
black hat
EUROPE 2019
DECEMBER 2-5, 2019
EXCEL LONDON, UK
Aleph Research
HCL AppScan
Simplifying iOS Research:
Booting the iOS Kernel to an Interactive
Bash Shell on QEMU
#BHEU @BLACK HAT EVENTS

How current iOS research is done

- Third party iOS emulator on a remote server
- Development fused iPhone
- Off the shelf iPhone – jailbroken
- Off the shelf iPhone – no jailbreak

iPhone panic log


```
{"bug_type":"210","timestamp":"2019-01-31 00:00:31.72 +0100","os_version":"iPhone OS 11.4.1
    (15G77)","incident_id":"C8D49F0C-DBCD-4217-848A-4E3409C968D2"}
{
  "build" : "iPhone OS 11.4.1 (15G77)",
  "product" : "iPhone9,4",
  "kernel" : "Darwin Kernel Version 17.7.0: Mon Jun 11 19:06:26 PDT 2018; root:xnu-4570.70.24~3\/RELEASE_ARM64_T8010",
  "incident" : "C8D49F0C-DBCD-4217-848A-4E3409C968D2",
  "crashReporterKey" : "bbd821d8854956ba61dd35c63d4c25144a58ad9b",
  "date" : "2019-01-31 00:00:28.39 +0100",
  "panicString" : "panic(cpu 0 caller 0xffffffff01b3f8cac): Unaligned kernel data abort. (saved state:
      0xffffffe024a0b290)\n\t  x0: 0xffffffe006405507  x1:  0xffffffe0005fb471  x2:  0x0000000000000001  x3:
      0x0000000000000000\n\t  x4: 0x00000000ffffffff  x5:  0x0000000000000001  x6:  0x0000000000000000  x7:
      0xffffffff01ace5a1c\n\t  x8: 0xffffffe0005fb470  x9:  0x0000000000000001  x10: 0x00000000001071ae  x11:
      0x00000000001071ad\n\t  x12: 0x0000000000000000 x13: 0x0000000000000138  x14: 0x0000000000000000  x15:
      0x0000000000000000\n\t  x16: 0xffffffff01b70a014 x17: 0x0000000000000000  x18: 0xffffffff01b2e1000  x19:
      0xffffffe0064054ff\n\t  x20: 0x0000000000000003 x21: 0x0000000000000001  x22: 0xffffffff01b825000  x23:
      0xffffffff01b2f87e8\n\t  x24: 0x0000000000000a31 x25: 0xffffffe007474000  x26: 0xffffffe0005fb470  x27:
      0xffffffe0003a5c00\n\t  x28: 0x0000000000000000 fp:  0xffffffe024a0b640  lr:  0xffffffff01b2f87a0  sp:
      0xffffffe024a0b5e0\n\t  pc:  0xffffffff01b301c7c cpsr: 0xa0400304         esr: 0x96000021          far:
      0xffffffe006405507\n\nDebugger message: panic\nMemory ID: 0x1\nOS version: 15G77\nKernel version: Darwin Kernel
      Version 17.7.0: Mon Jun 11 19:06:26 PDT 2018; root:xnu-4570.70.24~3\/RELEASE_ARM64_T8010\nKernelCache UUID:
      C50E403D58F345EADBFECCF458171791\niBoot version: iBoot-4076.70.15\nsecure boot?: YES\nPaniclog version: 9\nKernel
      slide:     0x0000000014200000\nKernel text base: 0xffffffff01b204000\nEpoch Time:        sec       usec\n  Boot    :
      0x5c522a5b 0x0008dae2\n  Sleep   : 0x00000000 0x00000000\n  Wake    : 0x00000000 0x00000000\n  Calendar: 0x5c522c57
      0x00026e6e\n\nPanicked task 0xffffffe0003e1ce8: 27417 pages, 234 threads: pid 0: kernel_task\nPanicked thread:
      0xffffffe0005fb470, backtrace: 0xffffffe024a0aa90, tid: 401\n\t\t  lr: 0xffffffff01b3f96e8  fp:
      0xffffffe024a0abd0\n\t\t  lr: 0xffffffff01b2e15f4  fp: 0xffffffe024a0abe0\n\t\t  lr: 0xffffffff01b313a44  fp:
      0xffffffe024a0af50\n\t\t  lr: 0xffffffff01b313dd4  fp: 0xffffffe024a0afb0\n\t\t  lr: 0xffffffff01b313c00  fp:
      0xffffffe024a0afd0\n\t\t  lr: 0xffffffff01b3f8cac  fp: 0xffffffe024a0b130\n\t\t  lr: 0xffffffff01b3f9dfc  fp:
      0xffffffe024a0b270\n\t\t  lr: 0xffffffff01b2e15f4  fp: 0xffffffe024a0b280\n\t\t  lr: 0xffffffff01b301c7c  fp:
      0xffffffe024a0b640\n\t\t  lr: 0xffffffff01b2f87a0  fp: 0xffffffe024a0b690\n\t\t  lr: 0xffffffff01b2fffa0  fp:
      0xffffffe024a0b770\n\t\t  lr: 0xffffffff01b306be8  fp: 0xffffffe024a0b850\n\t\t  lr: 0xffffffff01b339988  fp:
      0xffffffe024a0b8e0\n\t\t  lr: 0xffffffff01b634b84  fp: 0xffffffe024a0b950\n\t\t  lr: 0xffffffff01b654448  fp:
      0xffffffe024a0b980\n\t\t  lr: 0xffffffff01b654124  fp: 0xffffffe024a0bab0\n\t\t  lr: 0xffffffff01b654e1c  fp:
      0xffffffe024a0bb30\n\t\t  lr: 0xffffffff01b652ea8  fp: 0xffffffe024a0bc90\n\t\t  lr: 0xffffffff01b2ec500  fp:
      0x0000000000000000\n\n",
  "panicFlags" : "0x2",
  "otherString" : "\n** Stackshot Succeeded ** Bytes Traced 202128 **\n",
  "memoryStatus" :
      {"compressorSize":0,"compressions":0,"decompressions":0,"busyBufferCount":0,"pageSize":16384,"memoryPressure":false
      ,"memoryPages":
      {"active":46523,"throttled":0,"fileBacked":69049,"wired":32283,"purgeable":1282,"inactive":14971,"free":48793
      ,"speculative":29929}},
```

Jonathan Afek

- Aleph Research group manager at HCL/AppScan
- 15 years of experience in security research and low level development including vulnerability research, Linux kernel, storage systems, WiFi systems and FW, security systems and more.

iOS on QEMU work done by @zhuowei (Worth Doing Badly)

QEMU

From Wikipedia, the free encyclopedia

QEMU (short for **Quick EMUlator**)[2] is a free and open-source emulator that performs hardware virtualization.

Past Research – Worth Doing Badly (@zhuowei)

- Chosen version is iPhone X iOS 12 beta 4
- Extracted the kernel image and the device tree from the software update package
- Kernel (patched), device tree and the kernel boot arguments were loaded in memory
- iOS RAMDisk was loaded in memory
- UART serial support was achieved (output only)
- Kernel was booted
- Launchd was executed (no non-Apple executables executed)

Past Research – Worth Doing Badly (@zhuowei)

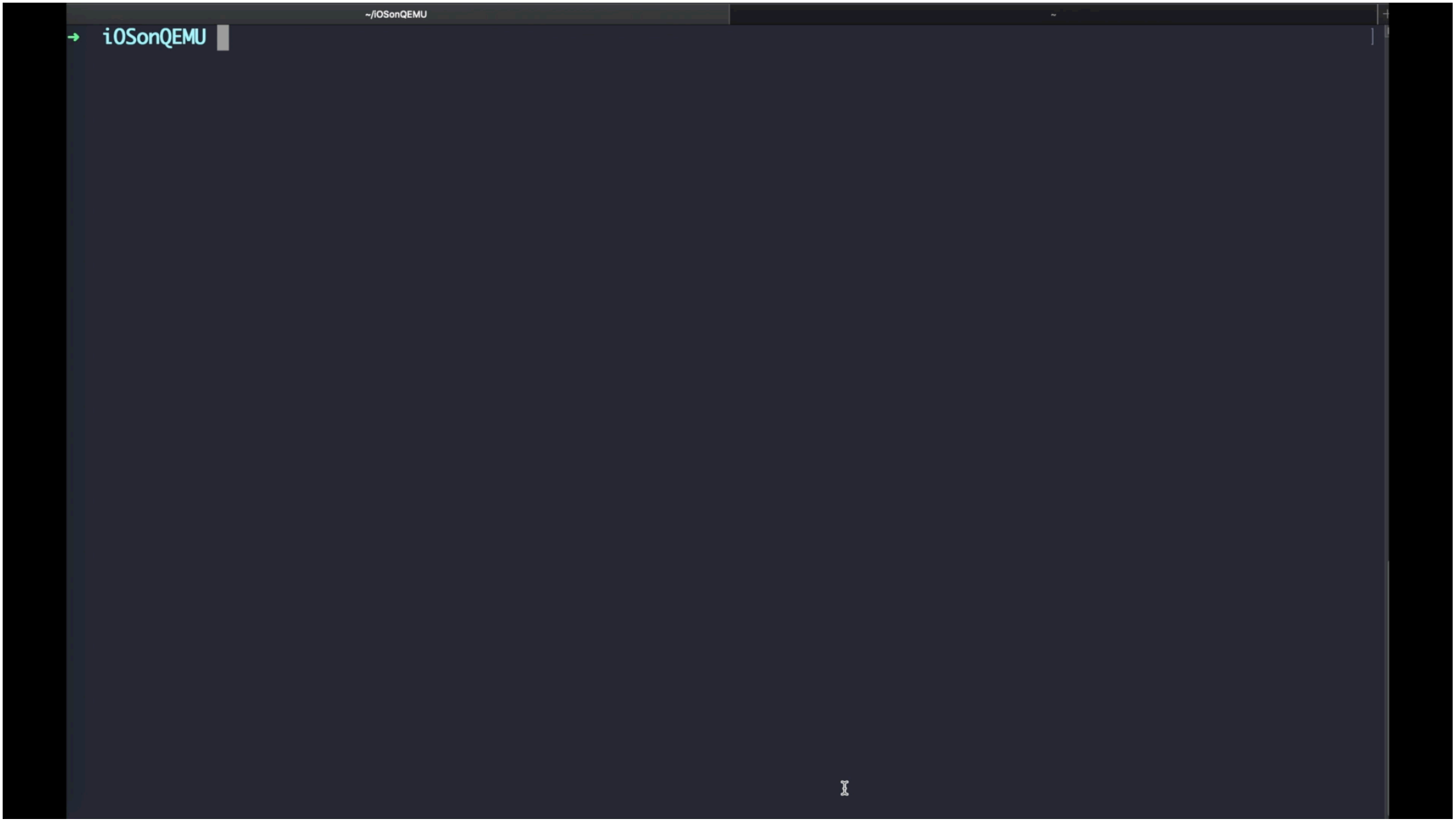
```
BSD root: md0, major 2, minor 0
apfs_vfsop_mountroot:1468: apfs: mountroot called!
apfs_vfsop_mount:1231: unable to root from devvp <ptr> (root_device): 2
apfs_vfsop_mountroot:1472: apfs: mountroot failed, error: 2
hfs: mounted PeaceSeed16A5327f.arm64UpdateRamDisk on device b(2, 0)
: : Darwin Bootstrapper Version 6.0.0: Mon Jul  9 00:39:56 PDT 2018; root:libxpc_execu
boot-args = debug=0x8 kextlog=0xfff cpus=1 rd=md0
Thu Jan  1 00:00:05 1970 localhost com.apple.xpc.launchd[1] <Notice>: Restore environm
```

Goals of our project

- Booting iOS on QEMU with no kernel patches
- Supporting hardware (disk, display, touch, sound, multiple CPUs, Interrupt controllers, etc…)
- Supporting different iOS versions
- Conducting iOS security research
- Learning about iOS and QEMU internals

Status of our project

- Booting Secure Monitor and the kernel (unpatched)
- Executing a user-mode app over launchd
- Running an interactive bash shell on an iOS kernel on QEMU
- Supporting only on iOS 12.1 for iPhone 6s plus

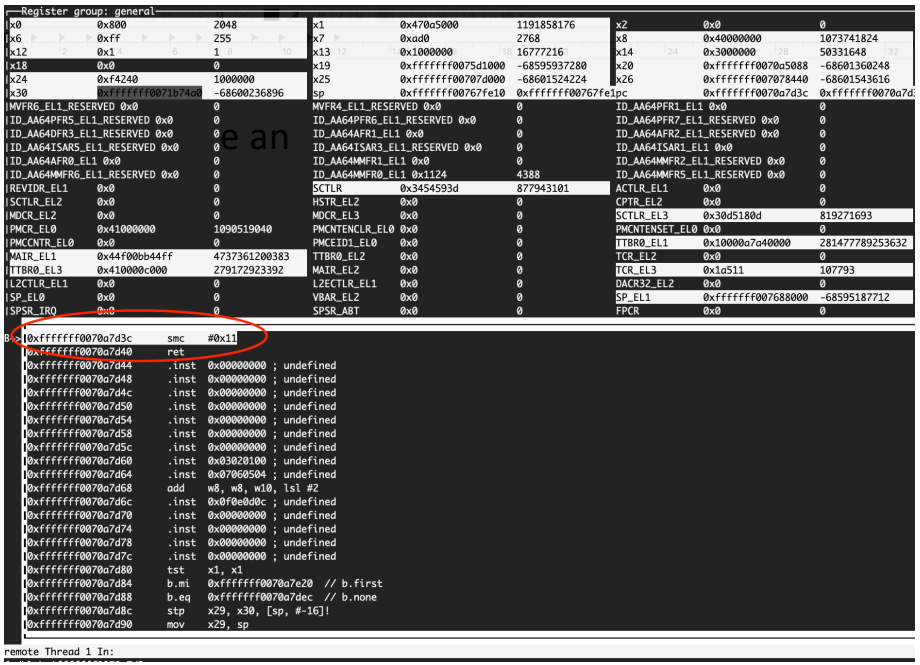

~/iOSonQEMU
~
iOSonQEMU

Agenda

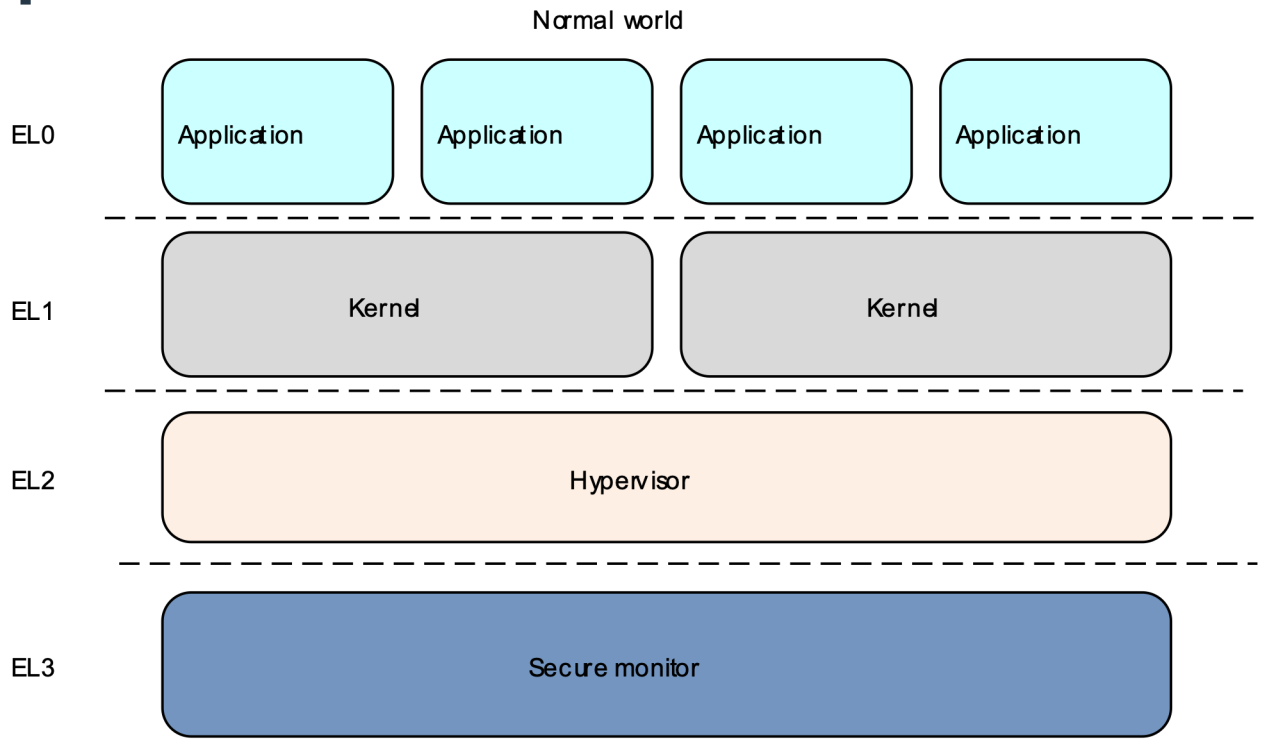
- Past public research on iOS on QEMU
- **iOS kernel boot process**
- Execution of non-apple executables with Trust Cache
- Bash execution with launchd
- UART interactive I/O
- Next steps

iOS kernel boot process

- Start booting the kernelcache code in EL1 as done by @zhuowei
- Crash on SMC instruction (Secure Monitor Call)



iOS kernel boot process



from https://developer.arm.com/docs/den0024/a/fundamentals-of-armv8

iOS kernel boot process

- Secure Monitor starts execution at boot in EL3
- It resides in a secure memory location inaccessible from EL1 (kernel code)
- It services SMC calls from the kernel (similar to how system calls from user apps to the kernel are serviced)
- It is responsible for KPP (Kernel Patch Protection) in our system

iOS kernel boot process

- iPhone X uses KTRR (hardware mechanism to prevent patches) and no longer uses a Secure Monitor for KPP (Kernel Patch Protection)

iOS kernel boot process

- The kernel needs a secure monitor to service its SMCs

iOS kernel boot process

Any ideas?

iOS kernel boot process

- Loading the Secure Monitor image for iOS 12.1 for iPhone 6s plus in EL3 and start executing
  - Loading the image at its preferred address (secure memory) with the image's boot args at the next page and start execution at the entry point in EL3

iOS kernel boot process

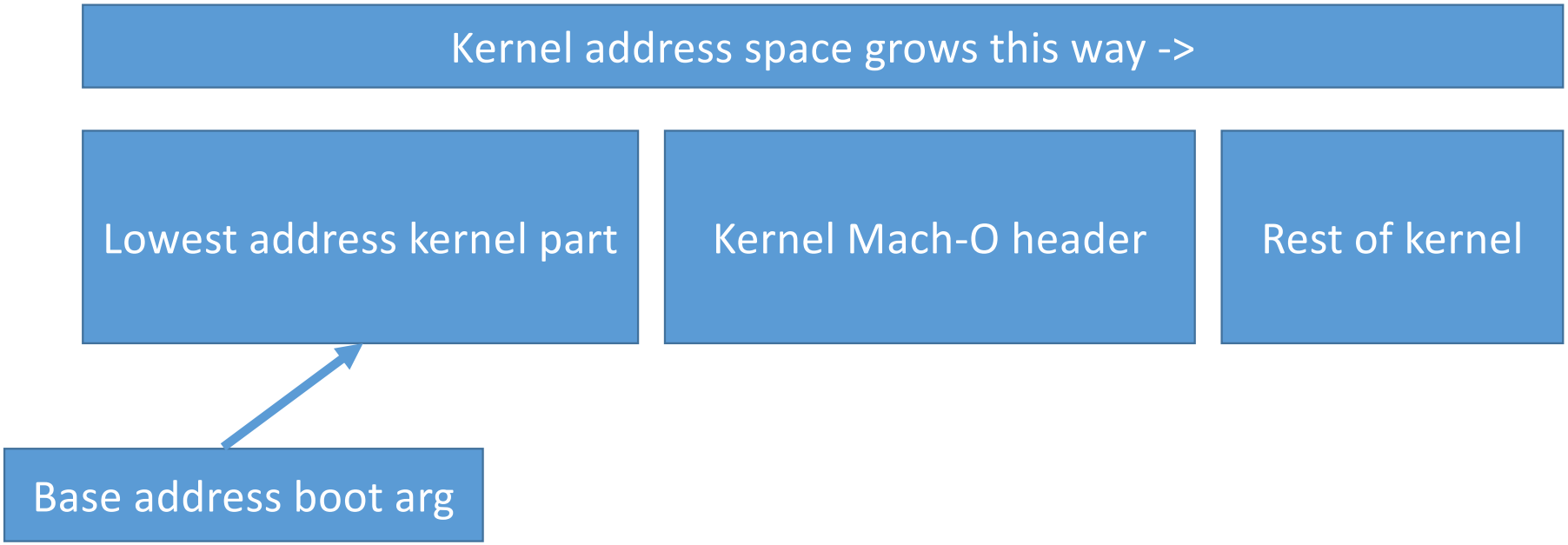
- Loading the Secure Monitor image for iOS 12.1 for iPhone 6s plus in EL3 and start executing
  - Data abort for trying to parse the kernelcache Mach-O header to decide which areas need which protections

iOS kernel boot process

```
...04100005230 d7 00 00 14     b           LAB_ffff00410000558c

                          LAB_ffff004100005234                            XREF[1]
...04100005234 68 0f 00 f9     str         x8,[x27, #0x18]=>DAT_ffff004100013248
...04100005238 d5 00 00 14     b           LAB_ffff00410000558c

                          LAB_ffff00410000523c                            XREF[2]

...0410000523c e8 02 40 b9     ldr         w8,[x23]
...04100005240 1f 65 00 71     cmp         w8,#0x19
...04100005244 41 1a 00 54     b.ne        LAB_ffff00410000558c
...04100005248 e8 12 40 f9     ldr         x8,[x23, #0x20]
...0410000524c 08 1a 00 b4     cbz         x8,LAB_ffff00410000558c
...04100005250 fc 22 00 91     add         x28,x23,#0x8
...04100005254 e2 03 1c 32     mov         w2,#0x10
...04100005258 e0 03 1c aa     mov         x0,x28
...0410000525c e1 03 14 aa     mov         x1=>s___TEXT_EXEC_ffff004100007064,x20
...04100005260 f3 06 00 94     bl          FUN_ffff004100006e2c
...04100005264 40 09 00 34     cbz         w0,LAB_ffff00410000538c
...04100005268 e2 03 1c 32     mov         w2,#0x10
...0410000526c e0 03 1c aa     mov         x0,x28
...04100005270 e1 03 15 aa     mov         x1=>s___PLK_TEXT_EXEC_ffff004100007070,
...04100005274 ee 06 00 94     bl          FUN_ffff004100006e2c
...04100005278 a0 08 00 34     cbz         w0,LAB_ffff00410000538c
...0410000527c e2 03 1c 32     mov         w2,#0x10
...04100005280 e0 03 1c aa     mov         x0,x28
...04100005284 e1 ef 00 10     adr         x1=>s___TEXT_ffff004100007080,-0xffbeff
...04100005288 1f 20 03 d5     nop
...0410000528c e8 06 00 94     bl          FUN_ffff004100006e2c
...04100005290 e0 0b 00 34     cbz         w0,LAB_ffff00410000540c
...04100005294 e2 03 1c 32     mov         w2,#0x10
...04100005298 e0 03 1c aa     mov         x0,x28
...0410000529c 41 ef 00 70     adr         x1=>s   PRELINK TEXT ffff004100007087,-
```

```
  DAT_ffff004100013238 = lParm2;
  DAT_ffff004100013250 = DAT_ffff004100013240;
  DAT_ffff004100013258 = DAT_ffff004100013248;
LAB_ffff00410000523c:
  ppuVar8 = DAT_ffff004100013240;
  if ((*piVar19 == 0x19) && (ppuVar8 = DAT_ffff004100013240, *(longlong *)(piVar
    piVar1 = piVar19 + 2;
    iVar6 = FUN_ffff004100006e2c(piVar1,"__TEXT_EXEC",0x10);
    if ((iVar6 == 0) || (iVar6 = FUN_ffff004100006e2c(piVar1,"__PLK_TEXT_EXEC",0
      ppuVar12 = DAT_ffff004100013230;
      puVar7 = *(ulonglong **)(piVar19 + 6);
      puVar10 = *(ulonglong **)(piVar19 + 8);
      ppuVar8 = DAT_ffff004100013230 + 1;
      DAT_ffff004100013230 = DAT_ffff004100013230 + 4;
      *ppuVar8 = puVar7;
      ppuVar12[2] = puVar10;
      ppuVar12[3] = (ulonglong *)0x2;
      if (puVar7 < DAT_ffff004100013260) {
        DAT_ffff004100013260 = puVar7;
      }
      uVar11 = ((longlong)puVar10 + (longlong)puVar7) - (longlong)DAT_ffff004100
      if (DAT_ffff004100013268 < uVar11) {
        DAT_ffff004100013268 = uVar11;
      }
      if (DAT_ffff004100013248 == (ulonglong **)0x0) {
        *ppuVar12 = (ulonglong *)0x0;
LAB_ffff0041000054a0:
        ppuVar8 = (ulonglong **)&DAT_ffff004100013248;
      }
      else {
        ppuVar8 = (ulonglong **)0x0;
        ppuVar5 = DAT_ffff004100013248;
        do {
```

iOS kernel boot process



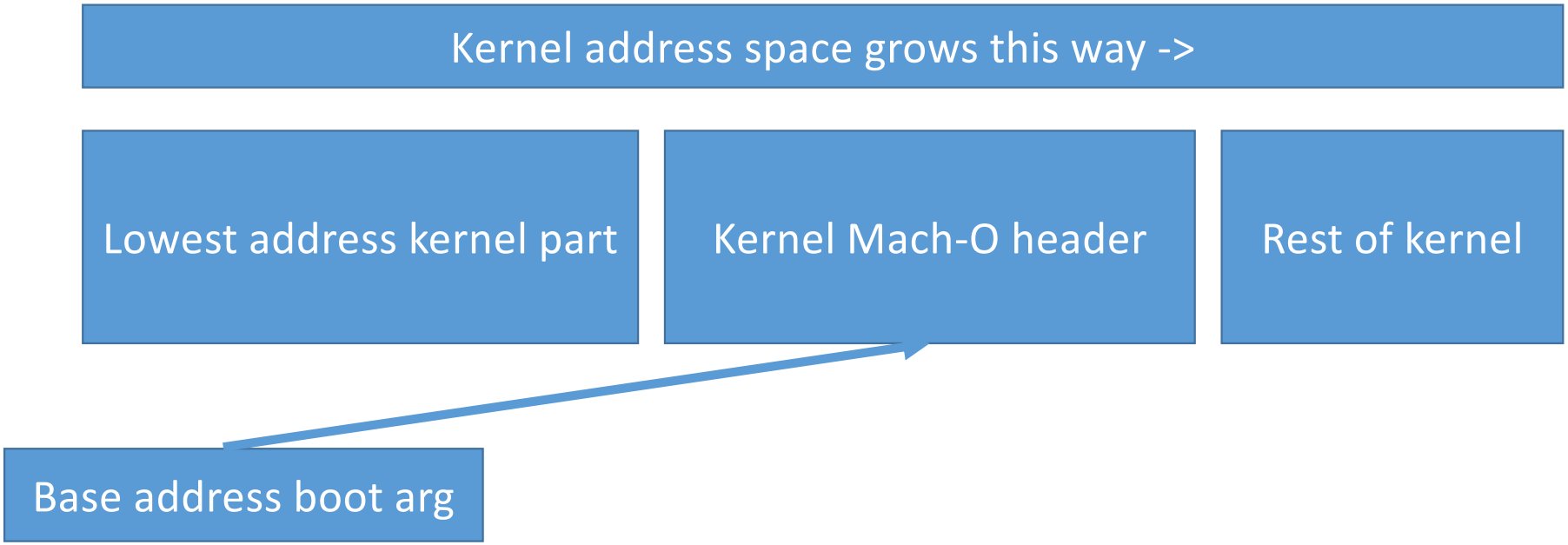
iOS kernel boot process

- The kernel needs a secure monitor to service its SMCs
- The secure monitor requires the *base address* boot arg to point to the kernel Mach-O header

iOS kernel boot process

Any ideas?

iOS kernel boot process



iOS kernel boot process

- Loading the Secure Monitor image for iOS 12.1 for iPhone 6s plus in EL3 and start executing
  - Tried many different solutions such as changing the base address to the loaded Mach-O header address (above the lowest loaded section/driver)

```
vm_offset_t$
ml_static_vtop(vm_offset_t va)$
{$
>---for (size_t i = 0; (i < PTOV_TABLE_SIZE) && (ptov_table[i].len != 0); i++) {$
>--->---if ((va >= ptov_table[i].va) && (va < (ptov_table[i].va + ptov_table[i].len)))$
>--->--->---return (va - ptov_table[i].va + ptov_table[i].pa);$
>---}$
>---if (((vm_address_t)(va) - gVirtBase) >= gPhysSize)$
>--->---panic("ml_static_vtop(): illegal VA: %p\n", (void*)va);$
>---return ((vm_address_t)(va) - gVirtBase + gPhysBase);$
}$
```

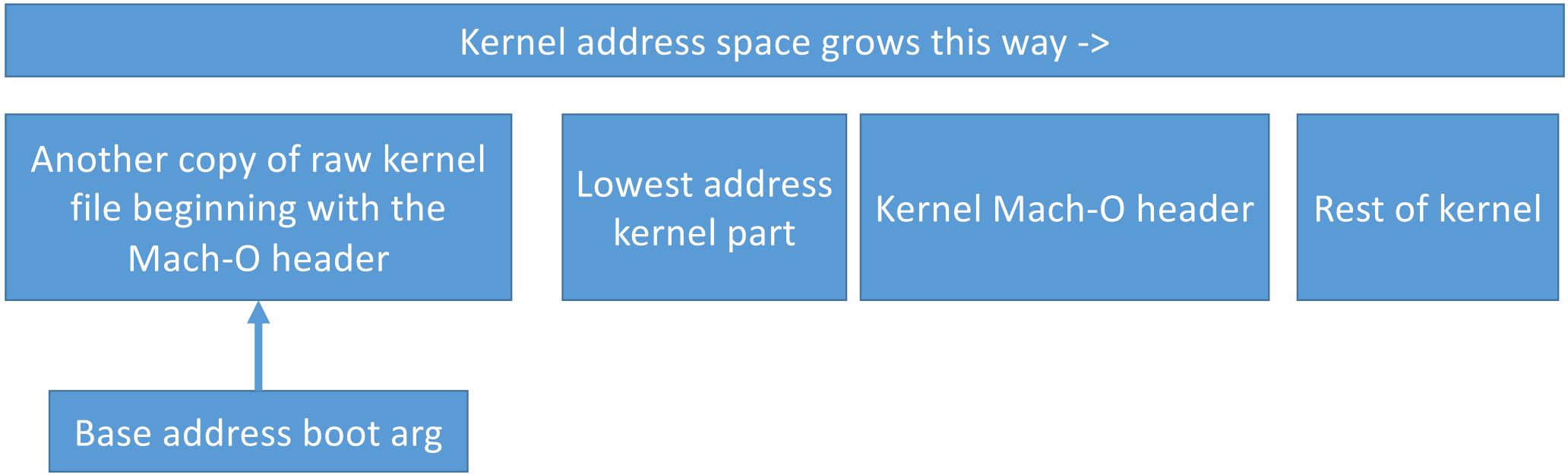
iOS kernel boot process

- The kernel needs a secure monitor to service its SMCs
- The secure monitor requires the *base address* boot arg to point to the kernel Mach-O header
- The *base address* boot arg needs to point to the lowest kernel address in order for the kernel to operate properly

iOS kernel boot process

Any ideas?

iOS kernel boot process



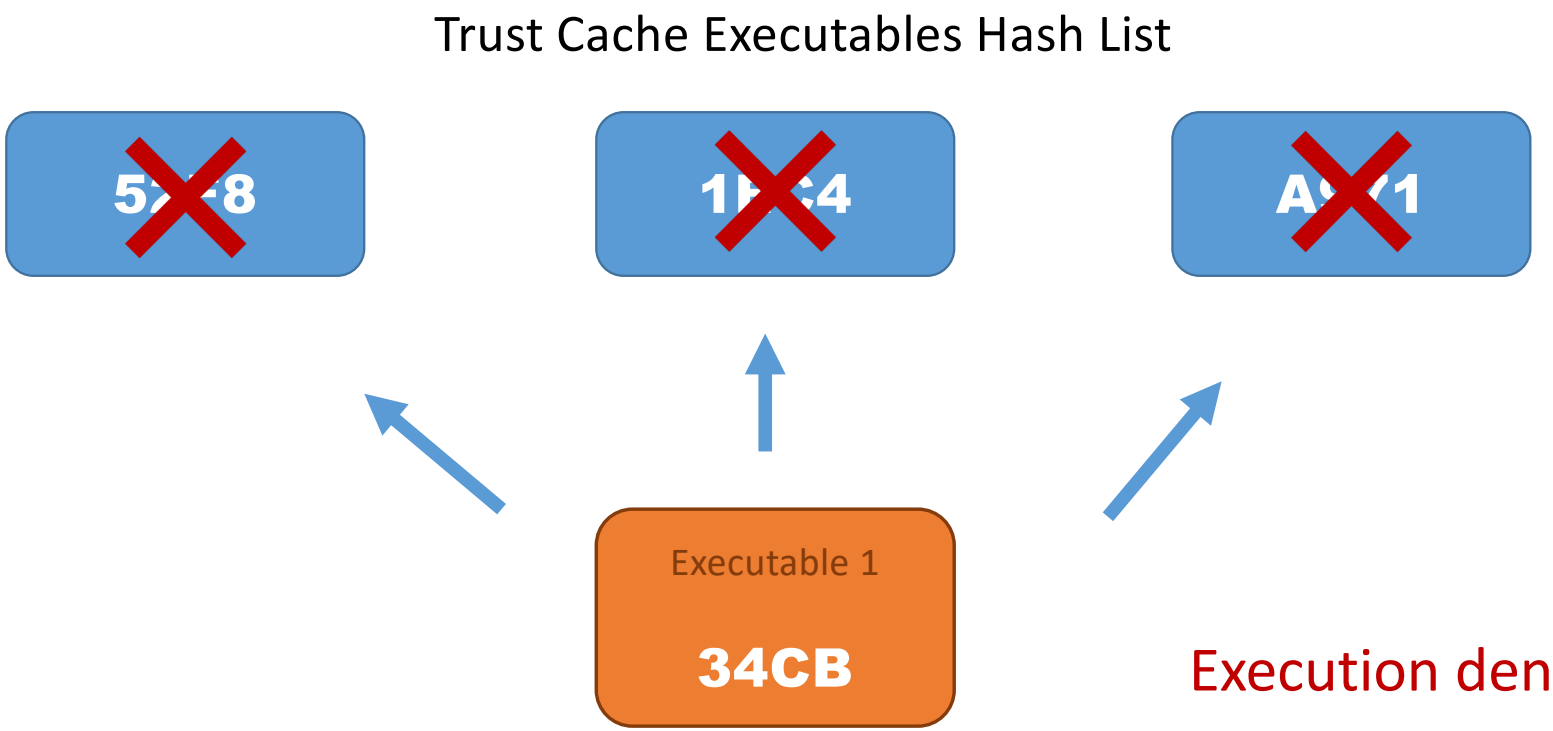
iOS kernel boot process

And it works!

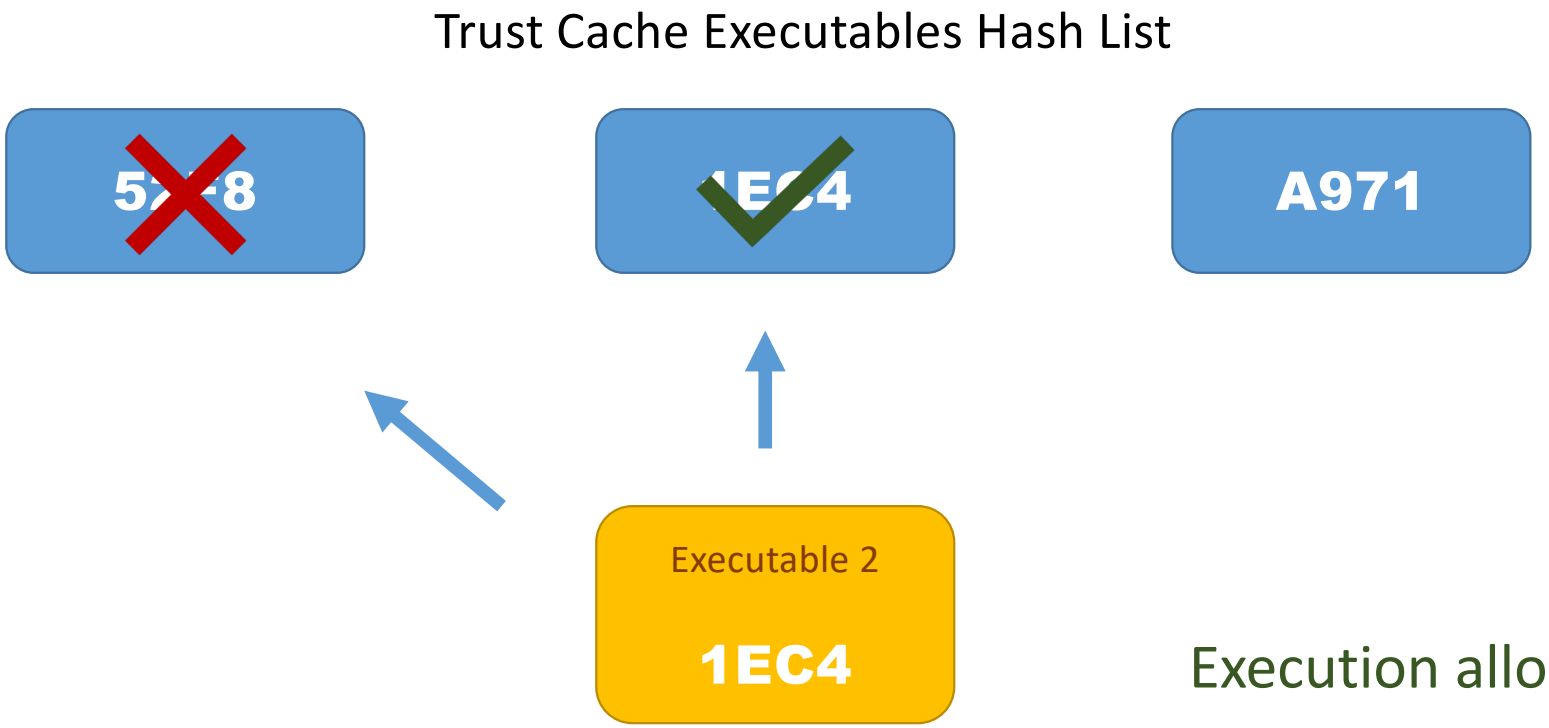
Agenda

- Past public research on iOS on QEMU
- iOS kernel boot process
- **Execution of non-apple executables with Trust Cache**
- Bash execution with launchd
- UART interactive I/O
- Next steps

Trust Cache


Trust Cache Executables Hash List
Executable 1
34CB
Execution denied!

Trust Cache


Trust Cache Executables Hash List
A971
Executable 2
1EC4
Execution allowed!

Trust Cache

- iOS has 3 different types of trust caches
  - A list of hardcoded hashes approved in the kernelcache
  - A dynamic trust cache that can be loaded at runtime from a file
  - A static trust cache in memory pointed from the device tree

Trust Cache

Device tree

From Wikipedia, the free encyclopedia

In computing, a **device tree** (also written **devicetree**) is a data structure describing the hardware components of a particular computer so that the operating system's kernel can use and manage those components, including the CPU or CPUs, the memory, the buses and the peripherals.

Trust Cache

- Top level CoreTrust validation where execution is decided

```
...ff0061e4eec e0 5b 40 f9     ldr        x0,[sp, #0xb0]
...ff0061e4ef0 fc 03 06 32     mov        w28,#0x4000000
...ff0061e4ef4 80 01 00 b4     cbz        x0,LAB_fffffff0061e4f24
...ff0061e4ef8 e1 ef 02 91     add        x1,sp,#0xbb
...ff0061e4efc e2 eb 02 91     add        x2,sp,#0xba
...ff0061e4f00 09 f8 ff 97     bl         FUN_fffffff0061e2f24
...ff0061e4f04 68 02 40 b9     ldr        w8,[x19]
...ff0061e4f08 08 01 00 2a     orr        w8,w8,w0
...ff0061e4f0c 68 02 00 b9     str        w8,[x19]
...ff0061e4f10 e8 ef 42 39     ldrb       w8,[sp, #0xbb]
...ff0061e4f14 1f 01 00 71     cmp        w8,#0x0
...ff0061e4f18 08 00 88 52     mov        w8,#0x4000
...ff0061e4f1c 08 80 a0 72     movk       w8,#0x400, LSL #16
...ff0061e4f20 9c 03 88 1a     csel       w28,w28,w8,eq

                        LAB_fffffff0061e4f24
...ff0061e4f24 e0 03 18 aa     mov        x0,x24
...ff0061e4f28 63 05 00 94     bl         _csblob_get_cdhash
...ff0061e4f2c f7 03 00 aa     mov        x23,x0
...ff0061e4f30 77 03 00 b4     cbz        x23,LAB_fffffff0061e4f9c
...ff0061e4f34 e0 03 17 aa     mov        x0,x23
...ff0061e4f38 05 f1 ff 97     bl         FUN_fffffff0061e134c
...ff0061e4f3c 60 07 00 34     cbz        w0,LAB_fffffff0061e5028
...ff0061e4f40 68 02 40 b9     ldr        w8,[x19]
...ff0061e4f44 02 01 1c 2a     orr        w2,w8,w28
...ff0061e4f48 62 02 00 b9     str        w2,[x19]
...ff0061e4f4c c3 d5 ff b0     adrp       x3,-0xffa363000
...ff0061e4f50 63 30 17 91     add        x3=>s_in-kernel_fffffff005c9d5cc,
...ff0061e4f54 e0 03 16 aa     mov        x0,x22
...ff0061e4f58 e1 03 18 aa     mov        x1,x24
...ff0061e4f5c e4 03 14 aa     mov        x4,x20
...ff0061e4f60 e5 03 15 aa     mov        x5,x21
```

```
66  plStack272 = (longlong *)0x0;
67  uVar7 = FUN_fffffff0061e2a28(&plStack272,uParm4,&uStack280);
68  if ((uVar7 & 1) == 0) {
69    FUN_fffffff0061e5e58
70              (uParm1,uParm8,uParm9,
71
72               "The signature could not be validated because AMFI could not load its er
                 for validation: %s"
73              );
74    goto LAB_fffffff0061e4f94;
75  }
76  uVar5 = 0x4000000;
77  if (plStack272 != (longlong *)0x0) {
78    uVar5 = FUN_fffffff0061e2f24(plStack272,&bStack261,&bStack262);
79    *puParm5 = *puParm5 | uVar5;
80    uVar5 = 0x4000000;
81    if (bStack261 != 0) {
82      uVar5 = 0x4004000;
83    }
84  }
85  puVar23 = (undefined *)_csblob_get_cdhash(uParm4);
86  if (puVar23 == (undefined *)0x0) {
87    pcVar17 = "Internal Error: No cdhash found.";
88LAB_fffffff0061e4fa4:
89    FUN_fffffff0061e5e58(uParm1,uParm8,uParm9,pcVar17);
90  }
91  else {
92    iVar6 = FUN_fffffff0061e134c(puVar23);
93    if (iVar6 == 0) {
94      iVar6 = FUN_fffffff0061e4990(puVar23);
95      if (iVar6 == 0) {
96LAB_fffffff0061e50e0:
97        puVar23 = (undefined *)_IOMalloc(0x1000);
```

Trust Cache

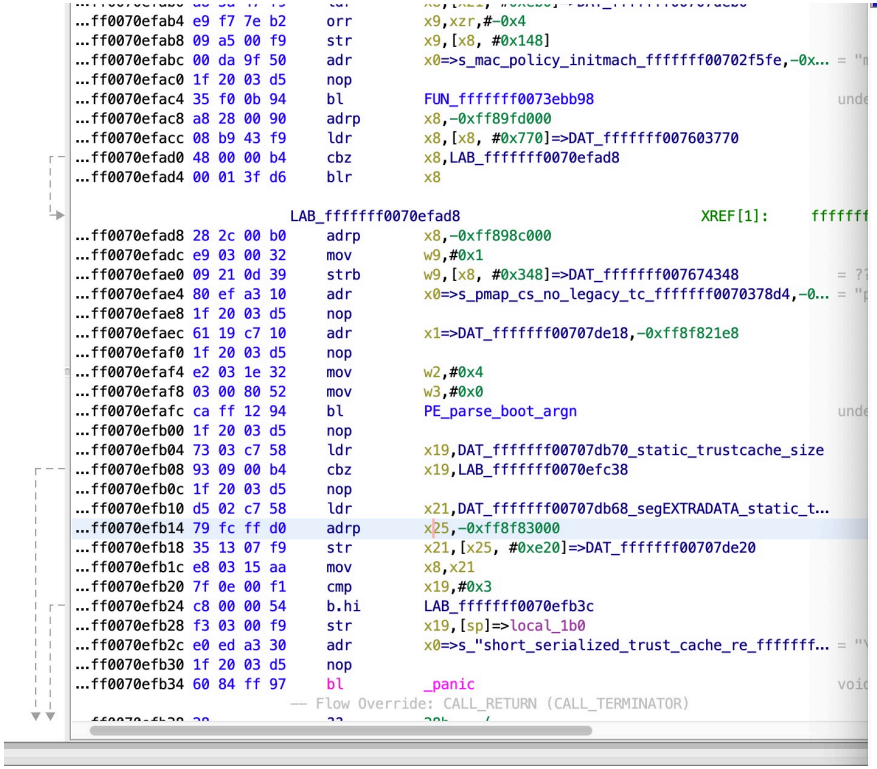
- From there dive deeper into the static trust cache lookup

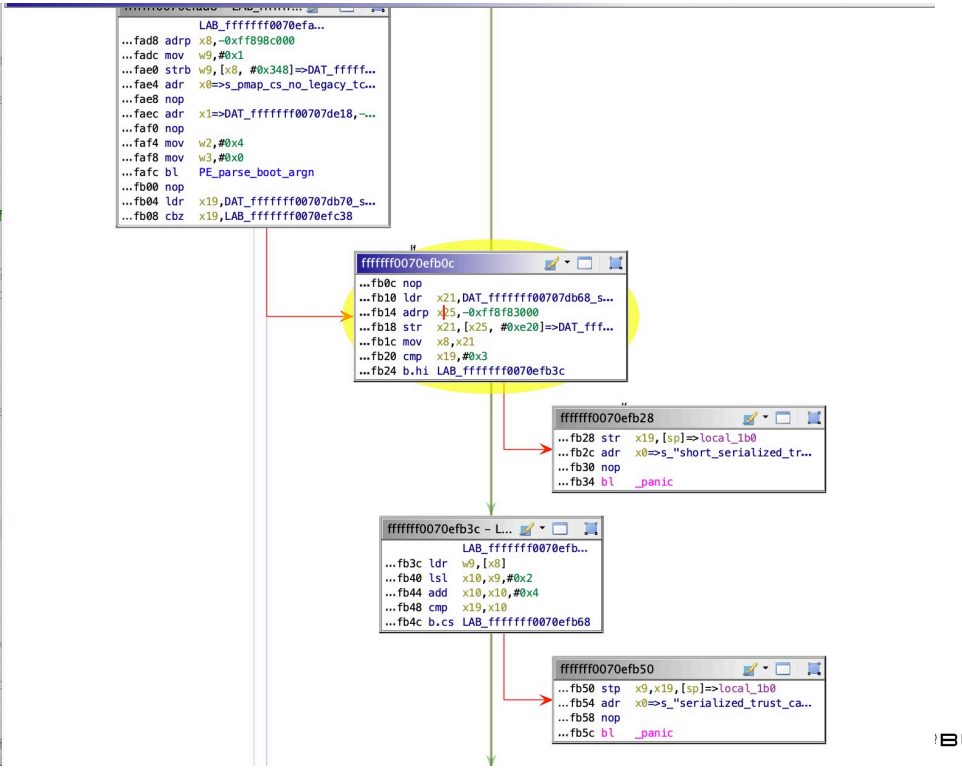
```
...ff0071b5c9c e0 07 9f 1a     cset        w0,ne
...ff0071b5ca0 04 00 00 14     b           LAB_ffffff0071b5cb0

                        LAB_ffffff0071b5ca4                             XREF[2]:     ffffffff
                                                                                     ffffffff
...ff0071b5ca4 9f fe 9f c8     stlr        xzr,[x20]=>DAT_ffffff00764bb80                  = ??
...ff0071b5ca8 85 1c 00 94     bl          FUN_ffffff0071bcebc                             unde
...ff0071b5cac e0 03 00 32     mov         w0,#0x1

                        LAB_ffffff0071b5cb0                             XREF[1]:     ffffffff
...ff0071b5cb0 fd 7b 42 a9     ldp         x29=>local_10,x30,[sp, #0x20]
...ff0071b5cb4 f4 4f 41 a9     ldp         x20,x19,[sp, #local_20]
...ff0071b5cb8 f6 57 c3 a8     ldp         x22,x21,[sp], #0x30
...ff0071b5cbc c0 03 5f d6     ret

                        *************************************************************
                        *                          FUNCTION                         *
                        *************************************************************
                        ulonglong __cdecl _pmap_lookup_in_static_trust_cache(byt...
          ulonglong           x0:8           <RETURN>
          byte *              x0:8           pbParm1_input_hash
          ulonglong           x8:8           uVar5_hash_table_part_size              XREF[1]:
          byte *              x9:8           pbVar5_pointer_to_current_part_of_hash...
          ulonglong           x11:8          uVar5_index_hash_table                  XREF[1]:
          undefined8          Stack[-0x10]:8 local_10                                XREF[2]:

          undefined8          Stack[-0x20]:8 local_20                                XREF[2]:

          undefined8          Stack[-0x30]:8 local_30                                XREF[1]:
                        _pmap_lookup_in_static_trust_cache                  XREF[3]:     Entry P
                                                                                         _pmap_l
                                                                                         _pmap_l
                                                                                         ffffffff
...ff0071b5cc0 ff c3 00 d1     sub         sp,sp,#0x30
...ff0071b5cc4 f4 4f 01 a9     stp         x20,x19,[sp, #local_20]
```

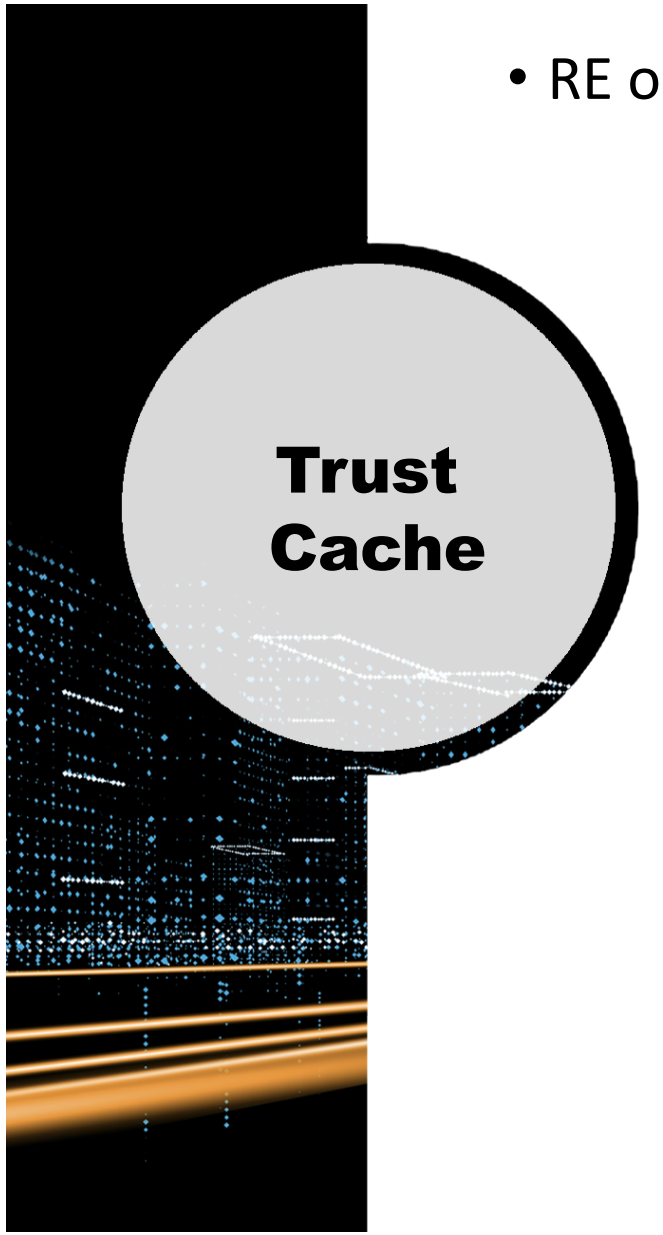
```
2  ulonglong _pmap_lookup_in_static_trust_cache(byte *pbParm1_input_hash)
3
4  {
5    longlong lVar1;
6    byte *pbVar2;
7    byte bVar3;
8    ushort uVar4;
9    ulonglong uVar5_hash_table_part_size;
10   byte *pbVar5_pointer_to_current_part_of_hash_table;
11   ulonglong uVar5;
12   ulonglong uVar5_index_hash_table;
13   ulonglong uVar6;
14   uint uVar7;
15   uint uVar8;
16
17   if (DAT_ffffff00707de28 != 0) {
18     uVar5_hash_table_part_size = (ulonglong)*(uint *)(DAT_ffffff00707de28 + 0x14);
19     if (*(uint *)(DAT_ffffff00707de28 + 0x14) != 0) {
20       pbVar5_pointer_to_current_part_of_hash_table = (byte *)(DAT_ffffff00707de28 + 0x18);
21       do {
22         uVar5_index_hash_table = uVar5_hash_table_part_size >> 1;
23         uVar7 = (uint)*pbParm1_input_hash;
24         bVar3 = pbVar5_pointer_to_current_part_of_hash_table[uVar5_index_hash_table * 0x16];
25         uVar8 = (uint)bVar3;
26         if (*pbParm1_input_hash == bVar3) {
27           uVar7 = (uint)pbParm1_input_hash[1];
28           uVar8 = (uint)pbVar5_pointer_to_current_part_of_hash_table
29                         [uVar5_index_hash_table * 0x16 + 1];
30           if (pbParm1_input_hash[1] ==
31               pbVar5_pointer_to_current_part_of_hash_table[uVar5_index_hash_table * 0x16 + 1]) {
32             uVar7 = (uint)pbParm1_input_hash[2];
33             uVar8 = (uint)pbVar5_pointer_to_current_part_of_hash_table
34                           [uVar5_index_hash_table * 0x16 + 2];
35             if (pbParm1_input_hash[2] ==
36                 pbVar5_pointer_to_current_part_of_hash_table[uVar5_index_hash_table * 0x16 + 2]) {
```

Trust Cache

- Using XREFs we can see that the static trust cache is set from here







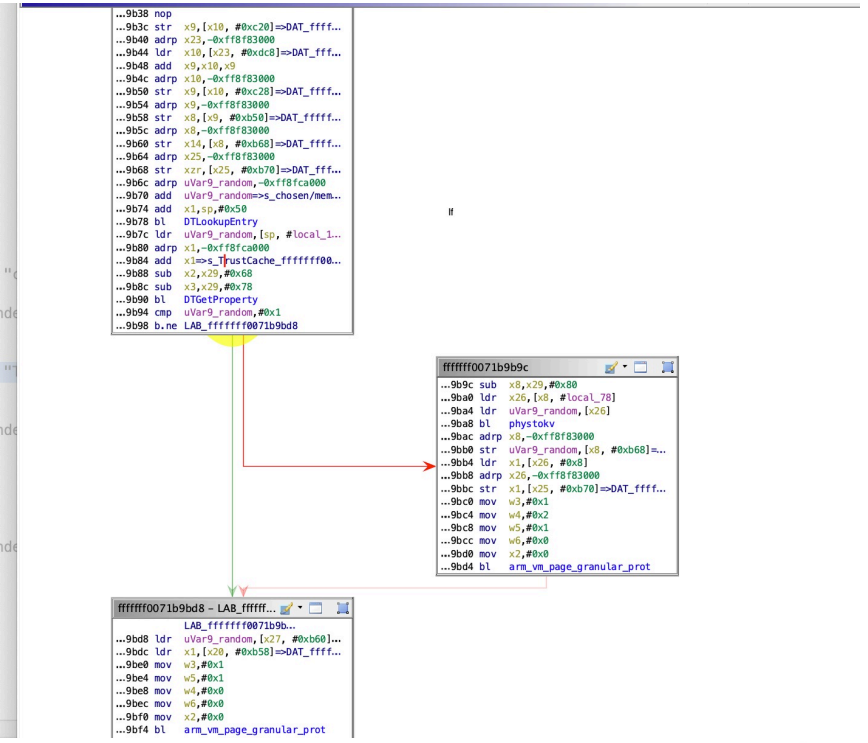
- RE on the previous function reveals this trust cache structure

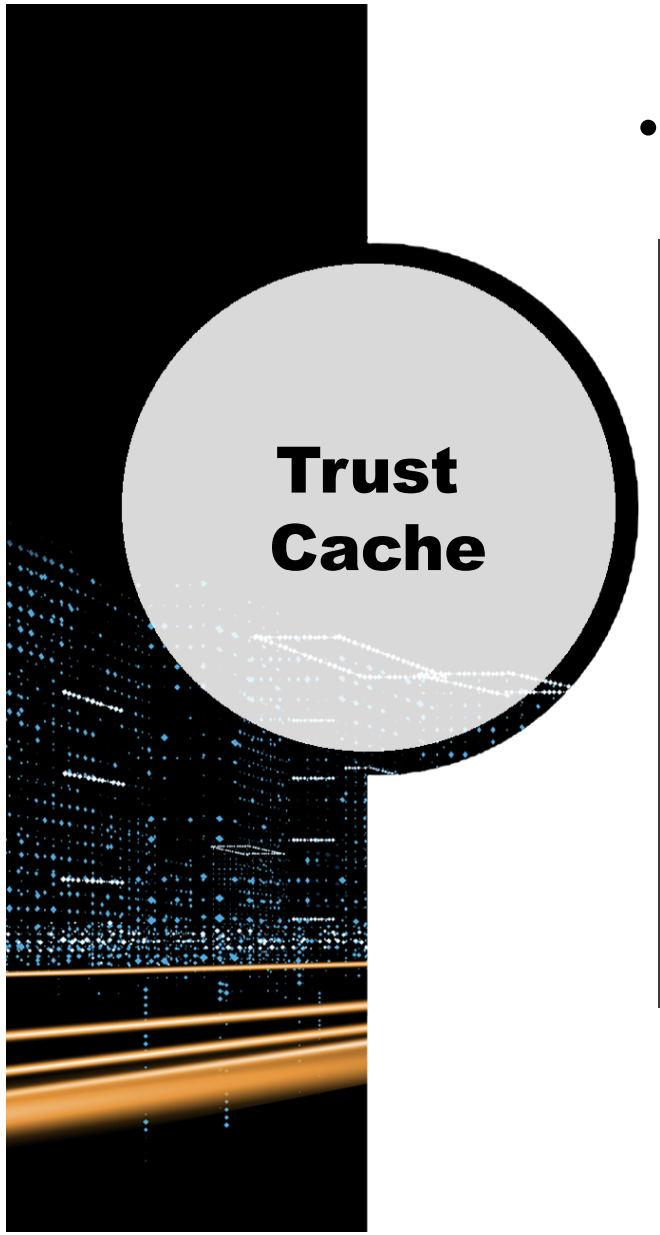
```
struct cdhash {
    uint8_t hash[20]; //first 20 bytes of the cdhash
    uint8_t hash_type; //left as 0
    uint8_t hash_flags; //left as 0
};

struct static_trust_cache_entry {
    uint64_t trust_cache_version; //should be 1
    uint64_t unknown1; //left as 0
    uint64_t unknown2; //left as 0
    uint64_t unknown3; //left as 0
    uint64_t unknown4; //left as 0
    uint64_t number_of_cdhashes;
    struct cdhash[];
};

struct static_trust_cache_buffer {
    uint64_t number_of_trust_caches_in_buffer;
    uint64_t offsets_to_trust_caches_from_beginning_of_buffer[];
    struct static_trust_cache_entry entries[];
};
```

Trust Cache

- Using XREFs we can see that this structure is read from the device tree

```
...ff0071b9b3c 49 11 06 f9     str        x9,[x10, #0xc20]=>DAT_fffffff00707dc20
...ff0071b9b40 37 f6 ff 90     adrp       x23,-0xff8f83000
...ff0071b9b44 ea e6 46 f9     ldr        x10,[x23, #0xdc8]=>DAT_fffffff00707ddc8
...ff0071b9b48 49 01 09 8b     add        x9,x10,x9
...ff0071b9b4c 2a f6 ff 90     adrp       x10,-0xff8f83000
...ff0071b9b50 49 15 06 f9     str        x9,[x10, #0xc28]=>DAT_fffffff00707dc28
...ff0071b9b54 29 f6 ff 90     adrp       x9,-0xff8f83000
...ff0071b9b58 28 a9 05 f9     str        x8,[x9, #0xb50]=>DAT_fffffff00707db50
...ff0071b9b5c 28 f6 ff 90     adrp       x8,-0xff8f83000
...ff0071b9b60 0e b5 05 f9     str        x14,[x8, #0xb68]=>DAT_fffffff00707db68_segEXTR...
...ff0071b9b64 39 f6 ff 90     adrp       x25,-0xff8f83000
...ff0071b9b68 3f bb 05 f9     str        xzr,[x25, #0xb70]=>DAT_fffffff00707db70_static...
...ff0071b9b6c e0 f3 ff b0     adrp       uVar9_random,-0xff8fca000
...ff0071b9b70 00 d0 21 91     add        uVar9_random=>s_chosen/memory-map_ffffff00703...
...ff0071b9b74 e1 43 01 91     add        x1,sp,#0x50
...ff0071b9b78 7c d6 0f 94     bl         DTLookupEntry
...ff0071b9b7c e0 2b 40 f9     ldr        uVar9_random,[sp, #local_150_physmap_base]
...ff0071b9b80 e1 f3 ff b0     adrp       x1,-0xff8fca000
...ff0071b9b84 21 18 22 91     add        x1=>s_TrustCache_fffffff007036886,x1,#0x886
...ff0071b9b88 a2 a3 01 d1     sub        x2,x29,#0x68
...ff0071b9b8c a3 e3 01 d1     sub        x3,x29,#0x78
...ff0071b9b90 6a d7 0f 94     bl         DTGetProperty
...ff0071b9b94 1f 04 00 71     cmp        uVar9_random,#0x1
...ff0071b9b98 01 02 00 54     b.ne       LAB_fffffff0071b9bd8
...ff0071b9b9c a8 03 02 d1     sub        x8,x29,#0x80
...ff0071b9ba0 1a 0d 40 f9     ldr        x26,[x8, #local_78]
...ff0071b9ba4 40 03 40 f9     ldr        uVar9_random,[x26]
...ff0071b9ba8 f8 05 00 94     bl         phystokv
...ff0071b9bac 28 f6 ff 90     adrp       x8,-0xff8f83000
...ff0071b9bb0 00 b5 05 f9     str        uVar9_random,[x8, #0xb68]=>DAT_fffffff00707db6...
...ff0071b9bb4 41 07 40 f9     ldr        x1,[x26, #0x8]
...ff0071b9bb8 3a f6 ff 90     adrp       x26,-0xff8f83000
...ff0071b9bbc 21 bb 05 f9     str        x1,[x25, #0xb70]=>DAT_fffffff00707db70_static_...
...ff0071b9bc0 e3 03 00 32     mov        w3,#0x1
...ff0071b9bc4 e4 03 1f 32     mov        w4,#0x2
...ff0071b9bc8 e5 03 00 32     mov        w5,#0x1
```





- Which Apple released the source code for

```
DTEntry memory_map;
MemoryMapFileInfo *trustCacheRange;
unsigned int trustCacheRangeSize;
int err;

err = DTLookupEntry(NULL, "chosen/memory-map", &memory_map);
assert(err == kSuccess);

err = DTGetProperty(memory_map, "TrustCache", (void**)&trustCacheRange
if (err == kSuccess) {
        assert(trustCacheRangeSize == sizeof(MemoryMapFileInfo));

        segEXTRADATA = phystokv(trustCacheRange->paddr);
        segSizeEXTRADATA = trustCacheRange->length;

        arm_vm_page_granular_RNX(segEXTRADATA, segSizeEXTRADATA, FALSE
}
```

Trust Cache

- Always works when only 1 hash in the list
- Only some items work when more than 1 item is in the list

Trust Cache

Any ideas?

Trust Cache

- Reversing this code revealed a binary search code which means the hashes are expected to be sorted in this list

```
...ff0071b5c9c e0 07 9f 1a     cset        w0,ne
...ff0071b5ca0 04 00 00 14     b           LAB_ffffff0071b5cb0

                       LAB_ffffff0071b5ca4                             XREF[2]:     ffffffff
...ff0071b5ca4 9f fe 9f c8     stlr        xzr,[x20]=>DAT_ffffff00764bb80                   = ??
...ff0071b5ca8 85 1c 00 94     bl          FUN_ffffff0071bcebc                              unde
...ff0071b5cac e0 03 00 32     mov         w0,#0x1

                       LAB_ffffff0071b5cb0                             XREF[1]:     ffffffff
...ff0071b5cb0 fd 7b 42 a9     ldp         x29=>local_10,x30,[sp, #0x20]
...ff0071b5cb4 f4 4f 41 a9     ldp         x20,x19,[sp, #local_20]
...ff0071b5cb8 f6 57 c3 a8     ldp         x22,x21,[sp], #0x30
...ff0071b5cbc c0 03 5f d6     ret

                       *************************************************************
                       *                          FUNCTION                         *
                       *************************************************************
                       ulonglong __cdecl _pmap_lookup_in_static_trust_cache(byt...
         ulonglong          x0:8           <RETURN>
         byte *             x0:8           pbParm1_input_hash
         ulonglong          x8:8           uVar5_hash_table_part_size              XREF[1]:
         byte *             x9:8           pbVar5_pointer_to_current_part_of_hash...
         ulonglong          x11:8          uVar5_index_hash_table                  XREF[1]:
         undefined8         Stack[-0x10]:8 local_10                                XREF[2]:

         undefined8         Stack[-0x20]:8 local_20                                XREF[2]:

         undefined8         Stack[-0x30]:8 local_30                                XREF[1]:
                       _pmap_lookup_in_static_trust_cache                 XREF[3]:     Entry P
                                                                                       _pmap_l
                                                                                       _pmap_l
                                                                                       ffffff
...ff0071b5cc0 ff c3 00 d1     sub         sp,sp,#0x30
...ff0071b5cc4 f4 4f 01 a9     stp         x20,x19,[sp, #local_20]
```

```
2  ulonglong _pmap_lookup_in_static_trust_cache(byte *pbParm1_input_hash)
3
4  {
5    longlong lVar1;
6    byte *pbVar2;
7    byte bVar3;
8    ushort uVar4;
9    ulonglong uVar5_hash_table_part_size;
10   byte *pbVar5_pointer_to_current_part_of_hash_table;
11   ulonglong uVar5;
12   ulonglong uVar5_index_hash_table;
13   ulonglong uVar6;
14   uint uVar7;
15   uint uVar8;
16
17   if (DAT_ffffff00707de28 != 0) {
18     uVar5_hash_table_part_size = (ulonglong)*(uint *)(DAT_ffffff00707de28 + 0x14);
19     if (*(uint *)(DAT_ffffff00707de28 + 0x14) != 0) {
20       pbVar5_pointer_to_current_part_of_hash_table = (byte *)(DAT_ffffff00707de28 + 0x18);
21       do {
22         uVar5_index_hash_table = uVar5_hash_table_part_size >> 1;
23         uVar7 = (uint)*pbParm1_input_hash;
24         bVar3 = pbVar5_pointer_to_current_part_of_hash_table[uVar5_index_hash_table * 0x16];
25         uVar8 = (uint)bVar3;
26         if (*pbParm1_input_hash == bVar3) {
27           uVar7 = (uint)pbParm1_input_hash[1];
28           uVar8 = (uint)pbVar5_pointer_to_current_part_of_hash_table
29                         [uVar5_index_hash_table * 0x16 + 1];
30           if (pbParm1_input_hash[1] ==
31               pbVar5_pointer_to_current_part_of_hash_table[uVar5_index_hash_table * 0x16 + 1]) {
32             uVar7 = (uint)pbParm1_input_hash[2];
33             uVar8 = (uint)pbVar5_pointer_to_current_part_of_hash_table
34                           [uVar5_index_hash_table * 0x16 + 2];
35             if (pbParm1_input_hash[2] ==
36                 pbVar5_pointer_to_current_part_of_hash_table[uVar5_index_hash_table * 0x16 + 2]) {
```

Trust Cache

And it works!

Agenda

- Past public research on iOS on QEMU
- iOS kernel boot process
- Execution of non-apple executables with Trust Cache
- **Bash execution with launchd**
- UART interactive I/O
- Next steps

Bash on Launchd

- Mount the RAMDisk image on OSX
- Remove all files in /System/Library/LaunchDaemons/
- Add a single file there for running bash (com.apple.bash.plist)
- Add the bash executable to the RAMDisk
- Add the bash executable hash to the Trust Cache
- Unmount the RAMDisk and run QEMU

Bash on Launchd

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.appl
<plist version="1.0">
<dict>
        <key>EnablePressuredExit</key>
        <false/>
        <key>Label</key>
        <string>com.apple.bash</string>
        <key>POSIXSpawnType</key>
        <string>Interactive</string>
        <key>ProgramArguments</key>
        <array>
                <string>/iosbinpack64/bin/bash</string>
        </array>
        <key>RunAtLoad</key>
        <true/>
        <key>StandardErrorPath</key>
        <string>/dev/console</string>
        <key>StandardInPath</key>
        <string>/dev/console</string>
        <key>StandardOutPath</key>
        <string>/dev/console</string>
        <key>Umask</key>
        <integer>0</integer>
        <key>UserName</key>
        <string>root</string>
</dict>
</plist>
```

Bash on Launchd

- System tries to execute bash
- Logs show missing libraries required for bash

Bash on Launchd

Any ideas?

Bash on Launchd

- The RAMDisk image comes without the dynamic loader cache on it, which is a file that holds most of the common runtime libs for iOS
- Copy this file into the RAMDisk at the correct path from the full disk images

Bash on Launchd

Any ideas?

Bash on Launchd

- Debug /usr/lib/dyld (the dynamic loader) which is responsible for loading the dynamic loader cache

Bash on Launchd

```
// map in shared cache to shared region
int fd = openSharedCacheFile();
if ( fd != -1 ) {
        uint8_t firstPages[8192];
        if ( ::read(fd, firstPages, 8192) == 8192 ) {
                dyld_cache_header* header = (dyld_cache_header*)firstF
#if __x86_64__
                const char* magic = (sHaswell ? ARCH_CACHE_MAGIC_H : A
#else
                const char* magic = ARCH_CACHE_MAGIC;
#endif
                if ( strcmp(header->magic, magic) == 0 ) {
                        const dyld_cache_mapping_info* const fileMappi
                        const dyld_cache_mapping_info* const fileMappi
                        shared_file_mapping_np  mappings[header->mappi
                        unsigned int mappingCount = header->mappingCou
```

Bash on Launchd

- Stepping through the execution path with gdb showed the error was in here

```
if (_shared_region_map_and_slide_np(fd, mappingCount, mappings, codeSi
        // successfully mapped cache into shared region
        sSharedCache = (dyld_cache_header*)mappings[0].sfm_address;
        sSharedCacheSlide = cacheSlide;
        dyld::gProcessInfo->sharedCacheSlide = cacheSlide;
        //dyld::log("sSharedCache=%p sSharedCacheSlide=0x%08lX\n", sSh
        // if cache has a uuid, copy it
        if ( header->mappingOffset >= 0x68 ) {
                memcpy(dyld::gProcessInfo->sharedCacheUUID, header->uu
        }
}
else {

        throw "dyld shared cache could not be mapped";

        if ( gLinkContext.verboseMapping )
                dyld::log("dyld: shared cached file could not be mappe
}
```

Bash on Launchd

- Since we have a kernel debugger in gdb we can step into the system call in the kernel

```
int
shared_region_map_and_slide_np(
        struct proc                                     *p,
        struct shared_region_map_and_slide_np_args      *uap,
        __unused int                                    *retvalp)
{
        struct shared_file_mapping_np   *mappings;
        unsigned int                    mappings_count = uap->count;
        kern_return_t                   kr = KERN_SUCCESS;
        uint32_t                        slide = uap->slide;

#define SFM_MAX_STACK   8
        struct shared_file_mapping_np   stack_mappings[SFM_MAX_STACK]

        /* Is the process chrooted?? */
        if (p->p_fd->fd_rdir != NULL) {
                kr = EINVAL;
                goto done;
        }
```

Bash on Launchd

- Stepping through this function we see that the call to _shared_region_map_and_slide() is the part that fails

```
kr = _shared_region_map_and_slide(p, uap->fd, mappings_count, mapping
                                  slide,
                                  uap->slide_start, uap->slide_size);
if (kr != KERN_SUCCESS) {
        return kr;
}


return kr;
```

- Stepping in that function reveals the error here

```
/* make sure vnode is owned by "root" */
VATTR_INIT(&va);
VATTR_WANTED(&va, va_uid);
error = vnode_getattr(vp, &va, vfs_context_current());
if (error) {
        SHARED_REGION_TRACE_ERROR(
                ("shared_region: %p [%d(%s)] map(%p:'%s'): "
                 "vnode_getattr(%p) failed (error=%d)\n",
                 (void *)VM_KERNEL_ADDRPERM(current_thread()),
                 p->p_pid, p->p_comm,
                 (void *)VM_KERNEL_ADDRPERM(vp), vp->v_name,
                 (void *)VM_KERNEL_ADDRPERM(vp), error));
        goto done;
}
if (va.va_uid != 0) {
        SHARED_REGION_TRACE_ERROR(
                ("shared_region: %p [%d(%s)] map(%p:'%s'): "
                 "owned by uid=%d instead of 0\n",
                 (void *)VM_KERNEL_ADDRPERM(current_thread()),
                 p->p_pid, p->p_comm,
                 (void *)VM_KERNEL_ADDRPERM(vp),
                 vp->v_name, va.va_uid));
        error = EPERM;
        goto done;
}
```

Bash on Launchd

- The code validates that the cache file is owned by root
- Mount the RAMDisk image in a different way to allow permission editing
- Copy the cache file and chown to root

Bash on Launchd

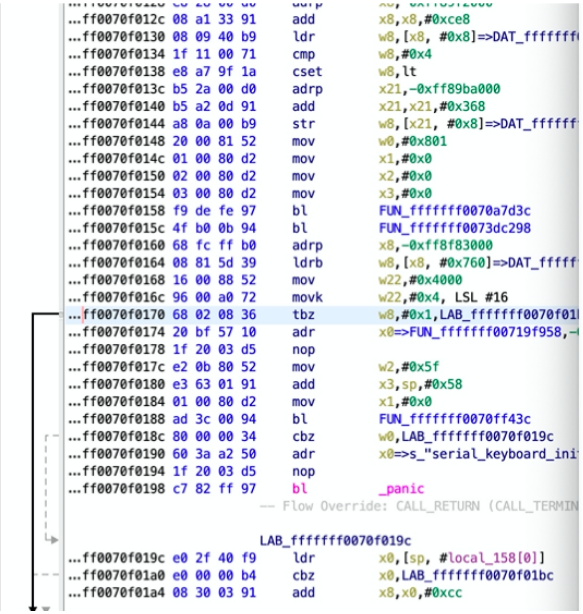
And it works!

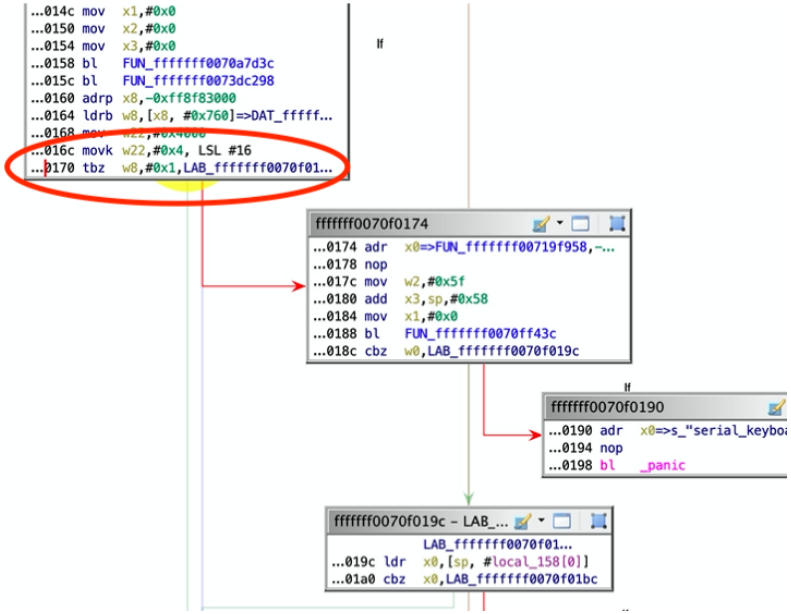
Agenda

- Past public research on iOS on QEMU
- iOS kernel boot process
- Execution of non-apple executables with Trust Cache
- Bash execution with launchd
- **UART interactive I/O**
- Next steps

Interactive UART

- UART output only was already possible with previous research
- Found where UART input is decided on in the kernel





Interactive UART

- Enabling UART input is decided based on bit #1 of a global var
- The global var is read from the “serial” kernel boot arg

```
...ff0071bad54 85 d9 0f 94     bl         _serial_init
...ff0071bad58 a8 1f 00 f0     adrp       x8,-0xff8a4f000
...ff0071bad5c 08 a1 04 91     add        x8,x8,#0x128
...ff0071bad60 09 b0 f2 10     adr        x9,-0xff8e5fca0
...ff0071bad64 1f 20 03 d5     nop
...ff0071bad68 1f 00 00 71     cmp        uVar9_random,#0x0
...ff0071bad6c 28 01 88 9a     csel       x8,x9,x8,eq
...ff0071bad70 49 22 00 b0     adrp       x9,-0xff89fd000
...ff0071bad74 28 b1 03 f9     str        x8=>_serial_putc,[x9, #0x760]=>_PE_kput
...ff0071bad78 01 f6 ff f0     adrp       x1,-0xff8f83000
...ff0071bad7c 21 80 1d 91     add        x1=>DAT_ffffff00707d760,x1,#0x760
...ff0071bad80 3f 00 00 b9     str        wzr,[x1]=>DAT_ffffff00707d760
...ff0071bad84 e0 f3 ff 90     adrp       uVar9_random,-0xff8fca000
...ff0071bad88 00 e0 20 91     add        uVar9_random=>s_serial_ffffff00703683
...ff0071bad8c e2 03 1e 32     mov        w2,#0x4
...ff0071bad90 03 00 80 52     mov        w3,#0x0
...ff0071bad94 24 d3 0f 94     bl         PE_parse_boot_argn
...ff0071bad98 40 02 00 34     cbz        uVar9_random,LAB_ffffff0071bade0
...ff0071bad9c 13 f6 ff f0     adrp       x19,-0xff8f83000
...ff0071bada0 68 62 47 b9     ldr        w8,[x19, #0x760]=>DAT_ffffff00707d760
...ff0071bada4 09 01 1e 12     and        w9,w8,#0x4
...ff0071bada8 e9 53 00 b9     str        w9,[sp, #local_150_physmap_base]
...ff0071badac 68 01 10 37     tbnz       w8,#0x2,LAB_ffffff0071badd8
...ff0071badb0 e0 f3 ff 90     adrp       uVar9_random,-0xff8fca000
...ff0071badb4 00 fc 20 91     add        uVar9_random=>s_drain_uart_sync_ffffff
...ff0071badb8 e2 03 1e 32     mov        w2,#0x4
...ff0071badbc e1 43 01 91     add        x1,sp,#0x50
...ff0071badc0 03 00 80 52     mov        w3,#0x0
...ff0071badc4 18 d3 0f 94     bl         PE_parse_boot_argn
...ff0071badc8 c0 00 00 34     cbz        uVar9_random,LAB_ffffff0071bade0
...ff0071badcc e8 53 40 b9     ldr        w8,[sp, #local_150_physmap_base]
...ff0071badd0 88 00 00 34     cbz        w8,LAB_ffffff0071bade0
```

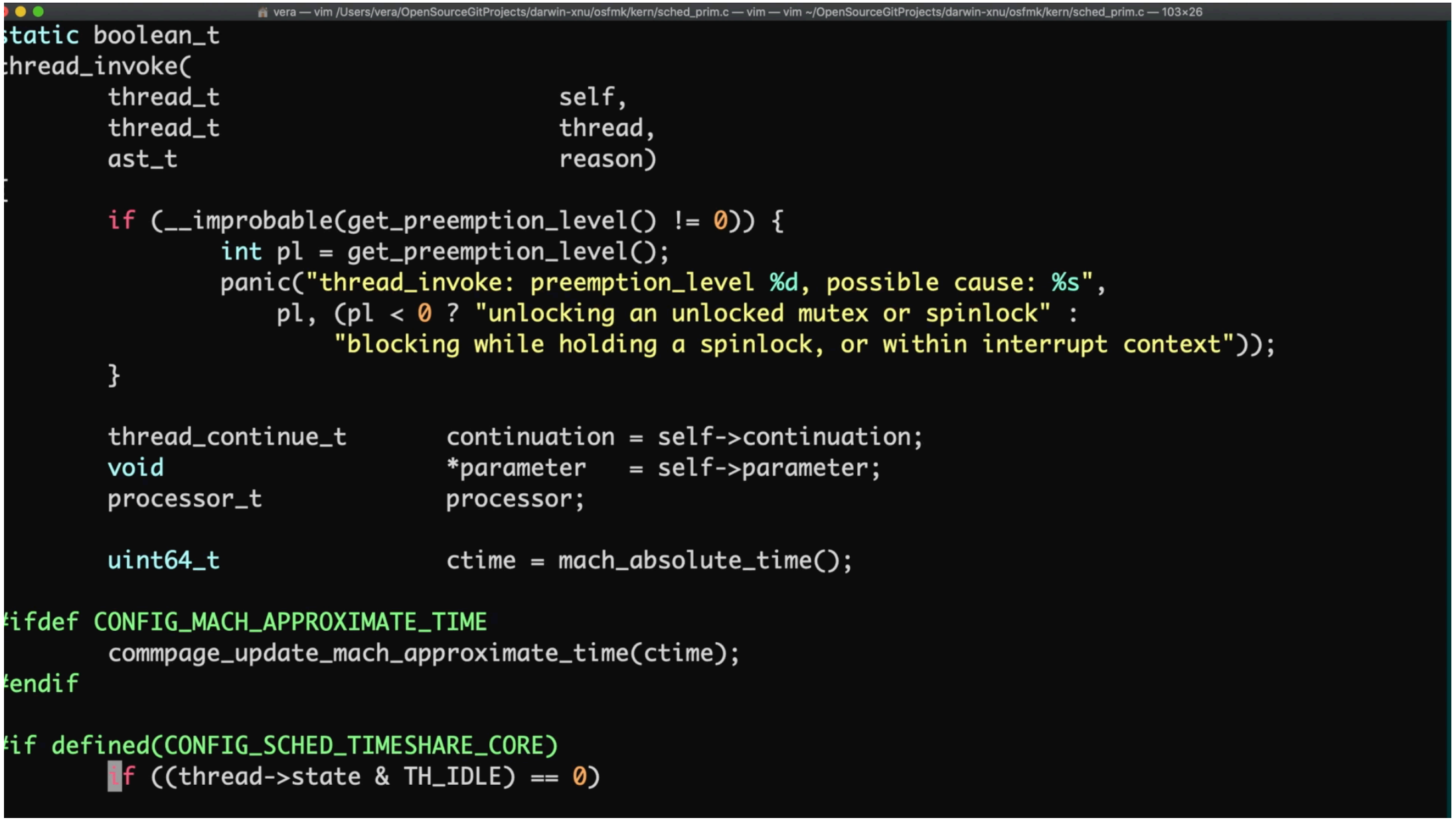
```
1012                    /* WARNING: Subroutine does not return */
1013    _panic("\"Platform Expert not initialized\"");
1014  }
1015  DAT_ffffff0076742c8 = 0x11;
1016  DAT_ffffff0076742c0 = 0;
1017  DataMemoryBarrier(2,3);
1018  iVar9 = PE_parse_boot_argn("debug",&local_150_physmap_base,4,0);
1019  if ((iVar9 != 0) && (((byte)local_150_physmap_base >> 3 & 1) != 0)) {
1020    DAT_ffffff007095cf0 = 0;
1021  }
1022  iVar9 = _serial_init();
1023  _PE_kputc = FUN_ffffff0071a0360;
1024  if (iVar9 != 0) {
1025    _PE_kputc = _serial_putc;
1026  }
1027  DAT_ffffff00707d760 = 0;
1028  iVar9 = PE_parse_boot_argn("serial",&DAT_ffffff00707d760,4,0);
1029  if ((iVar9 != 0) &&
1030     ((local_150_physmap_base =
1031            (longlong *)
1032            ((ulonglong)local_150_physmap_base & 0xffffffff00000000 |
1033            (ulonglong)_DAT_ffffff00707d760 & 0xffffffff00000004),
1034      (_DAT_ffffff00707d760 >> 2 & 1) != 0 ||
1035      ((iVar9 = PE_parse_boot_argn("drain_uart_sync",&local_150_physmap_base,4,0), iV
1036       ((uint)local_150_physmap_base != 0)))))) {
1037    _DAT_ffffff00707d760 = _DAT_ffffff00707d760 | 4;
1038  }
1039  if (DAT_ffffff007607910 == 0) {
1040    local_150_physmap_base = (longlong *)((ulonglong)local_150_physmap_base & 0xffff
1041    iVar9 = PE_parse_boot_argn("validation_disables",&local_150_physmap_base,4,0);
1042    local_150_physmap_base._0_4_ = DAT_ffffff007607910;
1043    if (iVar9 != 0) {
1044      local_150_physmap_base = (longlong *)((ulonglong)local_150_physmap_base | 1);
```

Interactive UART

- Setting the “serial” boot arg to 2

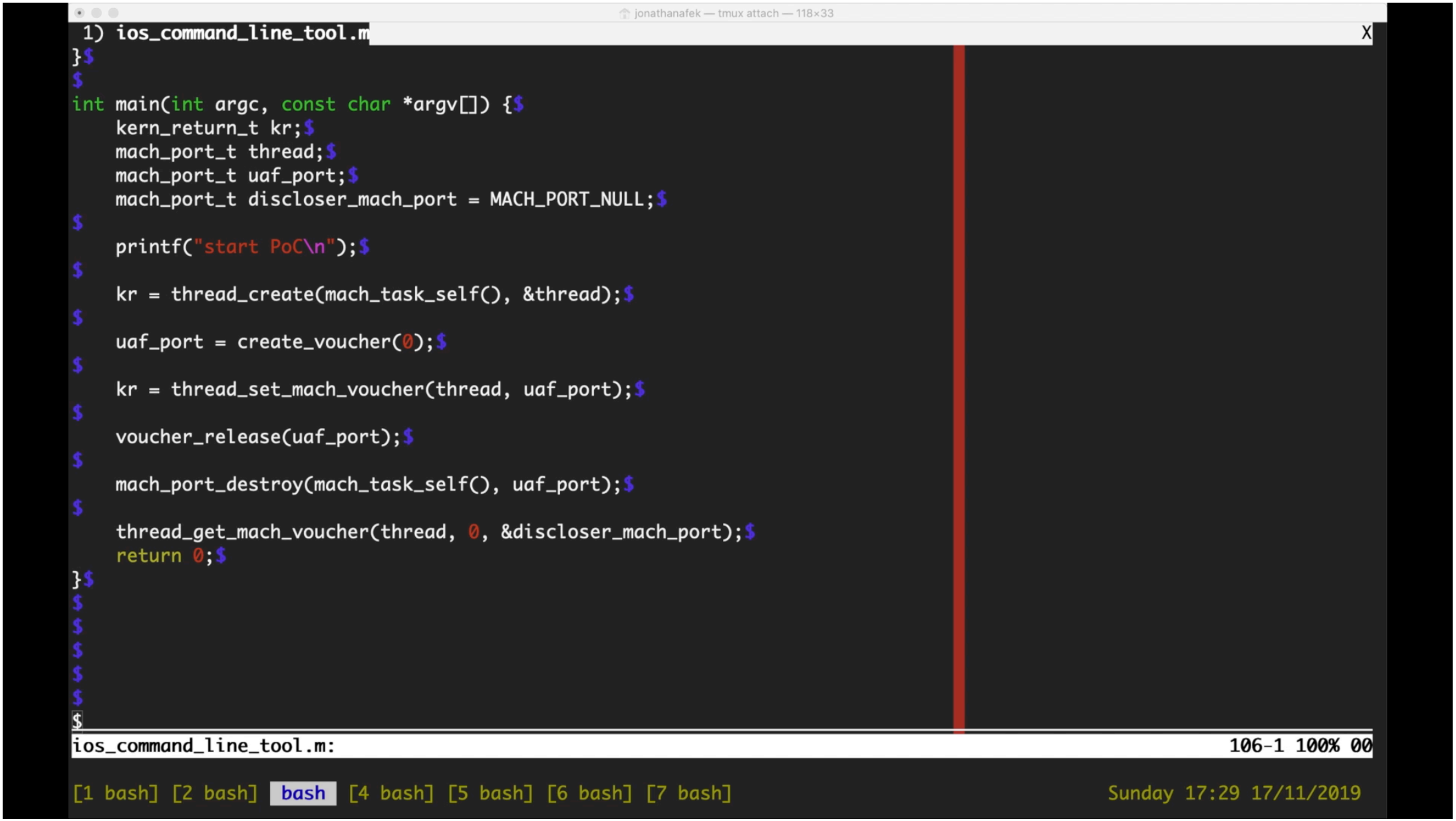
Interactive UART

And it works!


vera — vim /Users/vera/OpenSourceGitProjects/darwin-xnu/osfmk/kern/sched_prim.c — vim — vim ~/OpenSourceGitProjects/darwin-xnu/osfmk/kern/sched_prim.c — 103×26
static boolean_t
thread_invoke(
thread_t self,
thread_t thread,
ast_t reason)
if (__improbable(get_preemption_level() != 0)) {
int pl = get_preemption_level();
panic("thread_invoke: preemption_level %d, possible cause: %s",
pl, (pl < 0 ? "unlocking an unlocked mutex or spinlock" :
"blocking while holding a spinlock, or within interrupt context"));
}
thread_continue_t continuation = self->continuation;
void *parameter = self->parameter;
processor_t processor;
uint64_t ctime = mach_absolute_time();
#ifdef CONFIG_MACH_APPROXIMATE_TIME
commpage_update_mach_approximate_time(ctime);
#endif
#if defined(CONFIG_SCHED_TIMESHARE_CORE)
if ((thread->state & TH_IDLE) == 0)

Demo – Research a vulnerability – voucher_swap

- Research done by Brandon Azad
- iOS 12.1 jailbreak
- Trigger the vulnerability while debugging


jonathanafek — tmux attach — 118×33
1) ios_command_line_tool.m
}$
$
int main(int argc, const char *argv[]) {$
kern_return_t kr;$
mach_port_t thread;$
mach_port_t uaf_port;$
mach_port_t discloser_mach_port = MACH_PORT_NULL;$
$
printf("start PoC\n");$
$
kr = thread_create(mach_task_self(), &thread);$
$
uaf_port = create_voucher(0);$
$
kr = thread_set_mach_voucher(thread, uaf_port);$
$
voucher_release(uaf_port);$
$
mach_port_destroy(mach_task_self(), uaf_port);$
$
thread_get_mach_voucher(thread, 0, &discloser_mach_port);$
return 0;$
}$
ios_command_line_tool.m:
106-1 100% 00
[1 bash] [2 bash] bash [4 bash] [5 bash] [6 bash] [7 bash]
Sunday 17:29 17/11/2019

Agenda

- Past public research on iOS on QEMU
- iOS kernel boot process
- Execution of non-apple executables with Trust Cache
- Bash execution with launchd
- UART interactive I/O
- **Next steps**

Next steps and challenges

- IP communication
- Non-RAMDisk disk support
- More hardware devices (disk, screen, touch, sound, comms, etc..)
- Load all the regular iOS services in the original launchd dir instead of just bash
- More than a single CPU and an interrupt controller
- More iOS versions and devices including KTRR, PAC and other features
- More gdb scripts (allocation zones info, objects info, etc…)
- Security research

Black Hat Sound Bytes

- Use the the project and contribute! https://github.com/alephsecurity
- Check out our blog: https://alephsecurity.com
- Follow us on twitter: @alephsecurity @JonathanAfek
- Questions?

HCL AppScan

Aleph Research