
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 #BLACKHATEVENTS

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 0xffffff01b3f8cac): 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:
      0xffffff01ace5a1c\n\t  x8: 0xffffffe0005fb470  x9:  0x0000000000000001  x10: 0x00000000001071ae  x11:
      0x00000000001071ad\n\t  x12: 0x0000000000000000 x13: 0x0000000000000138  x14: 0x0000000000000000  x15:
      0x0000000000000000\n\t  x16: 0xffffff01b70a014 x17: 0x0000000000000000  x18: 0xffffff01b2e1000  x19:
      0xffffffe0064054ff\n\t  x20: 0x0000000000000003 x21: 0x0000000000000001  x22: 0xffffff01b825000  x23:
      0xffffff01b2f87e8\n\t  x24: 0x000000000000a31 x25: 0xffffffe007474000  x26: 0xffffffe0005fb470  x27:
      0xffffffe0003a5c00\n\t  x28: 0x0000000000000000 fp:  0xffffffe024a0b640  lr:  0xffffff01b2f87a0  sp:
      0xffffffe024a0b5e0\n\t  pc:  0xffffff01b301c7c 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:
      C50E403D58F345EAD8FECCF458171791\niBoot version: iBoot-4076.70.15\nsecure boot?: YES\nPaniclog version: 9\nKernel
      slide:     0x0000000014200000\nKernel text base: 0xffffff01b204000\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: 0xffffff01b3f96e8  fp:
      0xffffffe024a0abd0\n\t\t  lr: 0xffffff01b2e15f4  fp: 0xffffffe024a0abe0\n\t\t  lr: 0xffffff01b313a44  fp:
      0xffffffe024a0af50\n\t\t  lr: 0xffffff01b313dd4  fp: 0xffffffe024a0afb0\n\t\t  lr: 0xffffff01b313c00  fp:
      0xffffffe024a0afd0\n\t\t  lr: 0xffffff01b3f8cac  fp: 0xffffffe024a0b130\n\t\t  lr: 0xffffff01b3f9dfc  fp:
      0xffffffe024a0b270\n\t\t  lr: 0xffffff01b2e15f4  fp: 0xffffffe024a0b280\n\t\t  lr: 0xffffff01b301c7c  fp:
      0xffffffe024a0b640\n\t\t  lr: 0xffffff01b2f87a0  fp: 0xffffffe024a0b690\n\t\t  lr: 0xffffff01b2fffa0  fp:
      0xffffffe024a0b770\n\t\t  lr: 0xffffff01b306be8  fp: 0xffffffe024a0b850\n\t\t  lr: 0xffffff01b339988  fp:
      0xffffffe024a0b8e0\n\t\t  lr: 0xffffff01b634b84  fp: 0xffffffe024a0b950\n\t\t  lr: 0xffffff01b654448  fp:
      0xffffffe024a0b980\n\t\t  lr: 0xffffff01b654124  fp: 0xffffffe024a0bab0\n\t\t  lr: 0xffffff01b654e1c  fp:
      0xffffffe024a0bb30\n\t\t  lr: 0xffffff01b652ea8  fp: 0xffffffe024a0bc90\n\t\t  lr: 0xffffff01b2ec500  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, device tree and the kernel boot arguments were loaded in memory
- iOS RAMDisk was loaded in memory
- UART serial output was achieved
- Kernel was booted
- Launchd was 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

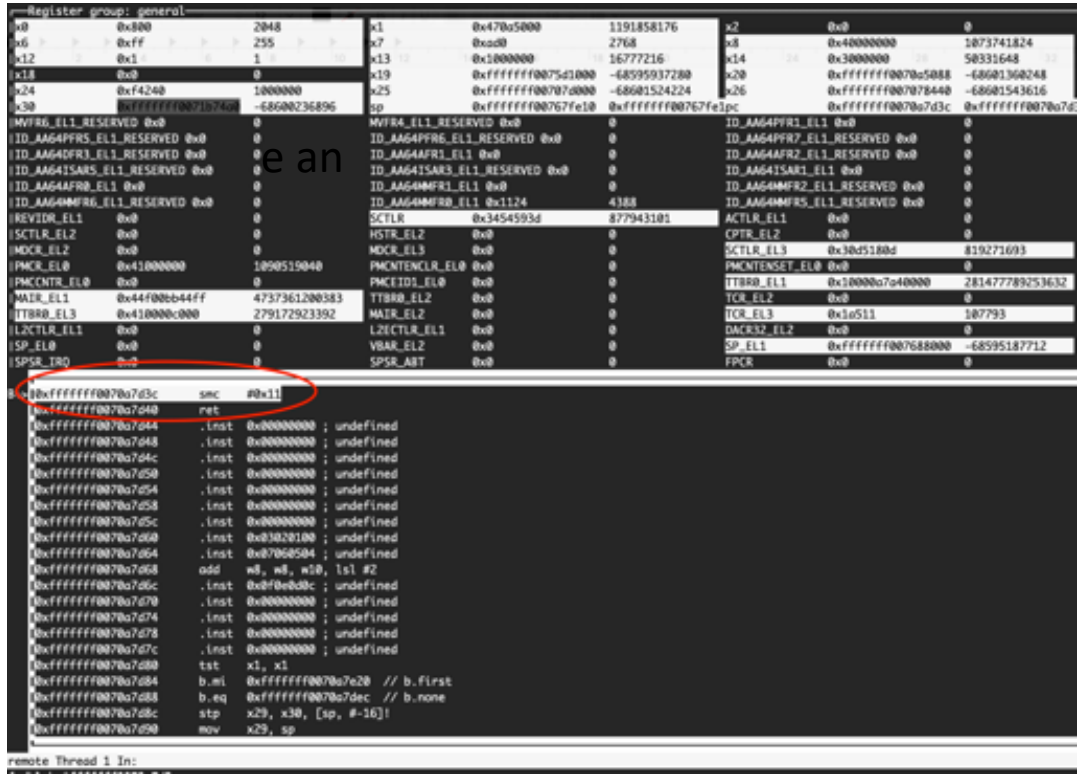
```
→ iOSonQEMU
```

Agenda

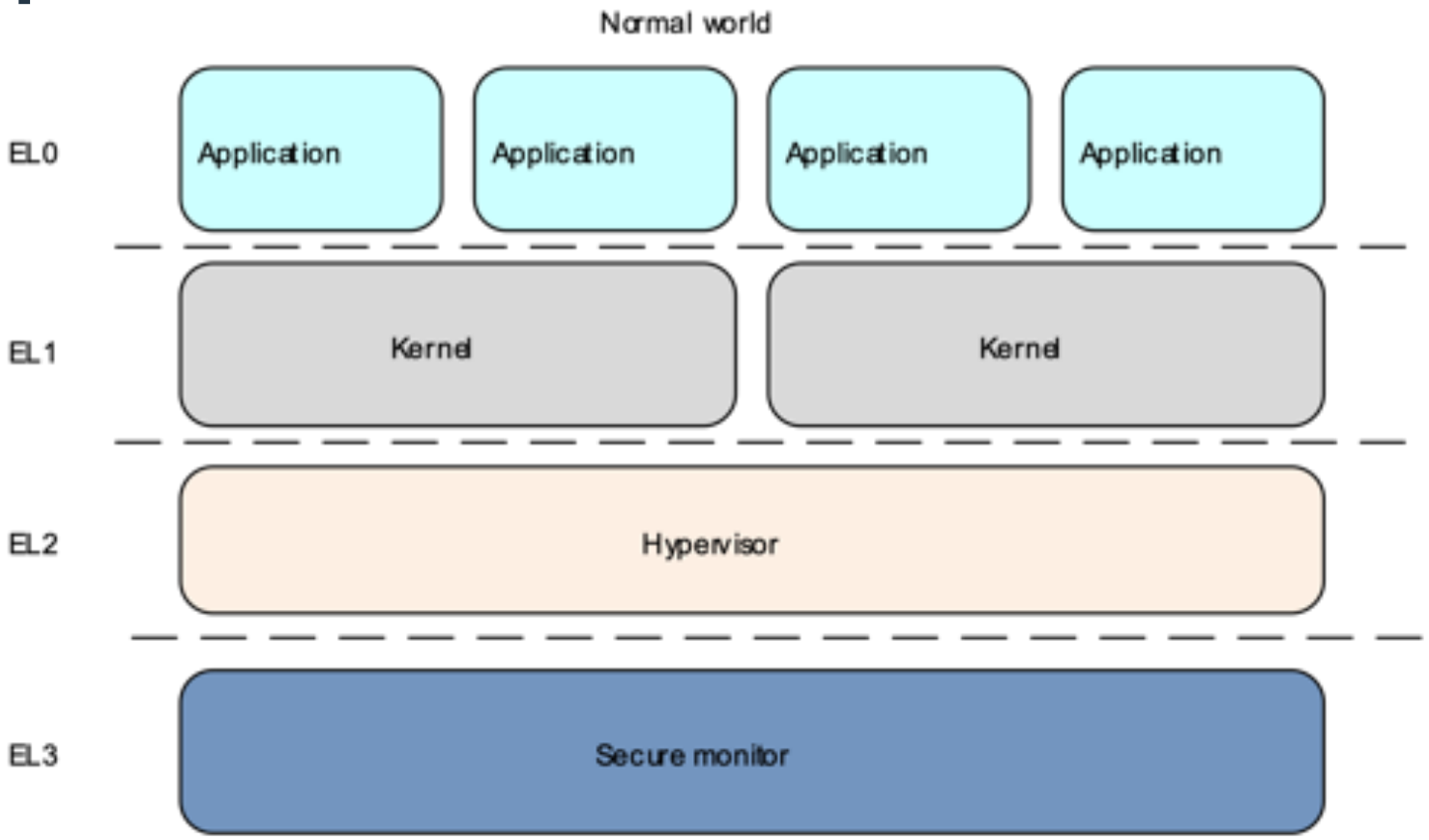
- Past public research on iOS on QEMU
- **iOS kernel boot process**
- Execution of non-apple executables with Trust Cache
- Bash execution with launchd
- UART interactive I/O
- Next steps

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 Loads 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

- The kernel needs a secure monitor to service its SMCs

iOS kernel boot process

Any ideas?

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)
- 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

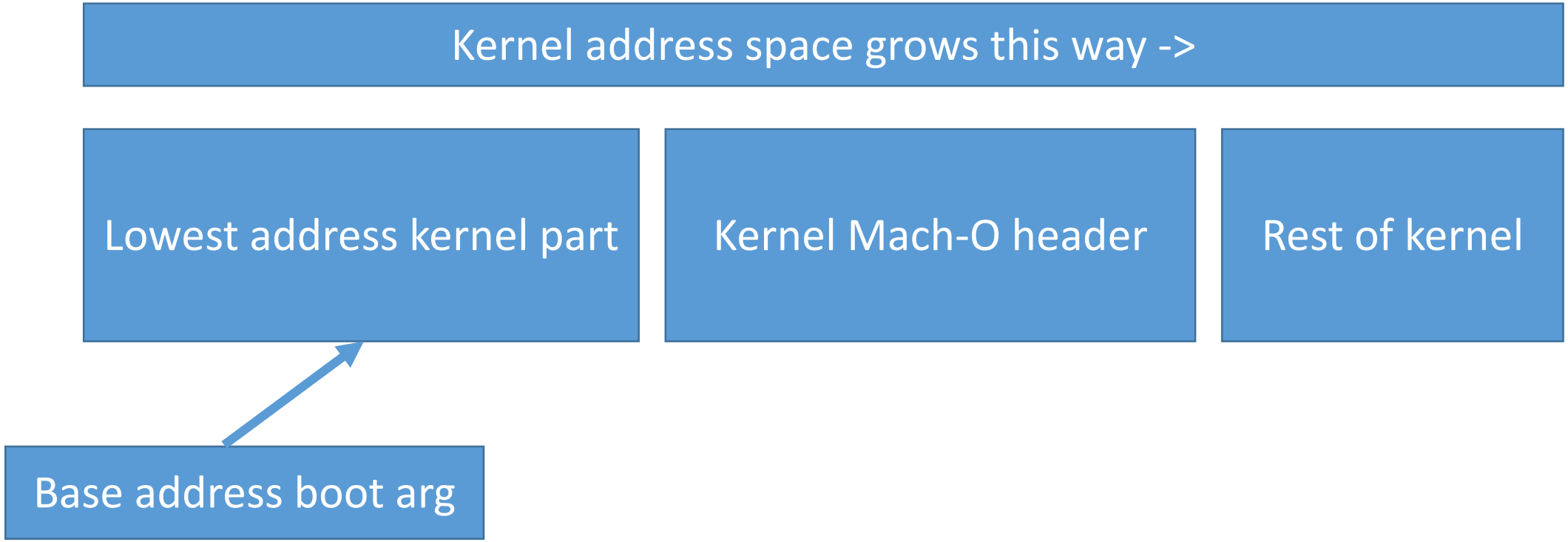
- Loading the Secure Monitor image for iOS 12.1 for iPhone 6s plus in EL3 and start executing
  - Data abort for trying to parse the kernelcache Mach-O header to decide which areas need which protections

iOS kernel boot process

```
...0410000523c e0 02 40 b9    ldr     w0,[x23]
...0410000524c 1f 65 00 71    cmp     w0,#0x19
...0410000524c 41 1a 00 54    b.ne    LAB_ffff0041000055bc
...0410000524c e0 12 40 f9    ldr     x0,[x23, #0x20]
...0410000524c 00 1a 00 b4    cbz     x0,LAB_ffff0041000055bc
...0410000524c fc 22 00 91    add     x28,x23,#0x8
...0410000524c e2 03 1c 32    mov     w2,#0x10
...0410000524c e0 03 1c aa    mov     x0,x28
...0410000524c e1 03 14 aa    mov     x1=>s___TEXT_EXEC_ffff0041000070b4,x20
...0410000524c f3 06 00 94    bl      FUN_ffff0041000006e2c
...0410000524c 40 09 00 34    cbz     w0,LAB_ffff0041000053bc
...0410000524c e2 03 1c 32    mov     w2,#0x10
...0410000524c e0 03 1c aa    mov     x0,x28
...0410000524c e1 03 15 aa    mov     x1=>s___PLK_TEXT_EXEC_ffff0041000070b0,
...0410000524c ee 06 00 94    bl      FUN_ffff0041000006e2c
...0410000524c a0 08 00 34    cbz     w0,LAB_ffff0041000053bc
...0410000524c e2 03 1c 32    mov     w2,#0x10
...0410000524c e0 03 1c aa    mov     x0,x28
...0410000524c e1 ef 00 10    adr     x1=>s___TEXT_ffff0041000070b0,-0xffbeff
...0410000524c 1f 20 03 d5    nop
...0410000524c e0 06 00 94    bl      FUN_ffff0041000006e2c
...0410000524c e0 0b 00 34    cbz     w0,LAB_ffff0041000054bc
...0410000524c e2 03 1c 32    mov     w2,#0x10
...0410000524c e0 03 1c aa    mov     x0,x28
...0410000524c 41 ef 00 70    adr     x1=>s__PRELINK_TEXT_ffff0041000070b7,-
```

```
  DAT_ffff0041000132b0 = lParm2;
  DAT_ffff0041000132b0 = DAT_ffff0041000132b0;
  DAT_ffff0041000132b0 = DAT_ffff0041000132b0;
LAB_ffff0041000523c:
  ppuVar8 = DAT_ffff0041000132b0;
  if ((*piVar19 == 0x19) && (ppuVar8 = DAT_ffff0041000132b0, *(longlong *)(piVar
    piVar1 = piVar19 + 2;
    iVar6 = FUN_ffff0041000006e2c(piVar1,"__TEXT_EXEC",0x10);
    if ((iVar6 == 0) || (iVar6 = FUN_ffff0041000006e2c(piVar1,"__PLK_TEXT_EXEC",0
      ppuVar12 = DAT_ffff0041000132b0;
      puVar7 = *(ulonglong **)(piVar19 + 6);
      puVar10 = *(ulonglong **)(piVar19 + 8);
      ppuVar8 = DAT_ffff0041000132b0 + 1;
      DAT_ffff0041000132b0 = DAT_ffff0041000132b0 + 4;
      *ppuVar8 = puVar7;
      ppuVar12[2] = puVar10;
      ppuVar12[3] = (ulonglong *)0x2;
      if (puVar7 < DAT_ffff0041000132b8) {
        DAT_ffff0041000132b8 = puVar7;
      }
      uVar11 = ((longlong)puVar10 + (longlong)puVar7) - (longlong)DAT_ffff0041000
      if (DAT_ffff0041000132b8 < uVar11) {
        DAT_ffff0041000132b8 = uVar11;
      }
      if (DAT_ffff0041000132b0 == (ulonglong **)0x0) {
        *ppuVar12 = (ulonglong *)0x0;
LAB_ffff0041000054a0:
        ppuVar8 = (ulonglong **)&DAT_ffff0041000132b0;
      }
      else {
        ppuVar8 = (ulonglong **)0x0;
        ppuVar5 = DAT_ffff0041000132b0;
        do {
```

iOS kernel boot process



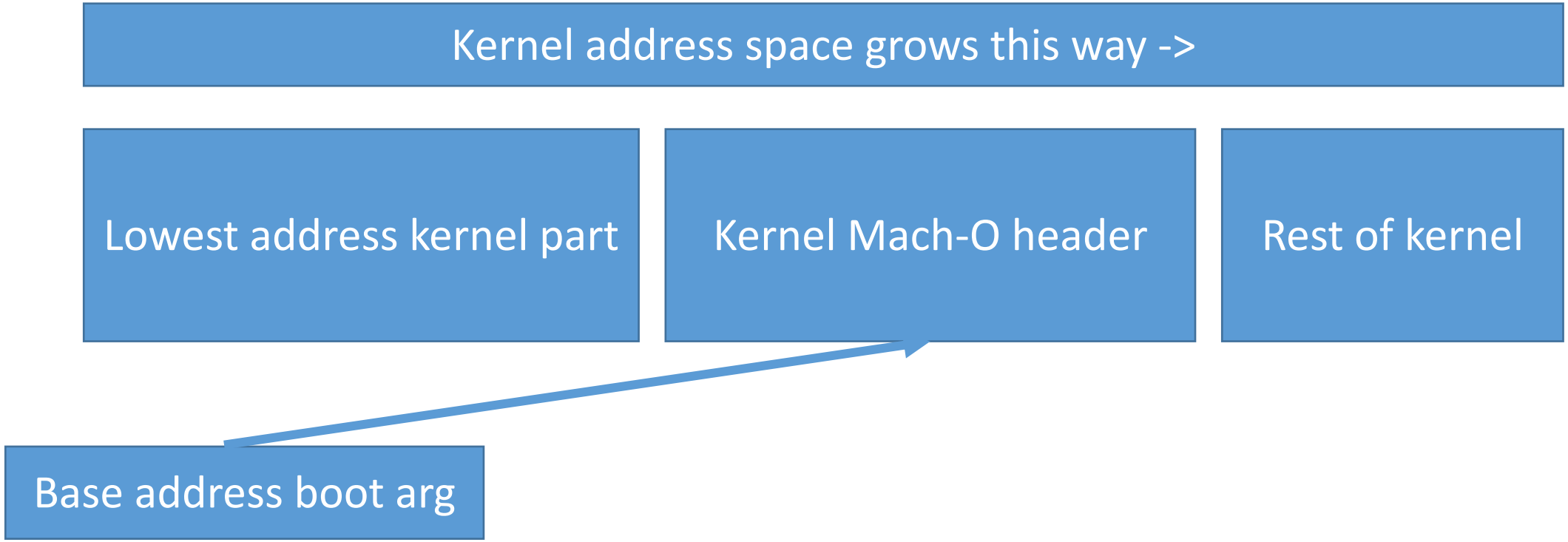
iOS kernel boot process

- The kernel needs a secure monitor to service its SMCs
- The secure monitor requires the *base address* boot arg to point to the kernel Macho-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);$
}$
```

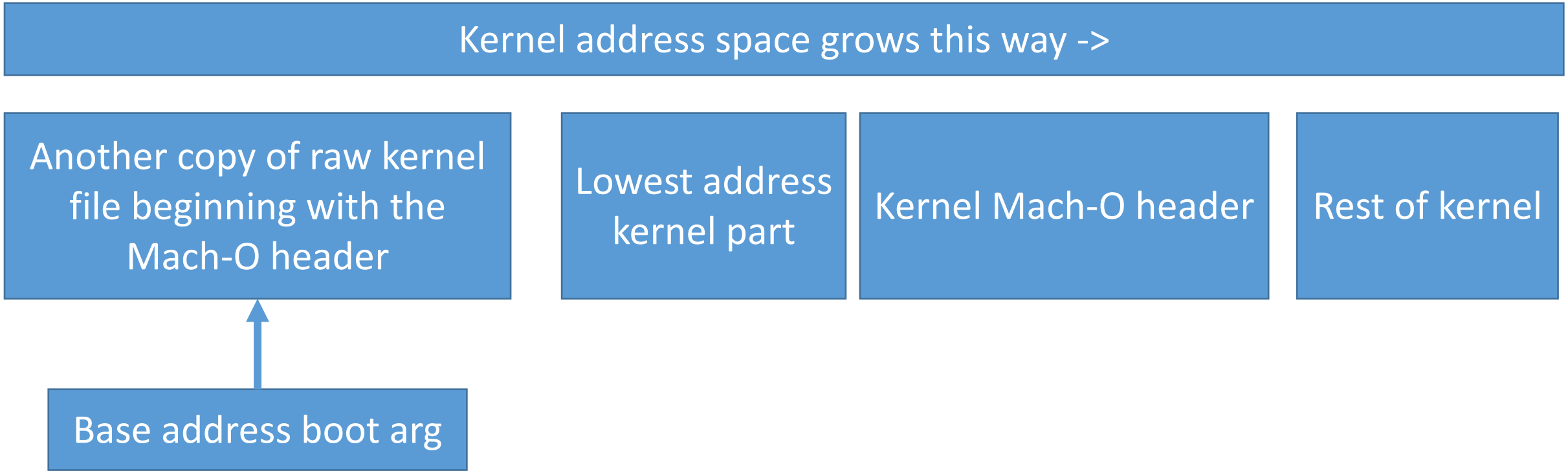
iOS kernel boot process

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

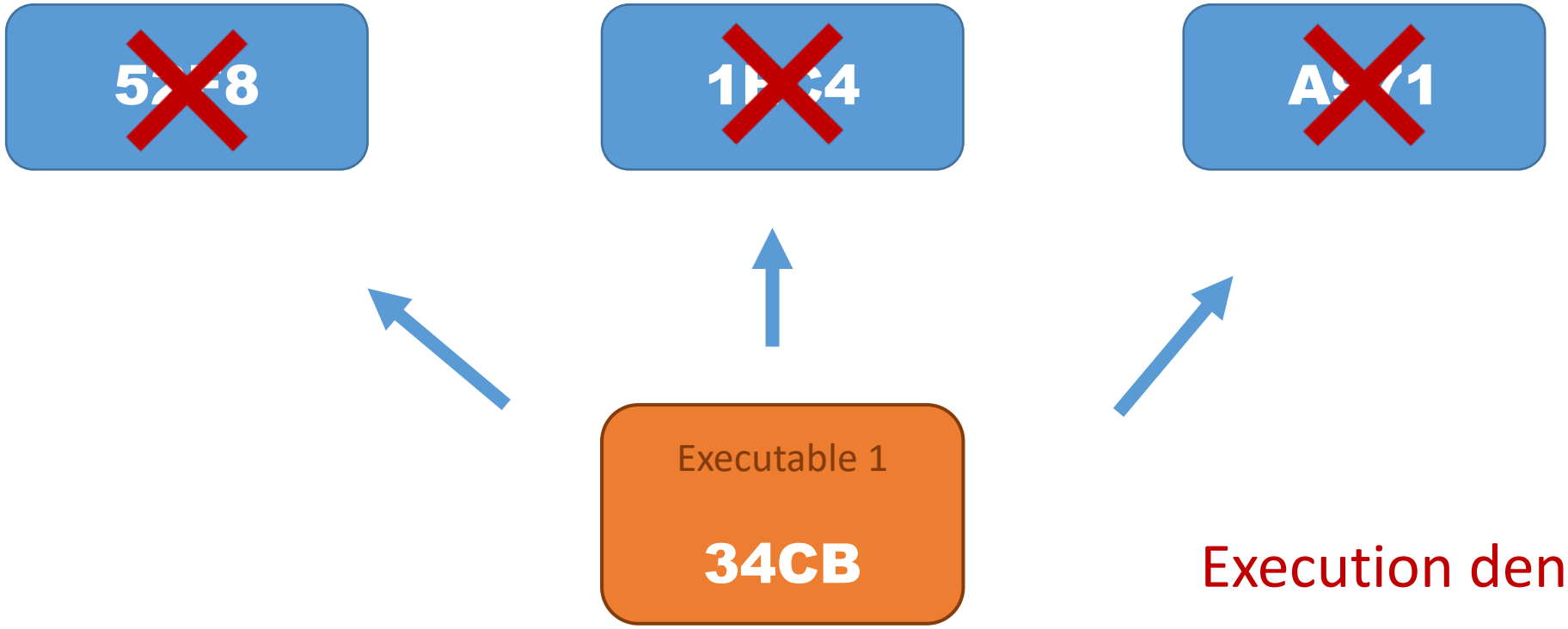
And it works!

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

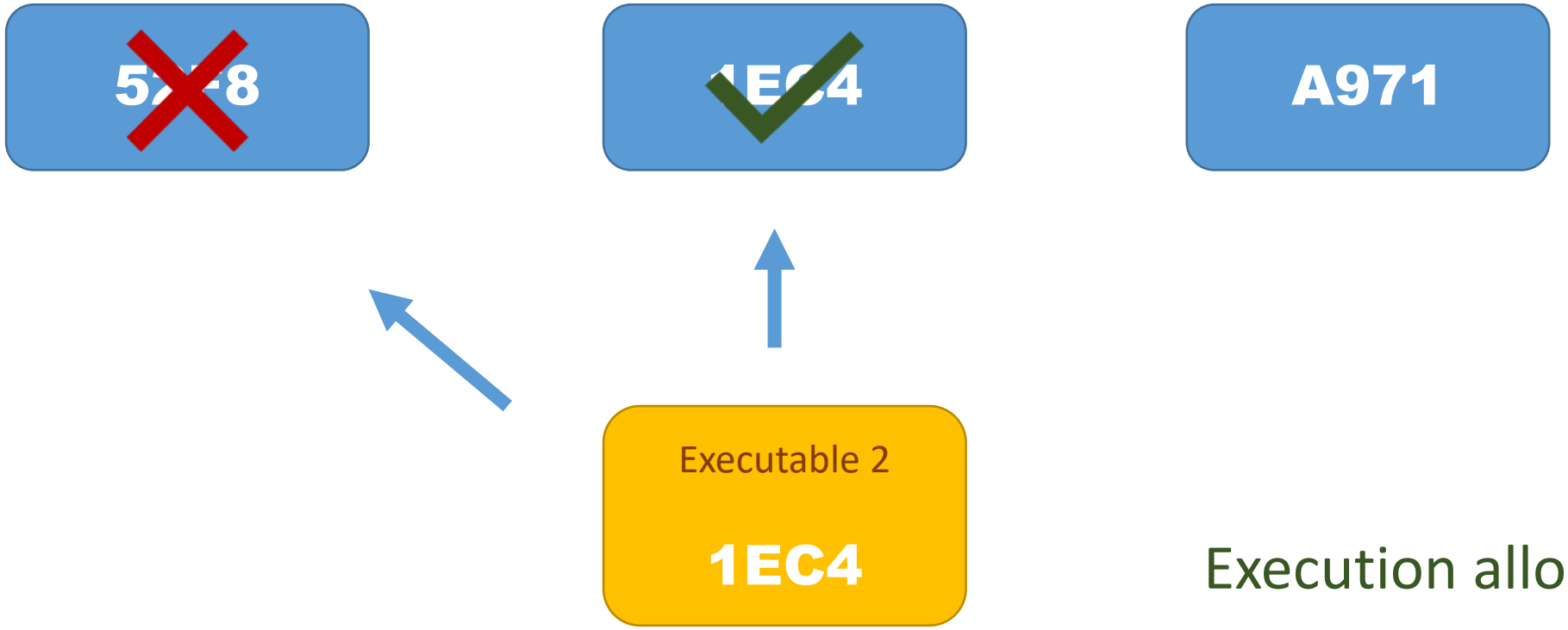
Trust Cache Executables Hash List



Execution denied!

Trust Cache

Trust Cache Executables Hash List



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

- Top level CoreTrust validation where execution is decided

```
...ff0063e4eec e8 5b 40 f9    ldr     x8,[sp, #0x60]
...ff0063e4ef0 fc 03 06 32    mov     w28,#0x4000000
...ff0063e4ef4 88 01 00 b4    cbz     x8,LAB_ffffff0061e4f24
...ff0063e4ef8 e1 ef 02 91    add     x1,sp,#0xbb
...ff0063e4efc e2 eb 02 91    add     x2,sp,#0xba
...ff0063e4f00 09 f8 ff 97    bl      FUN_ffffff0061e2f24
...ff0063e4f04 68 02 40 b9    ldr     w8,[x19]
...ff0063e4f08 08 01 00 2a    orr     w8,w8,w0
...ff0063e4f0c 68 02 00 b9    str     w8,[x19]
...ff0063e4f10 e8 ef 42 39    ldrb    w8,[sp, #0xbb]
...ff0063e4f14 1f 01 00 71    cmp     w8,#0x0
...ff0063e4f18 08 00 88 52    mov     w8,#0x4000
...ff0063e4f1c 08 50 a0 72    movk    w8,#0x400, LSL #16
...ff0063e4f20 9c 03 88 1a    csel    w28,w28,w8,eq

                    LAB_ffffff0061e4f24
...ff0063e4f24 e0 03 18 aa    mov     x0,x24
...ff0063e4f28 63 05 00 94    bl      _csblob_get_cdhash
...ff0063e4f2c f7 03 00 aa    mov     x23,x0
...ff0063e4f30 77 03 00 b4    cbz     x23,LAB_ffffff0061e4f9c
...ff0063e4f34 e0 03 17 aa    mov     x0,x23
...ff0063e4f38 05 f1 ff 97    bl      FUN_ffffff0061e134c
...ff0063e4f3c 60 07 00 34    cbz     w0,LAB_ffffff0061e5028
...ff0063e4f40 68 02 40 b9    ldr     w8,[x19]
...ff0063e4f44 02 01 1c 2a    orr     w2,w8,w28
...ff0063e4f48 62 02 00 b9    str     w2,[x19]
...ff0063e4f4c c3 d5 ff b0    adrp    x3,-0xffa363000
...ff0063e4f50 63 30 17 91    add     x3=>s_in-kernel_ffffff005c9d5cc,
...ff0063e4f54 e0 03 16 aa    mov     x0,x22
...ff0063e4f58 e1 03 18 aa    mov     x1,x24
...ff0063e4f5c e4 03 14 aa    mov     x4,x20
...ff0063e4f60 e5 03 15 aa    mov     x5,x21
```

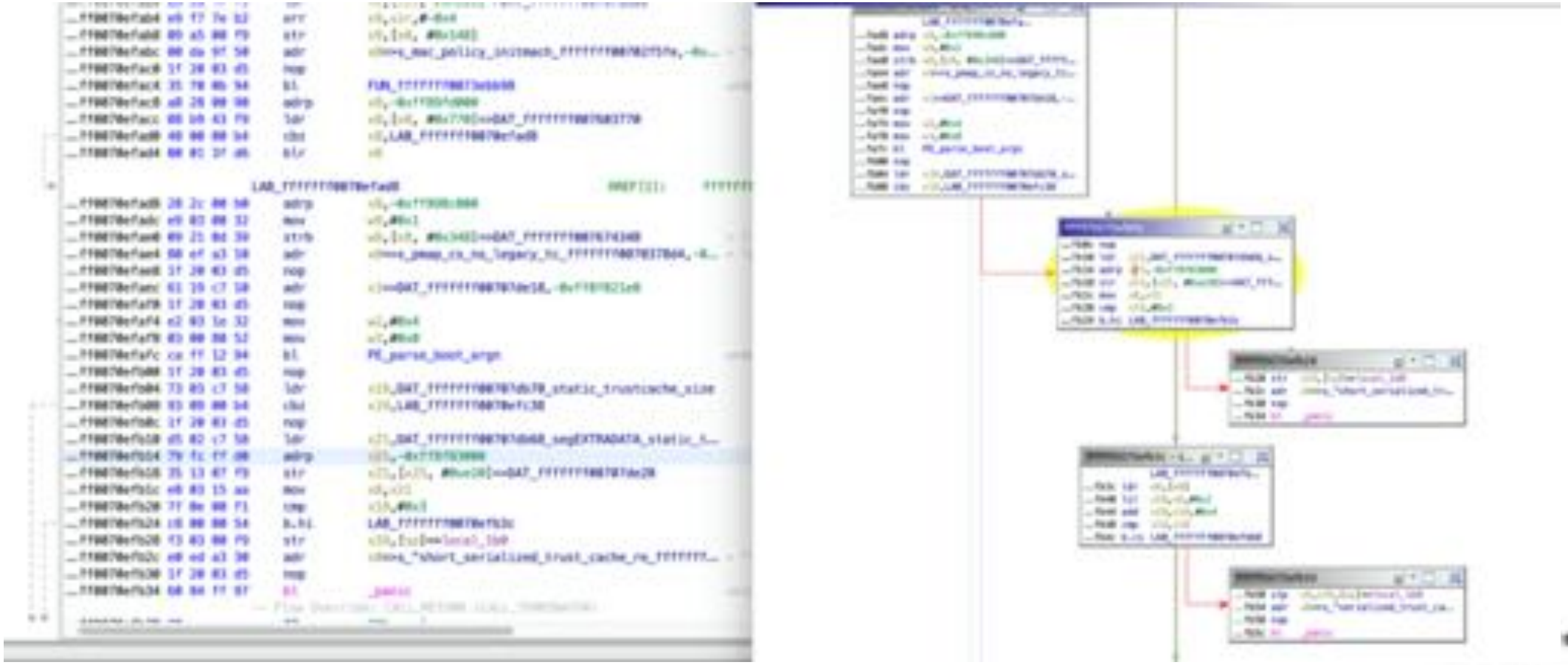
```
  plStack272 = (longlong *)0x0;
  uVar7 = FUN_ffffff0061e2a20(&plStack272,uParm4,&uStack280);
  if ((uVar7 & 1) == 0) {
    FUN_ffffff0061e5e58
              (uParm1,uParm8,uParm9,

               "The signature could not be validated because AMFI could not load its e
               for validation: %s"
              );
    goto LAB_ffffff0061e4f94;
  }
  uVar5 = 0x4000000;
  if (plStack272 != (longlong *)0x0) {
    uVar5 = FUN_ffffff0061e2f24(plStack272,&bStack281,&bStack282);
    *puParm5 = *puParm5 | uVar5;
    uVar5 = 0x4000000;
    if (bStack281 != 0) {
      uVar5 = 0x4004000;
    }
  }
  puVar23 = (undefined *)_csblob_get_cdhash(uParm4);
  if (puVar23 == (undefined *)0x0) {
    pcVar17 = "Internal Error: No cdhash found.";
LAB_ffffff0061e4fa4:
    FUN_ffffff0061e5e58(uParm1,uParm8,uParm9,pcVar17);
  }
  else {
    iVar6 = FUN_ffffff0061e134c(puVar23);
    if (iVar6 == 0) {
      iVar6 = FUN_ffffff0061e4990(puVar23);
      if (iVar6 == 0) {
LAB_ffffff0061e50e0:
        puVar23 = (undefined *)_IOMalloc(0x1000);
```

Trust Cache

- From there dive deeper into the static trust cache lookup

Trust Cache

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



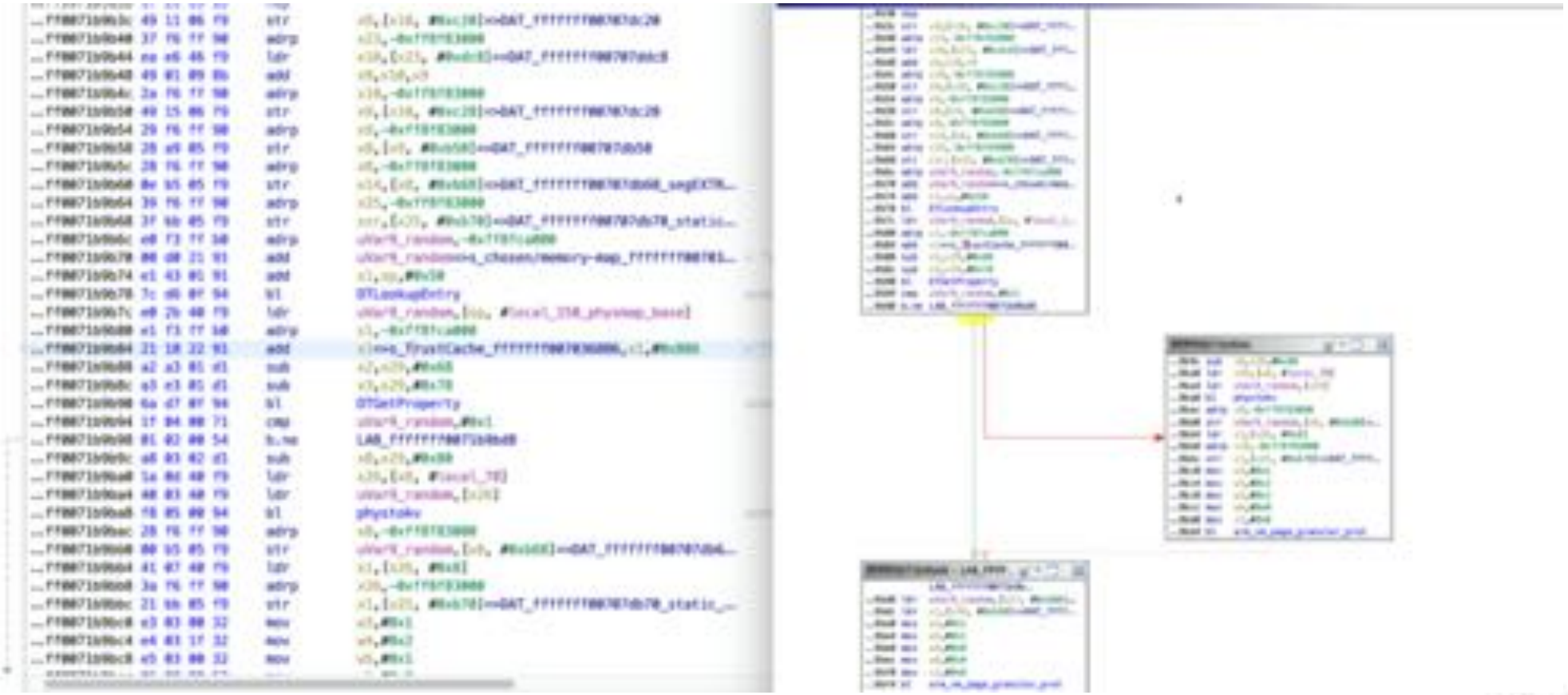
Trust Cache

- 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



Trust Cache

- 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

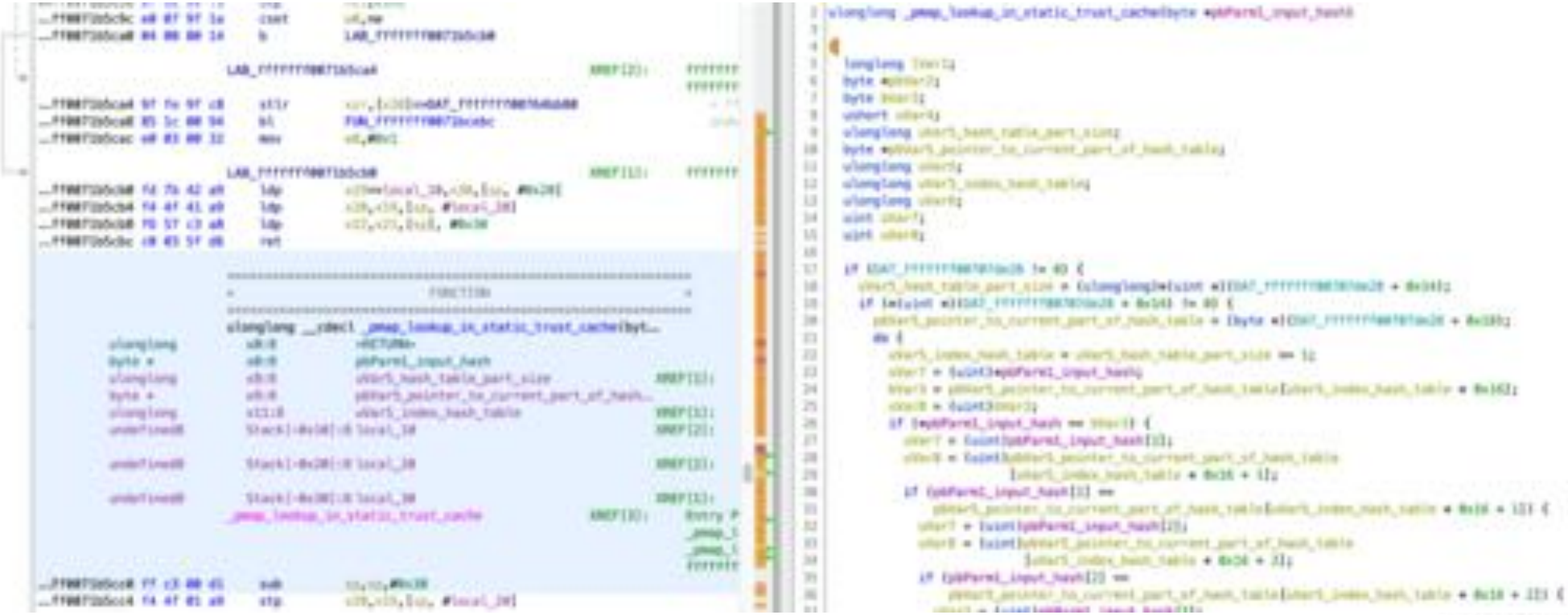
- Always works when only 1 hash in the list
- Only some items work when more than 1 item is in the list

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



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 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;
```

Bash on Launchd

- 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

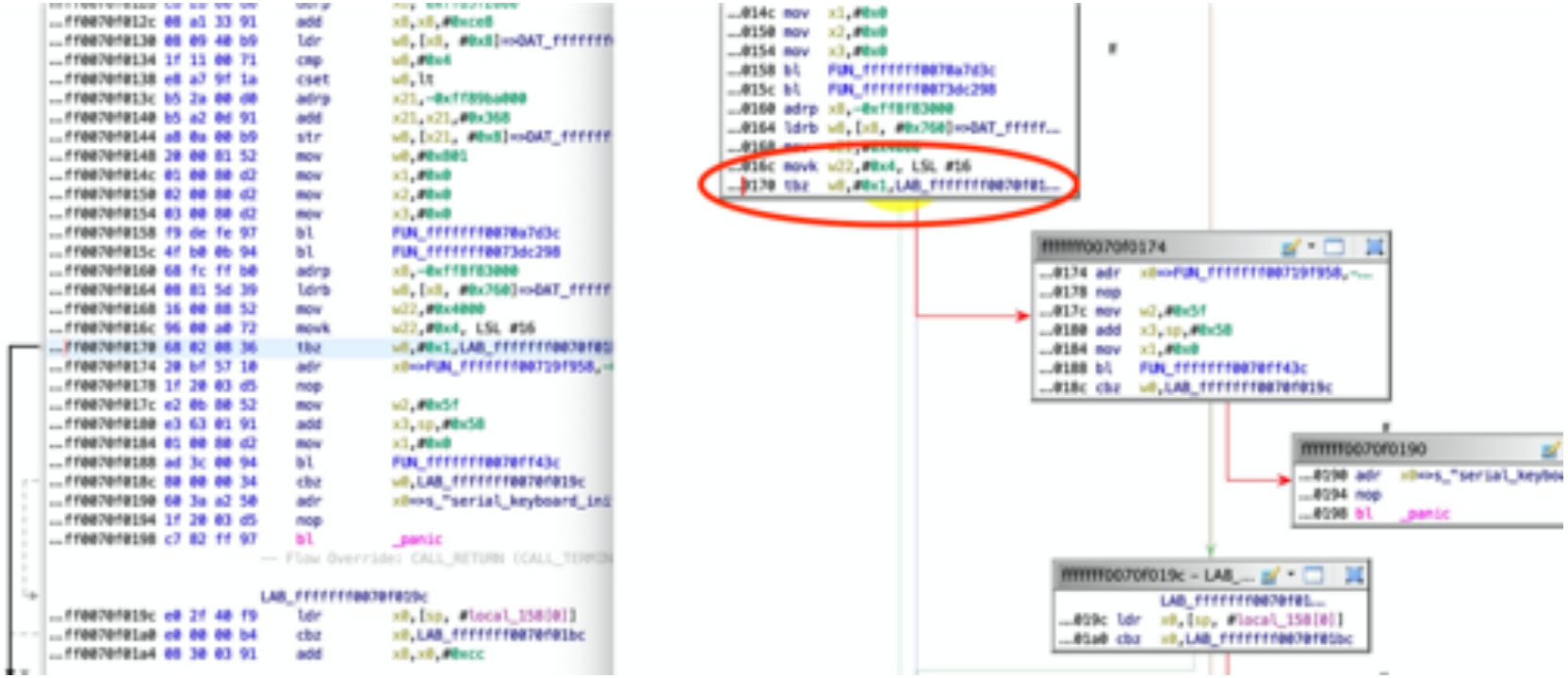
And it works!

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 8f 94    bl      _serial_init
...ff0071bad58 a8 1f 00 f0    adrp    x8,-0xff8a4f000
...ff0071bad5c 08 a1 84 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 8f 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 3e 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 8f 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
```

```
                    /* WARNING: Subroutine does not return */
  _panic("\"Platform Expert not initialized\"");
}
DAT_ffffff0076742c8 = 0x11;
DAT_ffffff0076742c0 = 0;
DataMemoryBarrier(2,3);
iVar9 = PE_parse_boot_argn("debug",&local_150_physmap_base,4,0);
if ((iVar9 != 0) && (((byte)local_150_physmap_base >> 3 & 1) != 0)) {
  DAT_ffffff007095cf0 = 0;
}
iVar9 = _serial_init();
_PE_kputc = FUN_ffffff0071a0360;
if (iVar9 != 0) {
  _PE_kputc = _serial_putc;
}
DAT_ffffff00707d760 = 0;
iVar9 = PE_parse_boot_argn("serial",&DAT_ffffff00707d760,4,0);
if (iVar9 != 0) {
  ((local_150_physmap_base =
          (longlong *)
          ((ulonglong)local_150_physmap_base & 0xffffffff00000000 |
          (ulonglong)_DAT_ffffff00707d760 & 0xffffffff00000004),
    (_DAT_ffffff00707d760 >> 2 & 1) != 0 ||
    ((iVar9 = PE_parse_boot_argn("drain_uart_sync",&local_150_physmap_base,4,0), iV
     ((uint)local_150_physmap_base != 0)))) {
  _DAT_ffffff00707d760 = _DAT_ffffff00707d760 | 4;
}
if (DAT_ffffff007507910 == 0) {
  local_150_physmap_base = (longlong *)((ulonglong)local_150_physmap_base & 0xffff
  iVar9 = PE_parse_boot_argn("validation_disables",&local_150_physmap_base,4,0);
  local_150_physmap_base._0_4_ = DAT_ffffff007507910;
  if (iVar9 != 0) {
    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!

```
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

```
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% 0C
[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