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
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
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
Raw Storage

Intel Chipset

Raw Storage

PCle

eSPI

PCle
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](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

PROJECT EXPERIENCE

Foxconn, China.

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

<table>
<thead>
<tr>
<th>Security (SEC)</th>
<th>Pre EFI Initialization (PEI)</th>
<th>Driver Execution Environment (DXE)</th>
<th>Boot Device Select (BDS)</th>
<th>Transient System Load (TSL)</th>
<th>Run Time (RT)</th>
<th>After Life (AL)</th>
</tr>
</thead>
</table>

[Image: tianocore.github.io/master/images/PI_Boot_Phases.JPG]
UEFI Platform Initialization (PI) Boot Phases

Reset Vector

Pre Verifier → Processor Init → Chipset Init → Board Init → Device, Bus, or Service Driver → EFI Driver Dispatcher → Boot Manager

UEFI Interface

OS-Absent App

Transient OS Environment

Transient OS Boot Loader

OS-Present App

Final OS Boot Loader

Final OS Environment

Security (SEC) | Pre EFI Initialization (PEI) | Driver Execution Environment (DXE) | Boot Dev Select (BDS) | Transient System Load (TSL) | Run Time (RT) | After Life (AL)
---|---|---|---|---|---|---

[Image: tianocore.github.io/master/images/PI_Boot_Phases.JPG]
UEFI Platform Initialization (PI) Boot Phases

Security (SEC) | Pre EFI Initialization (PEI) | Driver Execution Environment (DXE) | Boot Dev Select (BDS) | Transient System Load (TSL) | Run Time (RT) | After Life (AL)

UEFI Interface

OS-Absent App

Transient OS Environment

Transient OS Boot Loader

OS-Present App

Final OS Environment

EFI Driver Dispatcher

Boot Manager

Intrinsic Services

Board Init

Chipset Init

Processor Init

Pre-Valife verify

Device, Bus, or Service Driver
UEFI Platform Initialization (PI) Boot Phases

- Security (SEC)
- Pre EFI Initialization (PEI)
- Driver Execution Environment (DXE)
- Boot Dev Select (BDS)
- Transient System Load (TSL)
- Run Time (RT)
- After Life (AL)

[Image: tianocore.github.io/master/images/PI_Boot_Phases.JPG]
Intel Chipset

Flash Chip

NVARS

UEFI FW
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

1. T2 Boot ROM
2. T2 iBoot
3. bridgeOS Kernel
4. PID 0 (launchd)
T2 Early Boot

PID 0 (launchd)
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

PID 0 (launchd) → MacEFIUtil -i → MacEFI.img4

T2 userland
T2 Early Boot

PID 0 (launchd) | MacEFIUtil -i | MacEFIManager.kext

MacEFI.img4

T2 userland | T2 kernel
T2 Early Boot

PID 0 (launchd) → MacEFIUtil -i → AFU.kext

MacEFI.img4

MacEFIManager.kext
T2 Early Boot

PID 0 (launchd) -> MacEFIUtl -i

MacEFI.img4

AFU.kext

MacEFIManager.kext

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

// Do we have an overriding boot argument?
PE_parse_boot_argv("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

PID 0 (launchd) -> MacEFIUtil -i

MacEFI.img4

MacEFIManager.kext

AFU.kext

Internal Storage
T2 Early Boot

PID 0 (launchd) → MacEFIUtil -i → MacEFIManager.kext → eSPI DMA

MacEFI.img4

AFU.kext

Internal Storage
T2 Early Boot

1. PID 0 (launchd)
2. MacEFIUtil -i
3. MacEFI.img4
4. MacEFIManager.kext
5. AFU.kext
6. Internal Storage
7. eSPI DMA

Read/write
MacEFIUtil -i → AppleSSM .kext → AppleSMC .kext → SMC “NESN”
Getting to S0

T2 kernel

AppleSSM .kext → SleepWakeHandler → MacEFIManager .kext → DoS0 → AppleSMC .kext → 0x8970 → SMC “NESN”
(simplified)

![Diagram showing the boot phases in UEFI interface]

<table>
<thead>
<tr>
<th>Security (SEC)</th>
<th>Pre EFI Initialization (PEI)</th>
<th>Driver Execution Environment (DXE)</th>
<th>Boot Dev Select (BDS)</th>
<th>Transient System Load (TSL)</th>
<th>Run Time (RT)</th>
<th>After Life (AL)</th>
</tr>
</thead>
</table>

[Image source: tianocore.github.io/master/images/PI_Boot_Phases.JPG]
Attacking Secure Boot

T2

On Die Boot ROM  iBoot  bridge OS Kernel  UEFI FW  Internal Storage  eSPI DMA  Intel PCH
Attacking Secure Boot

T2

- On Die Boot ROM
- iBoot
- bridge OS Kernel
- UEFI FW
- Internal Storage
- eSPI DMA
- Intel PCH

Bidirectional external bus!
Attacking Secure Boot

Only done on upgrades / first boot!

Bidirectional external bus!
Attacking Secure Boot

Only done on upgrades / first boot!

Bidirectional external bus!
Attacking Secure Boot

On Die Boot ROM → iBoot → bridge OS Kernel → Internal Storage → eSPI DMA → Intel PCH

Only done on upgrades / first boot!

Bidirectional external bus!
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
device 2AC47A5D-E9EF localbridge iBridge
$ 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.avservice
com.apple.powerchime.remote
com.apple.bridgeOSUpdated
com.apple.private.avvc.xpc.remote ...
...
$ 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 \texttt{VHC128} interface

Behaves like a SPAN port for \texttt{en6}
<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>0.00</td>
<td>16.1.2</td>
<td>host</td>
<td>USB</td>
<td>URB_BULK in (submitted)</td>
</tr>
<tr>
<td>...</td>
<td>0.00</td>
<td>fe80:aede:48ff:fe00:1122</td>
<td>fe80:aede:48ff:fe00:1122</td>
<td>TCP</td>
<td>49155 - 51570 [ACK] Seq=1 Ack</td>
</tr>
<tr>
<td>...</td>
<td>0.00</td>
<td>16.1.2</td>
<td>host</td>
<td>USB</td>
<td>URB_BULK in (submitted)</td>
</tr>
<tr>
<td>...</td>
<td>0.00</td>
<td>16.1.1</td>
<td>host</td>
<td>USB</td>
<td>URB_BULK out (submitted)</td>
</tr>
<tr>
<td>...</td>
<td>0.00</td>
<td>fe80:aede:48ff:fe00:1122</td>
<td>fe80:aede:48ff:fe33:4455</td>
<td>HTTP2</td>
<td>SETTINGS[0], WINDOW_UPDATE[0]</td>
</tr>
<tr>
<td>...</td>
<td>0.00</td>
<td>fe80:aede:48ff:fe33:4455</td>
<td>fe80:aede:48ff:fe00:1122</td>
<td>TCP</td>
<td>49155 - 51570 [ACK] Seq=1 Ack</td>
</tr>
<tr>
<td>...</td>
<td>0.00</td>
<td>16.1.2</td>
<td>host</td>
<td>USB</td>
<td>URB_BULK in (submitted)</td>
</tr>
<tr>
<td>...</td>
<td>0.00</td>
<td>16.1.2</td>
<td>host</td>
<td>USB</td>
<td>URB_BULK in (submitted)</td>
</tr>
<tr>
<td>...</td>
<td>0.00</td>
<td>16.1.1</td>
<td>host</td>
<td>USB</td>
<td>URB_BULK out (submitted)</td>
</tr>
<tr>
<td>...</td>
<td>0.00</td>
<td>16.1.1</td>
<td>host</td>
<td>USB</td>
<td>URB_BULK out (submitted)</td>
</tr>
<tr>
<td>...</td>
<td>0.00</td>
<td>fe80:aede:48ff:fe00:1122</td>
<td>fe80:aede:48ff:fe33:4455</td>
<td>TCP</td>
<td>51570 - 49155 [ACK] Seq=163 Ack</td>
</tr>
<tr>
<td>...</td>
<td>0.00</td>
<td>fe80:aede:48ff:fe00:1122</td>
<td>fe80:aede:48ff:fe33:4455</td>
<td>HTTP2</td>
<td>SETTINGS[0]</td>
</tr>
<tr>
<td>...</td>
<td>0.00</td>
<td>fe80:aede:48ff:fe00:1122</td>
<td>fe80:aede:48ff:fe33:4455</td>
<td>HTTP2</td>
<td>WINDOW_UPDATE[0], SETTINGS[0]</td>
</tr>
<tr>
<td>...</td>
<td>0.00</td>
<td>16.1.2</td>
<td>host</td>
<td>USB</td>
<td>URB_BULK in (submitted)</td>
</tr>
<tr>
<td>...</td>
<td>0.00</td>
<td>16.1.1</td>
<td>host</td>
<td>USB</td>
<td>URB_BULK out (submitted)</td>
</tr>
<tr>
<td>...</td>
<td>0.00</td>
<td>fe80:aede:48ff:fe00:1122</td>
<td>fe80:aede:48ff:fe33:4455</td>
<td>TCP</td>
<td>51570 - 49155 [ACK] Seq=172 Ack</td>
</tr>
<tr>
<td>...</td>
<td>0.00</td>
<td>fe80:aede:48ff:fe33:4455</td>
<td>fe80:aede:48ff:fe00:1122</td>
<td>TCP</td>
<td>49155 - 51570 [ACK] Seq=53 Ack</td>
</tr>
<tr>
<td>...</td>
<td>0.00</td>
<td>16.1.2</td>
<td>host</td>
<td>USB</td>
<td>URB_BULK in (submitted)</td>
</tr>
<tr>
<td>...</td>
<td>0.00</td>
<td>fe80:aede:48ff:fe33:4455</td>
<td>fe80:aede:48ff:fe00:1122</td>
<td>HTTP2</td>
<td>DATA[1][Malformed Packet]</td>
</tr>
</tbody>
</table>
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 0000 = Stream Identifier: 1
[Pad Length: 0]

Data: 920bb029010100003000000000000000000000000000000...

0080 80 18 10 04 0c 15 00 00 01 01 08 0a 3d 97 6e ed
0090 3f a0 26 d9 00 00 48 00 00 00 00 00 00 00 00 01 92 0b b0
00a0 29 01 01 00 00 30 00 00 00 00 00 00 00 00 00 01 00 00
00b0 00 00 00 00 00 42 37 13 42 05 00 00 00 00 00 f0 00
00c0 00 20 00 00 00 00 01 00 00 00 52 45 51 55 45 53 54
00d0 5f 54 59 50 45 00 00 00 00 00 40 00 00 01 00 00
00e0 00 00 00 00 00
Decoding Message Layers
MBIM (USB)

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

Encapsulation

MBIM

Ethernet IP TCP (1)
Ethernet IP TCP (2)
Ethernet IP TCP (N)

MTU Chunking

HTTP/2

Raw Data

OR

XPC Wrapper

OR

Empty Payload, Signaling Only

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

MBIM

Encapsulation

Ethernet IP TCP (1)

Ethernet IP TCP (2)

Ethernet IP TCP (N)

MTU Chunking

HTTP/2

Raw Data

XPC Wrapper

OR

Empty Payload, Signaling Only

OR

XPC Object
# XPC Wrapper

<table>
<thead>
<tr>
<th>4 bytes</th>
<th>4 bytes</th>
<th>8 bytes</th>
<th>8 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>magic</td>
<td>flags</td>
<td>body length</td>
<td>message id</td>
</tr>
</tbody>
</table>

#### Flag bits:

| 00000000 00000000 00000000 00000000 | - Always set |
| 00000000 00000000 00000000 00000000 | - Data present |
| 00000000 00000001 00000000 00000000 | - Heartbeat request |
| 00000000 00000000 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 |

- Used for signalling. Often incremented or repeated between request/response.
Decoding XPC
Overview of XPC

```c
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

```c
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

```c
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

```c
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);
```
XPC Header

```
42 37 13 42 05 00 00 00 dictionary{...}
```
XPC Types

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

Types:

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPC_NULL</td>
<td>0x00001000</td>
<td>XPC_ARRAY</td>
<td>0x0000e000</td>
</tr>
<tr>
<td>XPC_BOOL</td>
<td>0x00002000</td>
<td>XPC_DICTIONARY</td>
<td>0x0000f000</td>
</tr>
<tr>
<td>XPC_INT64</td>
<td>0x00003000</td>
<td>XPC_ERROR</td>
<td>0x00010000</td>
</tr>
<tr>
<td>XPC_UINT64</td>
<td>0x00004000</td>
<td>XPC_CONNECTION</td>
<td>0x00011000</td>
</tr>
<tr>
<td>XPC_DOUBLE</td>
<td>0x00005000</td>
<td>XPC_ENDPOINT</td>
<td>0x00012000</td>
</tr>
<tr>
<td>XPC_POINTER</td>
<td>0x00006000</td>
<td>XPC_SERIALIZER</td>
<td>0x00013000</td>
</tr>
<tr>
<td>XPC_DATE</td>
<td>0x00007000</td>
<td>XPC_PIPE</td>
<td>0x00014000</td>
</tr>
<tr>
<td>XPC_DATA</td>
<td>0x00008000</td>
<td>XPC_MACH_RECV</td>
<td>0x00015000</td>
</tr>
<tr>
<td>XPC_STRING</td>
<td>0x00009000</td>
<td>XPC_BUNDLE</td>
<td>0x00016000</td>
</tr>
<tr>
<td>XPC_UUID</td>
<td>0x000a0000</td>
<td>XPC_SERVICE</td>
<td>0x00017000</td>
</tr>
<tr>
<td>XPC_FD</td>
<td>0x000b0000</td>
<td>XPC_SERVICE_INSTANCE</td>
<td>0x00018000</td>
</tr>
<tr>
<td>XPC_SHMEM</td>
<td>0x000c0000</td>
<td>XPC_ACTIVITY</td>
<td>0x00019000</td>
</tr>
<tr>
<td>XPC_MACH_SEND</td>
<td>0x000d0000</td>
<td>XPC_FILE_TRANSFER</td>
<td>0x0001a000</td>
</tr>
</tbody>
</table>
# XPC Types

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPC_NULL</td>
<td>0x00001000</td>
</tr>
<tr>
<td>XPC_BOOL</td>
<td>0x00002000</td>
</tr>
<tr>
<td>XPC_INT64</td>
<td>0x00003000</td>
</tr>
<tr>
<td>XPC_UINT64</td>
<td>0x00004000</td>
</tr>
<tr>
<td>XPC_DOUBLE</td>
<td>0x00005000</td>
</tr>
<tr>
<td>XPC_POINTER</td>
<td>0x00006000</td>
</tr>
<tr>
<td>XPC_DATE</td>
<td>0x00007000</td>
</tr>
<tr>
<td>XPC_DATA</td>
<td>0x00008000</td>
</tr>
<tr>
<td>XPC_STRING</td>
<td>0x00009000</td>
</tr>
<tr>
<td>XPC_UUID</td>
<td>0x0000a000</td>
</tr>
<tr>
<td>XPC_FD</td>
<td>0x0000b000</td>
</tr>
<tr>
<td>XPC_SHMEM</td>
<td>0x0000c000</td>
</tr>
<tr>
<td>XPC_MACH_SEND</td>
<td>0x0000d000</td>
</tr>
<tr>
<td>XPC_ARRAY</td>
<td>0x0000e000</td>
</tr>
<tr>
<td>XPC_DICTIONARY</td>
<td>0x0000f000</td>
</tr>
<tr>
<td>XPC_ERROR</td>
<td>0x00010000</td>
</tr>
<tr>
<td>XPC_CONNECTION</td>
<td>0x00011000</td>
</tr>
<tr>
<td>XPC_ENDPOINT</td>
<td>0x00012000</td>
</tr>
<tr>
<td>XPC_SERIALIZER</td>
<td>0x00013000</td>
</tr>
<tr>
<td>XPC_PIPE</td>
<td>0x00014000</td>
</tr>
<tr>
<td>XPC_MACH_RECV</td>
<td>0x00015000</td>
</tr>
<tr>
<td>XPC_BUNDLE</td>
<td>0x00016000</td>
</tr>
<tr>
<td>XPC_SERVICE</td>
<td>0x00017000</td>
</tr>
<tr>
<td>XPC_SERVICE_INSTANCE</td>
<td>0x00018000</td>
</tr>
<tr>
<td>XPC_ACTIVITY</td>
<td>0x00019000</td>
</tr>
<tr>
<td>XPC_FILE_TRANSFER</td>
<td>0x0001a000</td>
</tr>
</tbody>
</table>
XPC Fixed-size objects: uint64

<table>
<thead>
<tr>
<th>4-byte type</th>
<th>known-length value</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 40 00 00</td>
<td>05 00 00 00 00 00 00 00 00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><em><strong>type</strong></em></th>
<th><em><strong><strong><strong>value</strong></strong></strong></em>_</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint64</td>
<td>5</td>
</tr>
</tbody>
</table>
XPC Variable-length Objects: string

<table>
<thead>
<tr>
<th>4-byte type</th>
<th>4-byte length</th>
<th>N-byte value</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 90 00 00</td>
<td>09 00 00 00</td>
<td>64 75 6f 6c</td>
</tr>
<tr>
<td><em><strong>type</strong></em></td>
<td><em><strong>length</strong></em></td>
<td>61 62 73 21</td>
</tr>
<tr>
<td>string</td>
<td>9</td>
<td>00 00 00 00</td>
</tr>
<tr>
<td>duolabs!\0</td>
<td></td>
<td>00 00 00 00</td>
</tr>
</tbody>
</table>
XPC Compound Objects: dictionary

{“five”: 5, “six”: 6}
Other XPC Objects: file_transfer

<table>
<thead>
<tr>
<th>4-byte type</th>
<th>msg_id</th>
<th>dict type</th>
<th>length: 0x14</th>
<th>1 entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>key &quot;s&quot;</td>
<td>uint64 type</td>
<td>file_transfer_size</td>
<td></td>
</tr>
</tbody>
</table>

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

-\texttt{c} flag retrieves diagnostic information from T2 chip

We can monitor the communications on the \texttt{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**: 920bb029010100003000000000000000000100000000000000

---

<table>
<thead>
<tr>
<th>0080</th>
<th>80 18 10 04 0c 15 00 00 01 01 08 0a 3d 97 6e ed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0090</td>
<td>3f a0 26 d9 00 00 48 00 00 00 00 00 00 01 92 0b b0</td>
</tr>
<tr>
<td>00a0</td>
<td>29 01 01 00 00 30 00 00 00 00 00 00 00 01 00 00</td>
</tr>
<tr>
<td>00b0</td>
<td>00 00 00 00 00 00 42 37 13 42 05 00 00 00 00 f0 00</td>
</tr>
<tr>
<td>00c0</td>
<td>00 20 00 00 00 00 01 00 00 00 52 45 51 55 45 53 54</td>
</tr>
<tr>
<td>00d0</td>
<td>5f 54 59 50 45 00 00 00 00 00 00 40 00 00 01 00 00</td>
</tr>
<tr>
<td>00e0</td>
<td>00 00 00 00 00</td>
</tr>
</tbody>
</table>

...
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
}
Sysdiagnose Protocol (simplified)

REQUEST_TYPE: 1

RESPONSE_TYPE: 1

sysdiagnose tgz
Sysdiagnose Protocol (simplified)

macOS Client

REQUEST_TYPE: 1

RESPONSE_TYPE: 1

sysdiagnose tgz

T2
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

Twisted framework

hyper-h2 library

sysdiagnose client
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 (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
}
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

...  

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

```json
{
  "RESPONSE_TYPE": uint64 0x0000000000000001: 1
  "MSG_TYPE": uint64 0x0000000000000002: 2
  "FILE_TX":
    MessageId: 0x58
    File transfer size: 0x0000000000000001: 1
    "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!
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.
Backup Slides
Sysdiagnose Server Binary

```c
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

```c
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(...);

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