary

```
{
  "REQUEST_TYPE":
    uint64 0x0000000000000001: 1
}
```

```
switch ( REQUEST_TYPE ) {
    case 1u:
        sd_ops_sysdiagnose(...);
    case 2u:
        sd_ops_stackshot(...);
    case 4u:
        sd_ops_cancel(...);
    case 5u:
        sd_ops_cancelAll(...);
    case 6u:
        sd_ops_userinterrupt(...);
    case 7u:
        sd_ops_statusPoll(...);
    case 8u:
        sd_ops_airdrop(...);
    case 9u:
        sd_ops_watchList(...);
    case 10u:
        sd_ops_deleteArchive(...);
```

Sysdiagnose Server Binary

```
{
  "REQUEST_TYPE":
    uint64 0x0000000000000001: 1
}
```

```
switch ( REQUEST_TYPE ) {
    case 1u:
        sd_ops_sysdiagnose(...);
    case 2u:
        sd_ops_stackshot(...);
    case 4u:
        sd_ops_cancel(...);
    case 5u:
        sd_ops_cancelAll(...);
    case 6u:
        sd_ops_userinterrupt(...);
    case 7u:
        sd_ops_statusPoll(...);
    case 8u:
        sd_ops_airdrop(...);
    case 9u:
        sd_ops_watchList(...);
    case 10u:
        sd_ops_deleteArchive(...);
```