Simplifying iOS Research: Booting iOS on QEMU

@alephsecurity @JonathanAfek

github.com/alephsecurity
alephsecurity.com

Aleph, security research by HCL AppScan

How current iOS research is done

- Third party iOS emulator on a remote server
- Development fused iPhone
- Checkm8 demoted iPhone (@axi0mX)
- 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

Aleph Research

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

Aleph Research

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)

Aleph Research

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
```

Aleph Research

Goals of our project

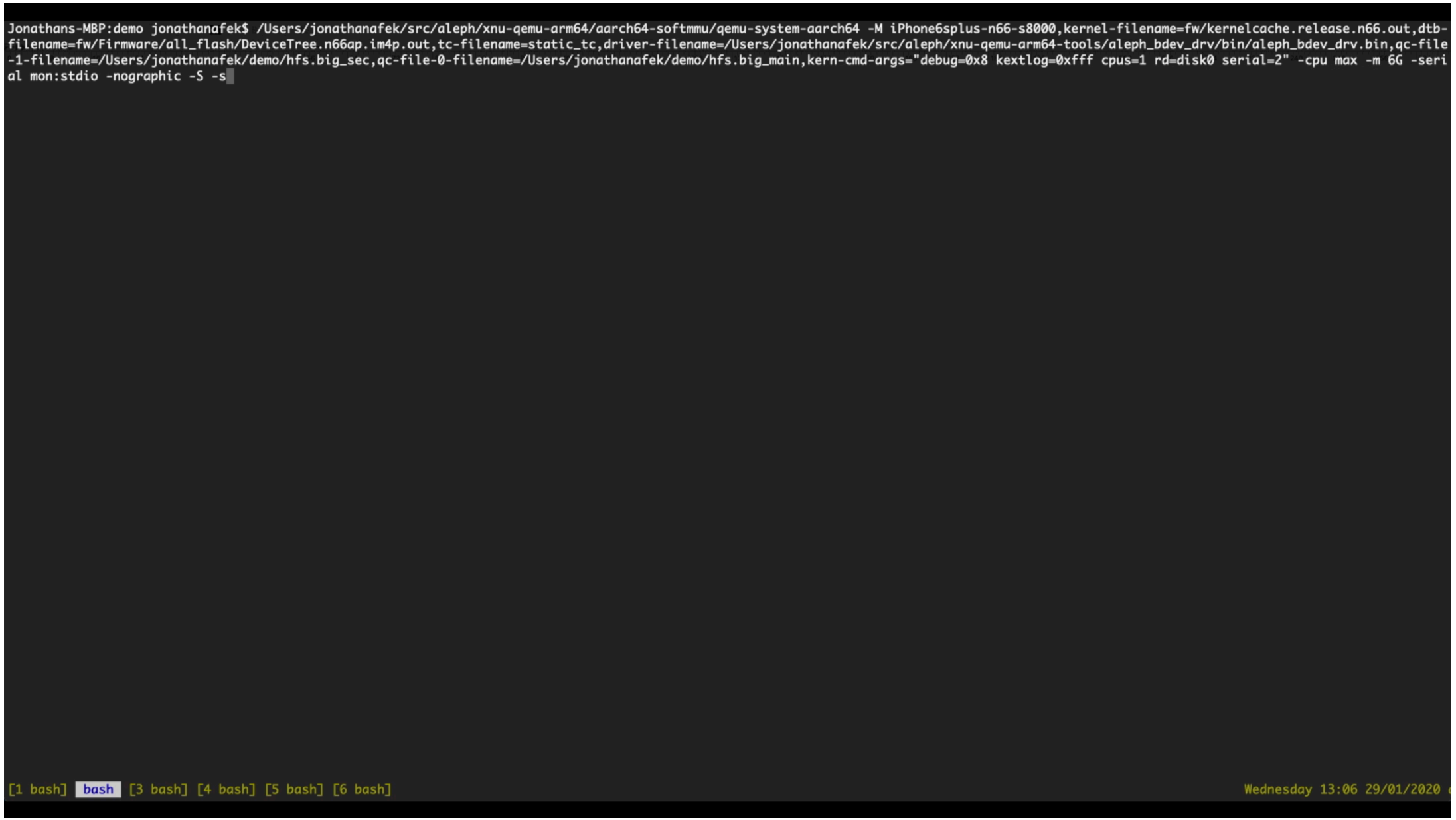
- Booting iOS on QEMU ~~with no kernel patches~~
- Supporting hardware (disk, display, touch, sound, multiple CPUs, Interrupt controllers, etc…)
- Supporting different iOS versions
- Conducting iOS security research
- Learning about iOS and QEMU internals

Aleph Research

Status of our project

- Booting ~~Secure Monitor~~ and the kernel ~~(unpatched)~~ on QEMU
- Executing user-mode apps over launchd
- Running an interactive bash shell on an iOS kernel on QEMU
- Running gdb scripts listing and switching (tasks, threads, zones)
- Allowing access to tfp0 from user mode
- Mounting full disk images with our own block device driver
- Running most of the iOS user mode services
- Connecting to an SSH server running on the iOS system
- Displaying a textual iOS framebuffer
- Supporting only on iOS 12.1 for iPhone 6s plus for now

Aleph Research


Jonathans-MBP:demo jonathanafek$ /Users/jonathanafek/src/aleph/xnu-qemu-arm64/aarch64-softmmu/qemu-system-aarch64 -M iPhone6splus-n66-s8000,kernel-filename=fw/kernelcache.release.n66.out,dtb-filename=fw/Firmware/all_flash/DeviceTree.n66ap.im4p.out,tc-filename=static_tc,driver-filename=/Users/jonathanafek/src/aleph/xnu-qemu-arm64-tools/aleph_bdev_drv/bin/aleph_bdev_drv.bin,qc-file-1-filename=/Users/jonathanafek/demo/hfs.big_sec,qc-file-0-filename=/Users/jonathanafek/demo/hfs.big_main,kern-cmd-args="debug=0x8 kextlog=0xfff cpus=1 rd=disk0 serial=2" -cpu max -m 6G -serial mon:stdio -nographic -S -s
[1 bash] bash [3 bash] [4 bash] [5 bash] [6 bash]
Wednesday 13:06 29/01/2020

```
(reverse-i-search)`ramf': /Users/jonathanafek/src/aleph/xnu-qemu-arm64/aarch64-softmmu/qemu-system-aarch64 -M iPhone6splus-n66-s8000,kernel-filename=fw/kernelcache.release.n66.out,dtb-filenam
e=fw/Firmware/all_flash/DeviceTree.n66ap.im4p.out,tc-filename=static_tc,driver-filename=/Users/jonathanafek/src/aleph/xnu-qemu-arm64-tools/aleph_bdev_drv/bin/aleph_bdev_drv.bin,qc-file-1-file
name=/Users/jonathanafek/demo/hfs.big_sec,qc-file-0-filename=/Users/jonathanafek/demo/hfs.big_main,xnu-ramfb=on,kern-cmd-args="debug=0x8 kextlog=0xfff cpus=1 rd=disk0 serial=2" -cpu max -m 6G
 -serial mon:stdio
```

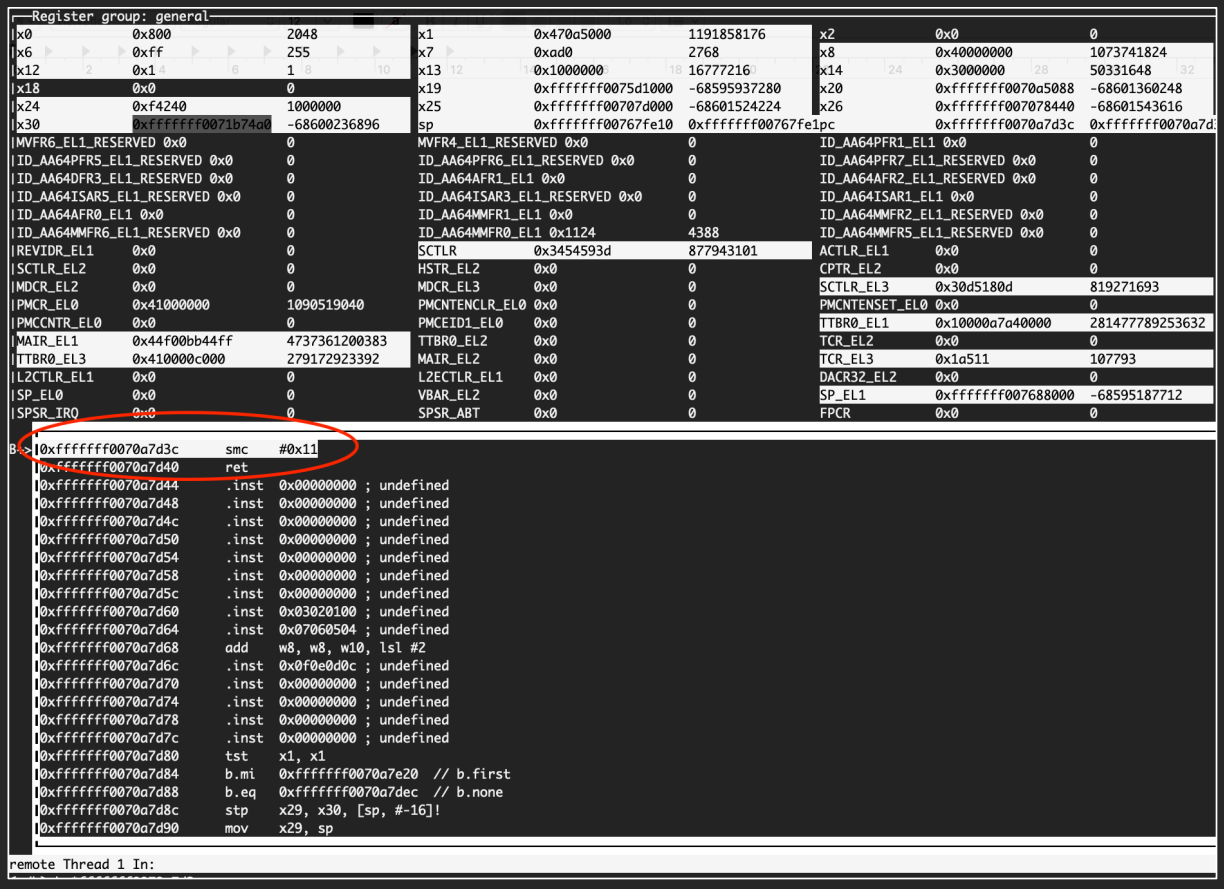
Agenda

- Past public research on iOS on QEMU
- **iOS kernel boot process**
- Execution of non-Apple executables with Trust Cache
- Bash execution with launchd
- tfp0
- SSH connection
- Textual framebuffer
- Block device driver
- No secure monitor
- Launchd craziness
- Next steps

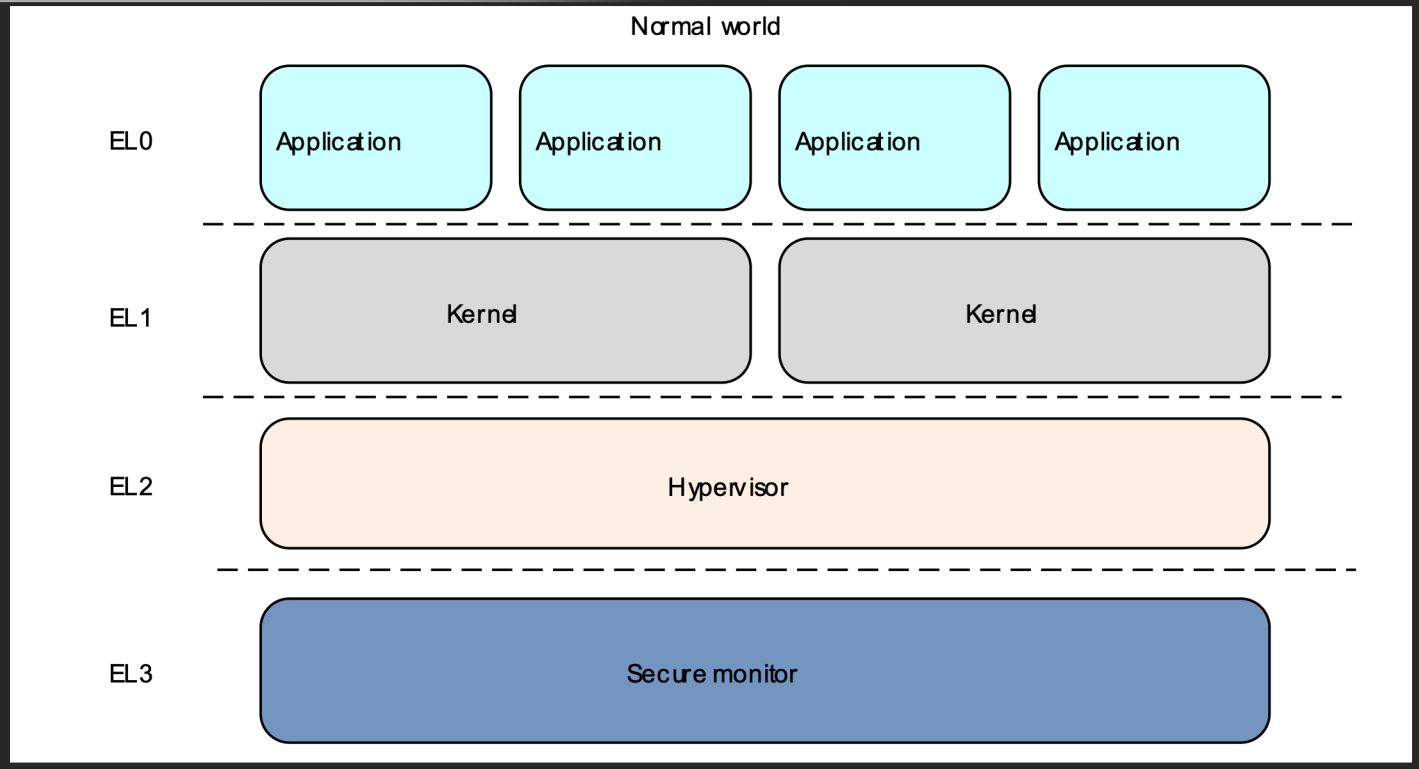
Aleph Research

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

- Load the Secure Monitor image for iOS 12.1 for iPhone 6s plus in the secure memory
- Load the secure monitor boot arguments in the secure memory with the relevant values
- Start executing the secure monitor at its entry point in EL3
- The secure monitor finishes its boot sequence and jumps to the kernel at its entry point in EL1
- The kernel finishes its boot seq and uses the secure monitor to service its SMC calls

Aleph Research

And it works!

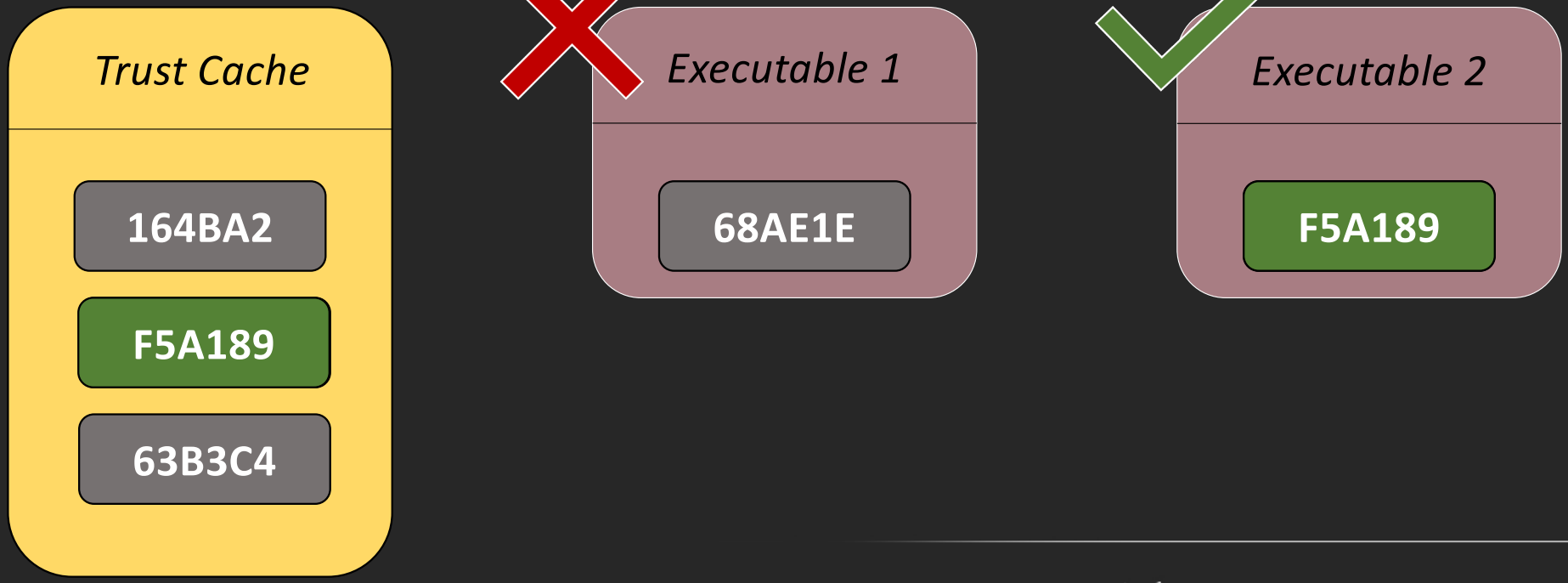
Aleph Research

Agenda

- Past public research on iOS on QEMU
- iOS kernel boot process
- **Execution of non-Apple executables with Trust Cache**
- Bash execution with launchd
- tfp0
- SSH connection
- Textual framebuffer
- Block device driver
- No secure monitor
- Launchd craziness
- Next steps

Aleph Research

Trust Cache



Aleph Research

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

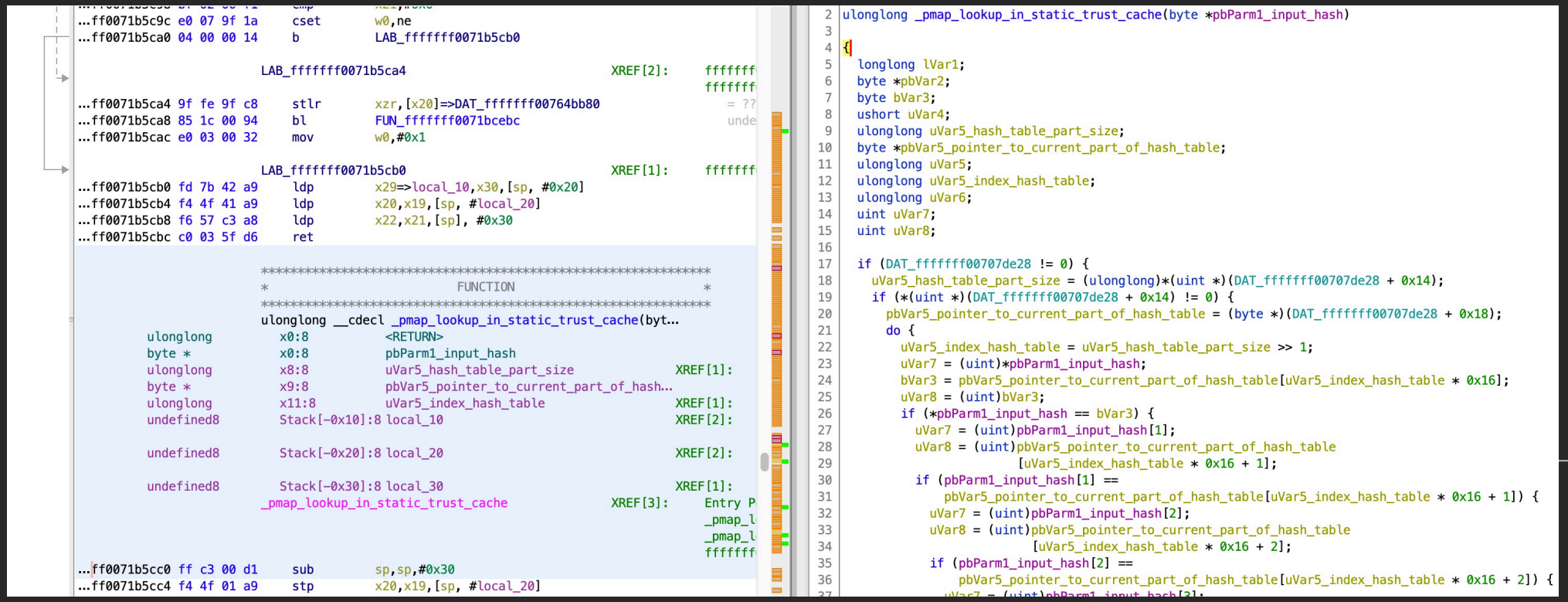
From Wikipedia, the free encyclopedia

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.

Aleph Research

Trust Cache

- By reversing this code and other relevant code we were able to understand the input structure and construct our own static TC



```
...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
                                                                                     ffffffff
...ff0071b5cc0 ff c3 00 d1     sub         sp,sp,#0x30
...ff0071b5cc4 f4 4f 01 a9     stp         x20,x19,[sp, #local_20]
```

```
ulonglong _pmap_lookup_in_static_trust_cache(byte *pbParm1_input_hash)

{
  longlong lVar1;
  byte *pbVar2;
  byte bVar3;
  ushort uVar4;
  ulonglong uVar5_hash_table_part_size;
  byte *pbVar5_pointer_to_current_part_of_hash_table;
  ulonglong uVar5;
  ulonglong uVar5_index_hash_table;
  ulonglong uVar6;
  uint uVar7;
  uint uVar8;

  if (DAT_ffffff00707de28 != 0) {
    uVar5_hash_table_part_size = (ulonglong)*(uint *)(DAT_ffffff00707de28 + 0x14);
    if (*(uint *)(DAT_ffffff00707de28 + 0x14) != 0) {
      pbVar5_pointer_to_current_part_of_hash_table = (byte *)(DAT_ffffff00707de28 + 0x18);
      do {
        uVar5_index_hash_table = uVar5_hash_table_part_size >> 1;
        uVar7 = (uint)*pbParm1_input_hash;
        bVar3 = pbVar5_pointer_to_current_part_of_hash_table[uVar5_index_hash_table * 0x16];
        uVar8 = (uint)bVar3;
        if (*pbParm1_input_hash == bVar3) {
          uVar7 = (uint)pbParm1_input_hash[1];
          uVar8 = (uint)pbVar5_pointer_to_current_part_of_hash_table
                        [uVar5_index_hash_table * 0x16 + 1];
          if (pbParm1_input_hash[1] ==
              pbVar5_pointer_to_current_part_of_hash_table[uVar5_index_hash_table * 0x16 + 1]) {
            uVar7 = (uint)pbParm1_input_hash[2];
            uVar8 = (uint)pbVar5_pointer_to_current_part_of_hash_table
                          [uVar5_index_hash_table * 0x16 + 2];
            if (pbParm1_input_hash[2] ==
                pbVar5_pointer_to_current_part_of_hash_table[uVar5_index_hash_table * 0x16 + 2]) {
```

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
}
```

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**
- tfp0
- SSH connection
- Textual framebuffer
- Block device driver
- No secure monitor
- Launchd craziness
- Next steps

Aleph Research

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

Aleph Research

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

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

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

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
- **tfp0**
- SSH connection
- Textual framebuffer
- Block device driver
- No secure monitor
- Launchd craziness
- Next steps

Aleph Research

tfp0

- task_for_pid() is a mach trap (like a system call) that allows getting access to another task port on the same host from usermode
- For pid 0 this gives access to the kernel task port
- Access to this port gives full control over the other task (memory read/write, code execution, etc..)

tfp0

```
kern_return_t task_for_pid(struct task_for_pid_args *args)
{
    // ...

    /* Always check if pid == 0 */
    if (pid == 0) {
        (void ) copyout((char *)&t1, task_addr, sizeof(mach_port_name_t));
        AUDIT_MACH_SYSCALL_EXIT(KERN_FAILURE);
        return(KERN_FAILURE);
    }

    // ...
}
```

from https://github.com/Siguza/hsp4/blob/master/README.md

Aleph Research

tfp0

```
task_t convert_port_to_task_with_exec_token(ipc_port_t port, uint32_t *exec_token)
{
    // ...

    if (task == kernel_task && current_task() != kernel_task) {
        ip_unlock(port);
        return TASK_NULL;
    }

    // ...
}
```

from https://github.com/Siguza/hsp4/blob/master/README.md

Aleph Research

tfp0 (@s1guza)

```
vm_map_remap(
    kernel_map,
    &remap_addr,
    sizeof(task_t),
    0,
    VM_FLAGS_ANYWHERE | VM_FLAGS_RETURN_DATA_ADDR,
    zone_map,
    kernel_task,
    false,
    &dummy,
    &dummy,
    VM_INHERIT_NONE
);
// mach_vm_wire_external as of High Sierra
mach_vm_wire(&realhost, kernel_map, remap_addr, sizeof(task_t), VM_PROT_READ | VM_PROT_WRITE);
ipc_port_t port = ipc_port_alloc_special(ipc_space_kernel);
ipc_kobject_set(port, remap_addr, IKOT_TASK);
realhost.special[4] = ipc_port_make_send(port);
```

from https://github.com/Siguza/hsp4/blob/master/README.md

tfp0

- Allocate static virtual memory for the remapped kernel task
- QEMU “MMIO” to redirect reads/writes from/to our fake kernel task to the real kernel task

```
ktpp->remap.orig_pa = vtop_mmu(global_kernel_ptr,  ktpp->cs);$
ktpp->remap.pa = remap_task_paddr;$
ktpp->remap.orig_as = ktpp->as;$
ktpp->remap.size = KERNEL_TASK_ALLOC_SIZE;$

xnu_dev_remap_create(sysmem, &ktpp->remap, "kernel_task_remap_dev");
```

Aleph Research

tfp0

- Create a copy of the kernel task port pointing to the remapped kernel task

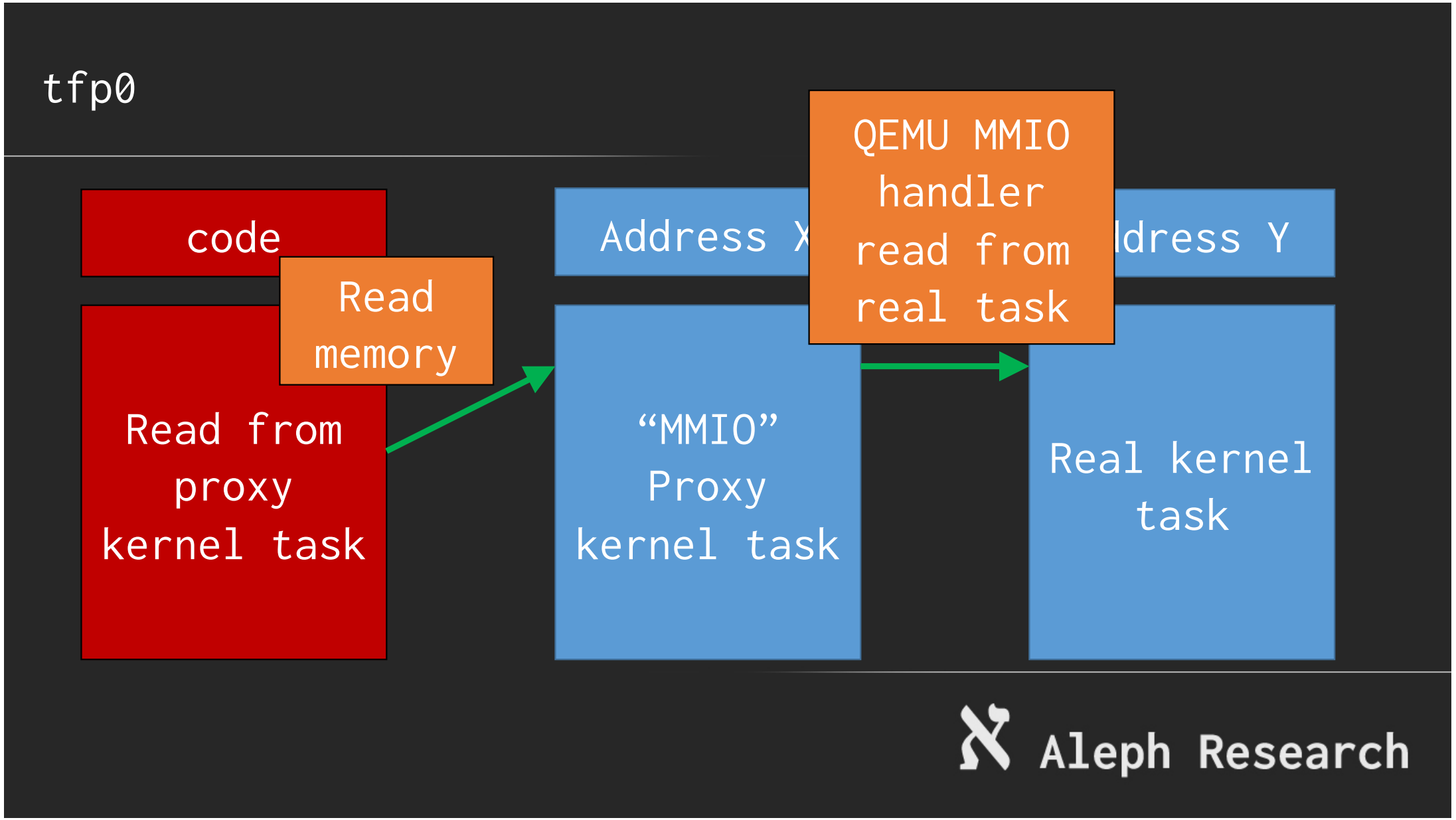
```
static void copy_kernel_task_port(hwaddr kernel_port_new_addr,$
                                  hwaddr kernel_task_new_addr,$
                                  hwaddr kernel_port_orig_addr,$
                                  AddressSpace *as)$
{$
    uint8_t port_buffer[KERNEL_PORT_SIZE_16B92];$
$
    //read out the kernel task port$
    address_space_rw(as, kernel_port_orig_addr, MEMTXATTRS_UNSPECIFIED,$
                     &port_buffer[0], KERNEL_PORT_SIZE_16B92, 0);$
$
    //ovr write the kernel task pointer with the new copy$
    *(uint64_t *)&port_buffer[KERNEL_TASK_OFFSET_16B92] = \$
        ptov_static(kernel_task_new_addr);$
$
    //write the new kernel task port to memory$
    address_space_rw(as, kernel_port_new_addr,$
                     MEMTXATTRS_UNSPECIFIED, &port_buffer[0],$
                     KERNEL_PORT_SIZE_16B92, 1);$
}$
```

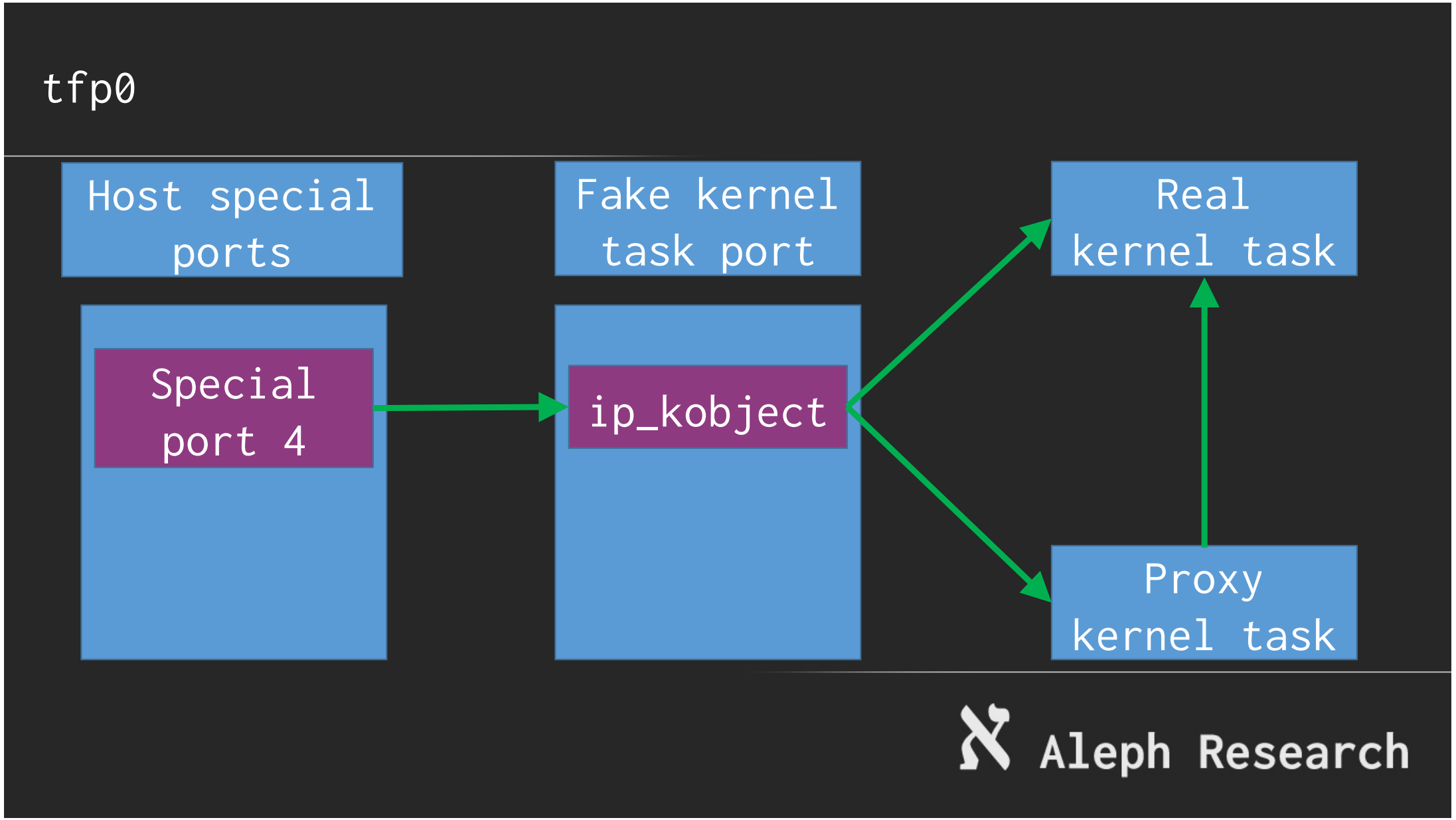
tfp0

- Copy the port to host special port 4

```
address_space_rw(ktpp->as, vtop_mmu(special_port_4_vaddr, ktpp->cs),$
                 MEMTXATTRS_UNSPECIFIED, (uint8_t *)&fake_port_vaddr,$
                 sizeof(hwaddr), 1);$
```

Aleph Research


tfp0
code
Read memory
Read from proxy kernel task
Address X
“MMIO” Proxy kernel task
QEMU MMIO handler read from real task
dress Y
Real kernel task
Aleph Research


tfp0
Host special ports
Fake kernel task port
Real kernel task
Special port 4
ip_kobject
Proxy kernel task
Aleph Research

tfp0

And it works!

```
-bash-4.4# kmem 0xffffff005c5c000 256
[*] Reading 256 bytes from 0xffffff005c5c000
CF FA ED FE 0C 00 00 01  00 00 00 00 0B 00 00 00  |................|
03 00 00 00 78 00 00 00  02 00 00 00 00 00 00 00  |....x...........|
19 00 00 00 48 00 00 00  5F 5F 4C 49 4E 4B 45 44  |....H..__LINKED|
49 54 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |IT..............|
68 0A 00 00 00 00 00 00  00 10 00 00 00 00 00 00  |h...............|
68 0A 00 00 00 00 00 00  01 00 00 00 01 00 00 00  |h...............|
00 00 00 00 04 00 00 00  02 00 00 00 18 00 00 00  |................|
00 10 00 00 44 00 00 00  40 14 00 00 28 06 00 00  |....D...@...(...|
CF FA ED FE 0C 00 00 01  00 00 00 00 0B 00 00 00  |................|
03 00 00 00 78 00 00 00  02 00 00 00 00 00 00 00  |....x...........|
19 00 00 00 48 00 00 00  5F 5F 4C 49 4E 4B 45 44  |....H..__LINKED|
49 54 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |IT..............|
E4 96 00 00 00 00 00 00  00 10 00 00 00 00 00 00  |................|
E4 96 00 00 00 00 00 00  01 00 00 00 01 00 00 00  |................|
00 00 00 00 04 00 00 00  02 00 00 00 18 00 00 00  |................|
00 10 00 00 99 03 00 00  90 49 00 00 54 5D 00 00  |.........I..T]..|
```

Agenda

- Past public research on iOS on QEMU
- iOS kernel boot process
- Execution of non-Apple executables with Trust Cache
- Bash execution with launchd
- tfp0
- **SSH connection**
- Textual framebuffer
- Block device driver
- No secure monitor
- Launchd craziness
- Next steps

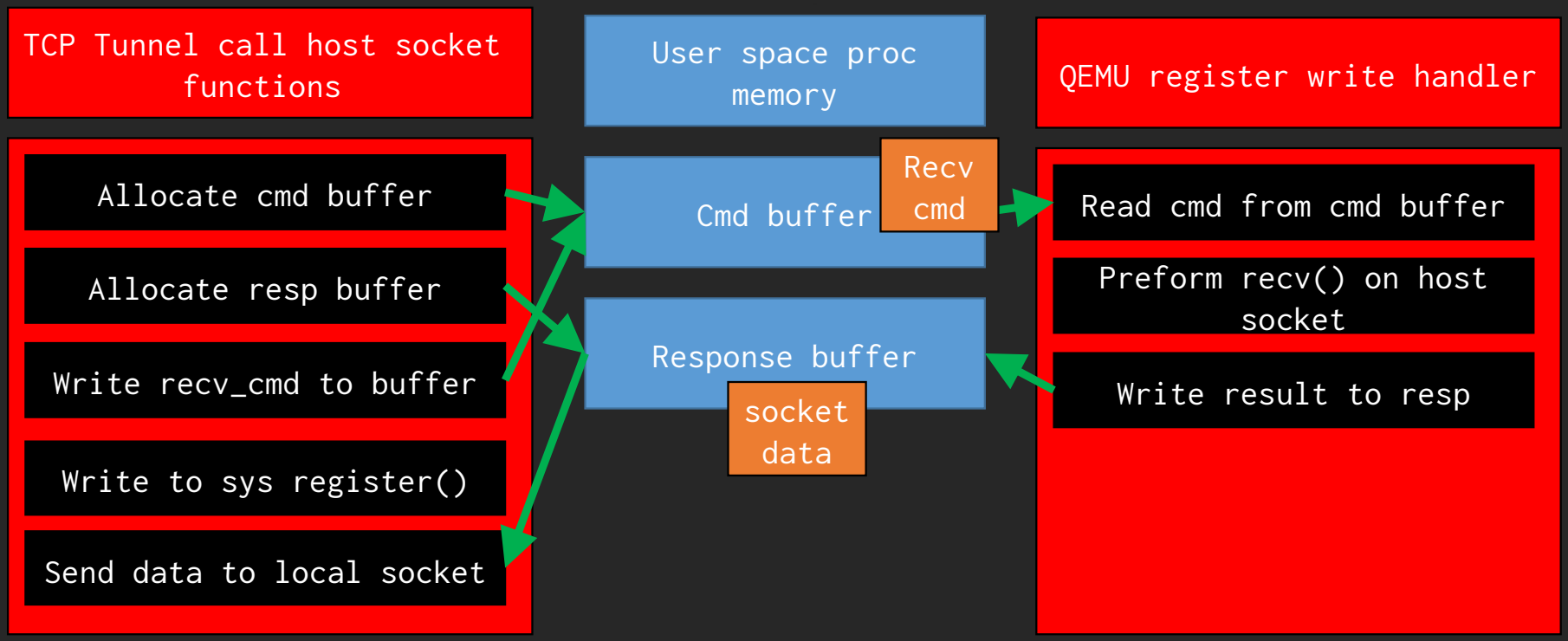
Aleph Research

TCP connections and SSH (@levaronsky)

- The “normal” way emulating the networking hardware
- The “normal” way with a driver and an emulated network controller for that driver
- Our way with communicating with the QEMU host from a usermode TCP tunnel proc using QEMU’s register write callbacks

Aleph Research

TCP connections and SSH



Aleph Research

TCP connections and SSH

- recv() from host socket and send() to guest socket and vice versa
- Very inefficient with a busy loop running endlessly without blocking at all when no traffic is available
- Works smoothly enough for now

```
Jonathans-MBP:~ jonathanafek$ ssh -p 2222 root@127.0.0.1
root@127.0.0.1's password:
[104] Dec 31 16:07:57 lastlog_perform_login: Couldn't stat /var/log/lastlog: No such file or directory
[104] Dec 31 16:07:57 lastlog_openseek: /var/log/lastlog is not a file or directory!
-bash-4.4# export PATH=$PATH:/iosbinpack64/usr/bin:/iosbinpack64/bin:/iosbinpack64/usr/sbin:/iosbinpack64/sbin
-bash-4.4# uname -a
Darwin localhost 18.2.0 Darwin Kernel Version 18.2.0: Tue Oct 16 21:02:23 PDT 2018; root:xnu-4903.222.5~1/RELEASE_ARM64_S8000 iPhone8,2
```

Aleph Research

Agenda

- Past public research on iOS on QEMU
- iOS kernel boot process
- Execution of non-Apple executables with Trust Cache
- Bash execution with launchd
- tfp0
- SSH connection
- **Textual framebuffer**
- Block device driver
- No secure monitor
- Launchd craziness
- Next steps

Aleph Research

Textual framebuffer (@V3rochka)

- Setup a new ramfb device with our own display parameters on the QEMU host side

```
void xnu_define_ramfb_device(AddressSpace* as, hwaddr ramfb_pa){$
$
    DeviceState *fb_dev;$
    fb_dev = qdev_create(NULL, TYPE_XNU_RAMFB_DEVICE);$
    qdev_prop_set_uint64(fb_dev, "as", (hwaddr)as);$
    qdev_prop_set_uint64(fb_dev, "fb_pa", ramfb_pa);$
    qdev_prop_set_uint32(fb_dev, "fb_size", RAMFB_SIZE);$
    qdev_prop_set_uint32(fb_dev, "display_cfg.height", V_HEIGHT);$
    qdev_prop_set_uint32(fb_dev, "display_cfg.width", V_WIDTH);$
    qdev_prop_set_uint32(fb_dev, "display_cfg.linesize", V_LINESIZE);$
    qdev_init_nofail(fb_dev);$
}$
```

Aleph Research

Textual framebuffer

- Setup the iOS boot args to configure the textual framebuffer

```
void  xnu_get_video_bootargs(void *opaque, hwaddr ramfb_pa){$
$
    video_boot_args* v_bootargs = (video_boot_args*)opaque ;$
    v_bootargs->v_baseAddr = ramfb_pa;$
    v_bootargs->v_depth = V_DEPTH;$
    v_bootargs->v_display = V_DISPLAY;$
    v_bootargs->v_height = V_HEIGHT;$
    v_bootargs->v_width = V_WIDTH;$
    v_bootargs->v_rowBytes = V_LINESIZE;$
$
}$
```

```
#define V_DEPTH      16$
#define V_HEIGHT     800$
#define V_WIDTH      600$
#define V_DISPLAY    0$
#define V_LINESIZE   (V_WIDTH * 3)$
```

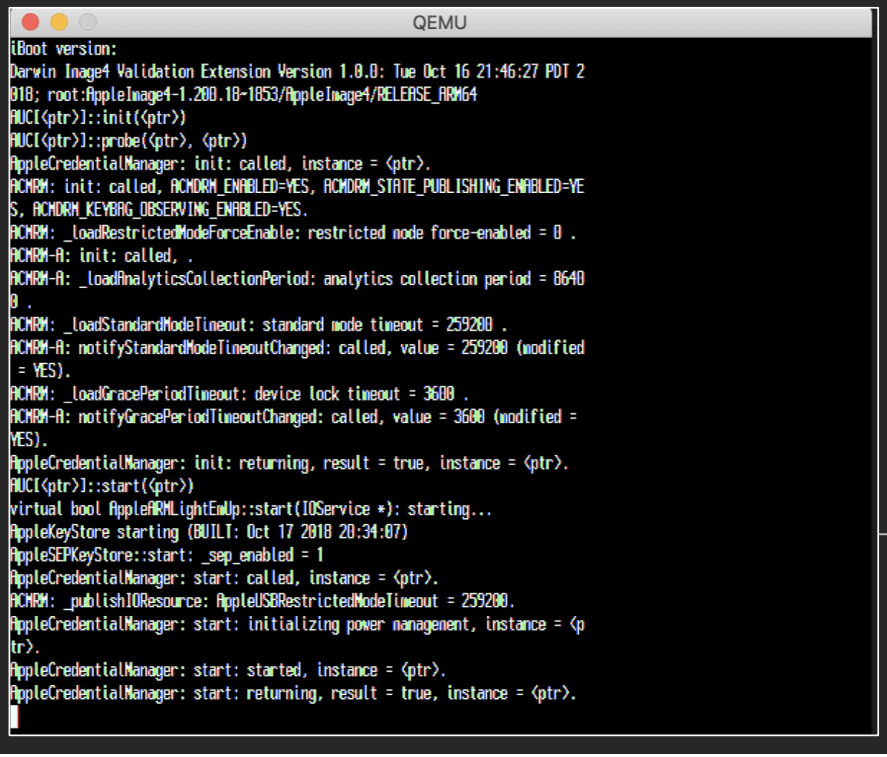
Aleph Research

Textual framebuffer

- Connect the QEMU ramfb device fb to the iOS framebuffer

Textual framebuffer

And it works!



Aleph Research

Agenda

- Past public research on iOS on QEMU
- iOS kernel boot process
- Execution of non-Apple executables with Trust Cache
- Bash execution with launchd
- tfp0
- SSH connection
- Textual framebuffer
- **Block device driver**
- No secure monitor
- Launchd craziness
- Next steps

Aleph Research

Block Device Driver

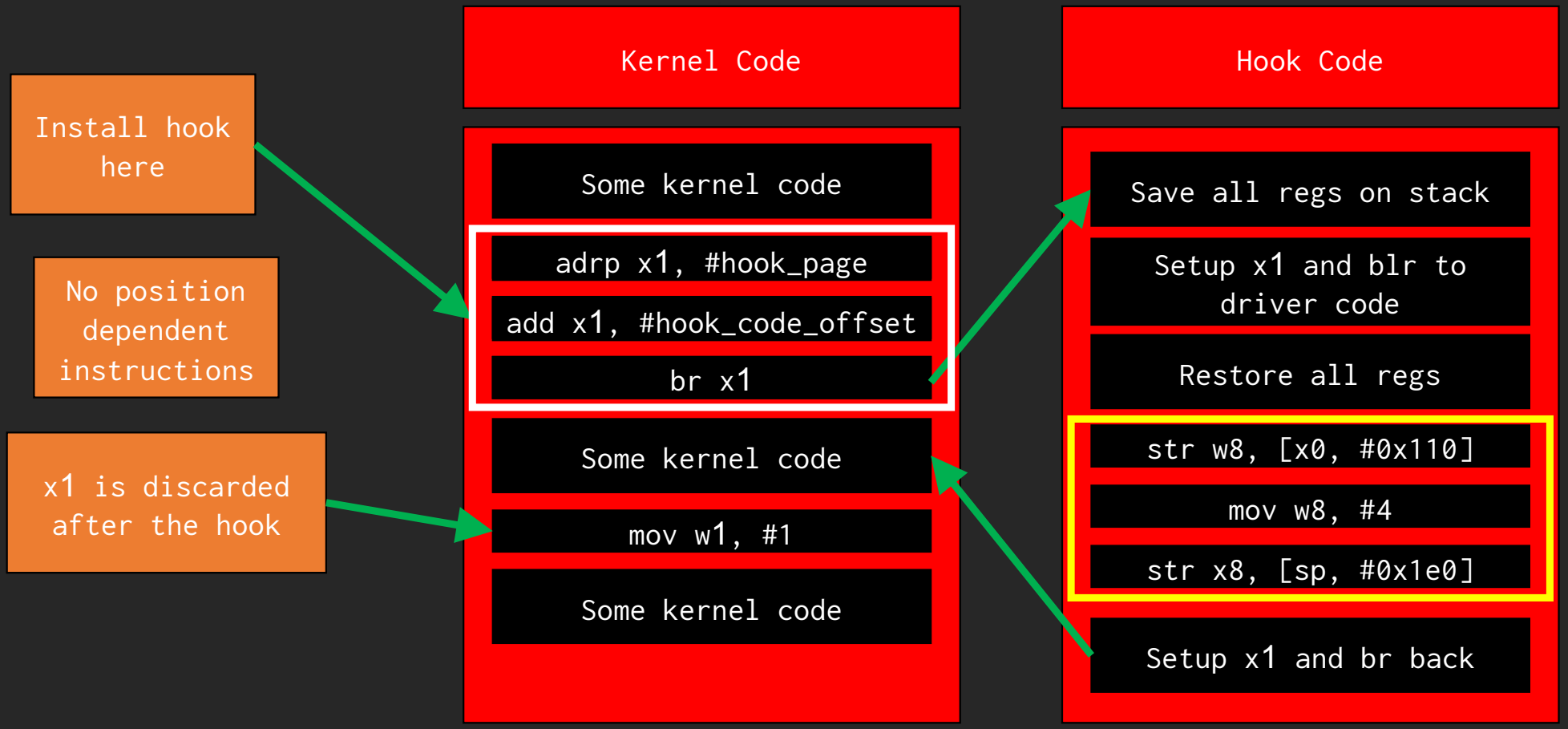
- RAMDisk block devices do not support disks larger than 1.5 GB
- Full disk image is larger than 4GB
- Using the RAMDisk we only have 1 block device
- We want a BD for the root mount and another for the R/W data mount - same as a regular iOS system
- Write our own block device driver to load the 2 large block devices
- Patch the kernel and call our driver during kernel boot

Aleph Research

Block Device Driver - Hook

- Allocate kernel memory at a fixed location
- Trampoline hook code in assembly
- Flat binary compilation of c code for the driver
- Parse the page table and make all these pages RWX

Block Device Driver - Hook



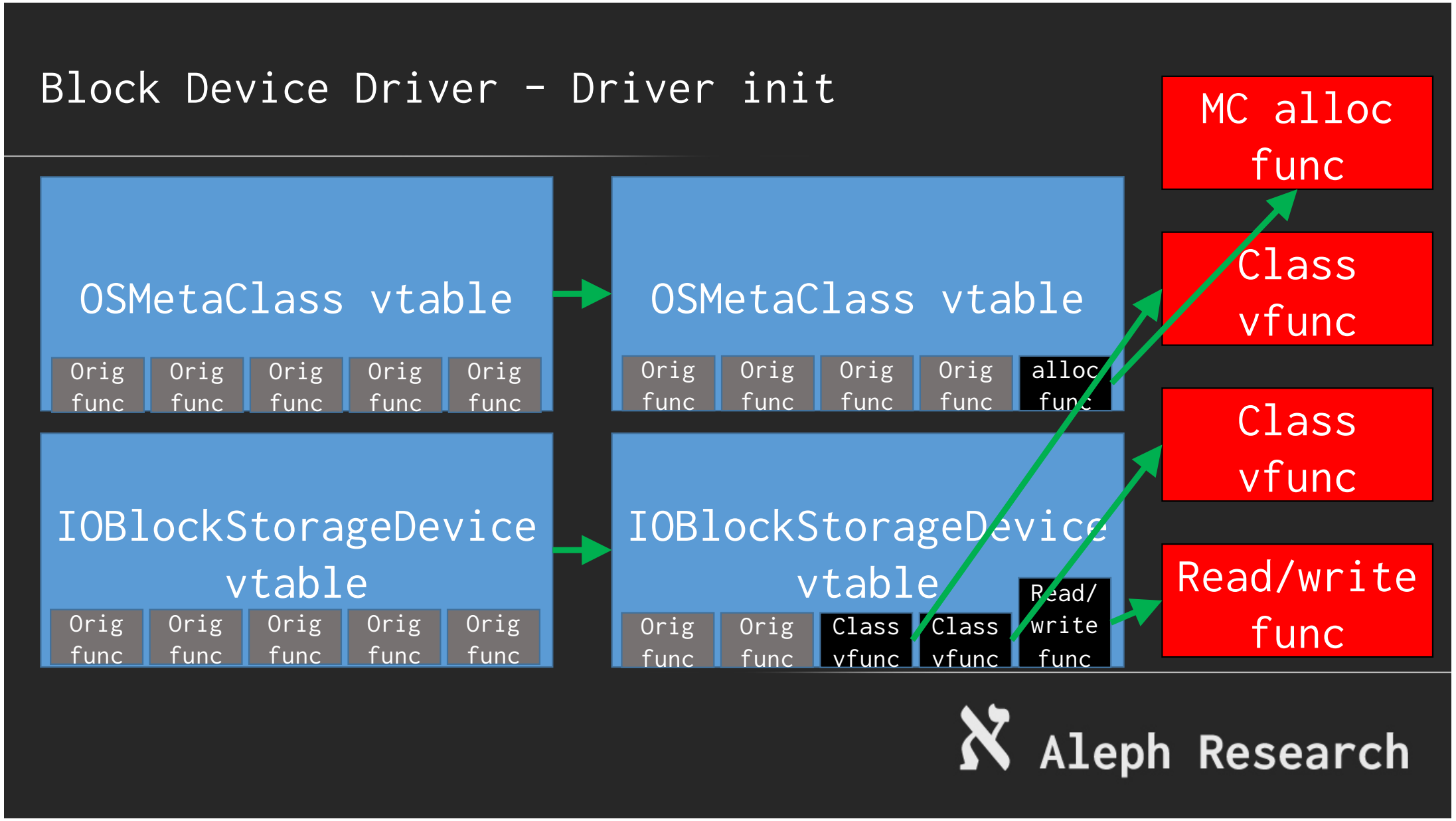
Block Device Driver - Driver init

- We do not have a kernel dev kit for iOS or a compatible cpp compiler
- Create our own driver inheriting from IOBlockStorageDevice with c code and build compatible cpp objects for the driver and its metaclass
- Create vtables for the new class and its OSMetaClass

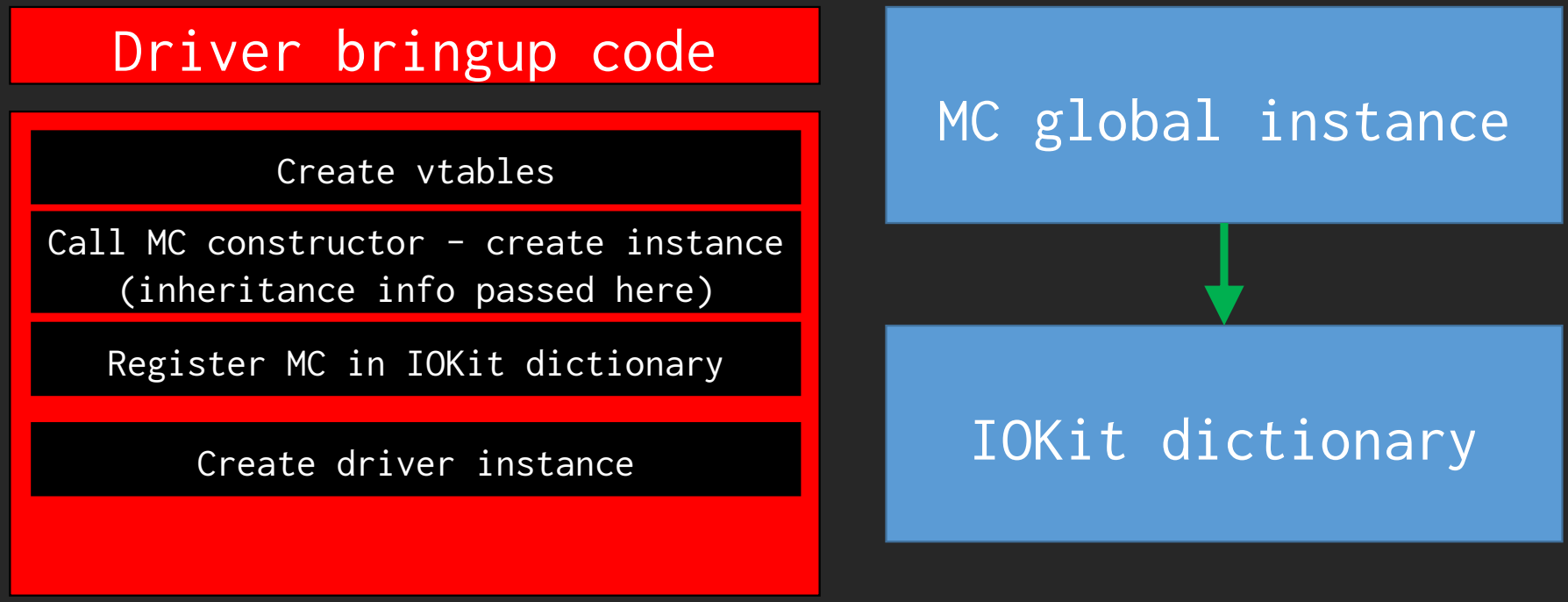
Block Device Driver - Driver init

- Call the OSMetaClass constructor and register the new class in the sAllClassesDict global classes dictionary
- Call OSMetaClass::allocClassWithName(“AlephStorageBlockDevice”) to create 2 instances of our class for the 2 block devices
- Attach them to a node in the IOKit registry and call ::registerService() on them to enable the new devices

Block Device Driver - Driver init



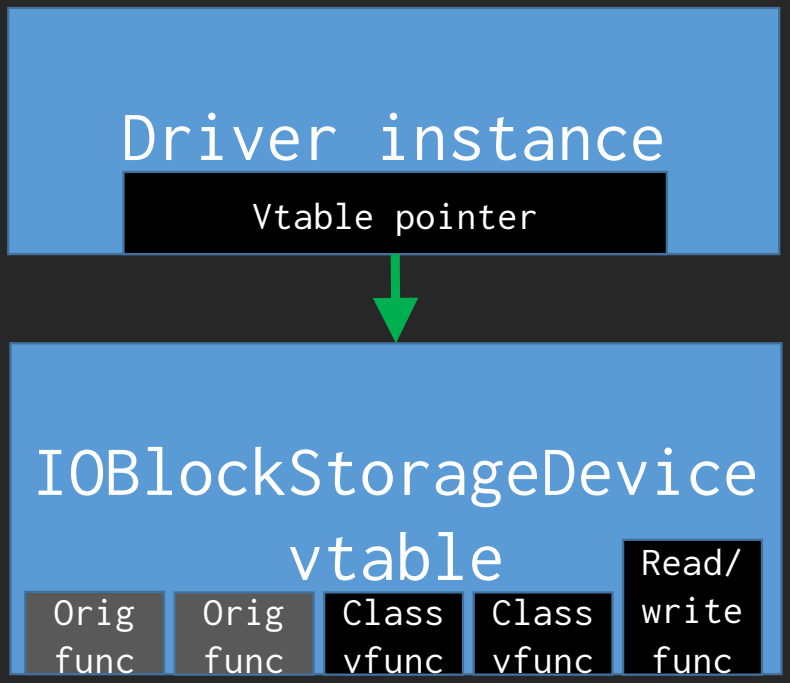
Block Device Driver - Driver init



Block Device Driver - Driver init

MC ::alloc() vfunc

- Allocate driver class instance
- Call super constructor
- Fix vtable pointer
- Call instanceConstructed() on MC inst



Aleph Research

Block Device Driver - Driver init

Driver bringup code

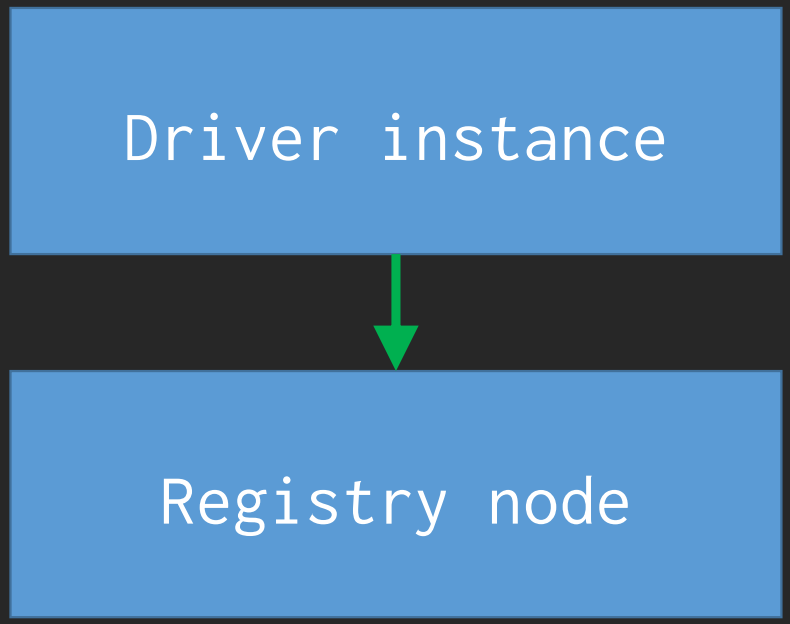
Find existing registry node

Attach driver to existing node

Call ::registerService() on inst

IOKit attaches drivers to inst

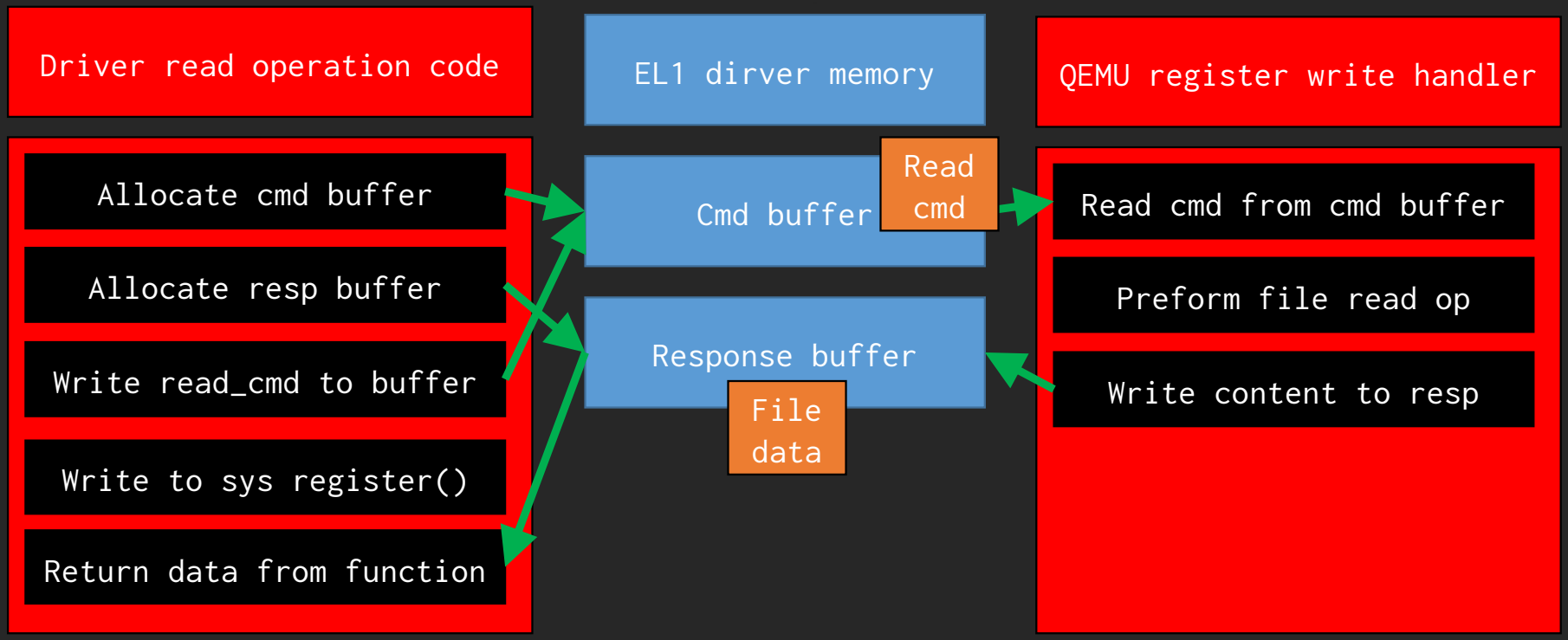
These drivers create a new BSD device



Block Device Driver - Driver read/write

- Implement the virtual functions to read/write from/to the new block devices
- Write to a system register and the QEMU callback reads/writes directly from/to memory buffers on the guest to transfer the data
- The block device data on the QEMU host is backed by files

Block Device Driver - Driver read/write



Aleph Research

```
    | | |
    | | +-o AlephStorageBlockDevice  <class IORegistryEntry:IOService:IOBlockStorageDevice:AlephStorageBlockDevice, id 0x1000001c7, registered, matched, active, busy 0 (633 ms), retain 7>
    | | | {
    | | |   "device-type" = "Generic"
    | | |   "IOMinimumSegmentAlignmentByteCount" = 0x4
    | | | }
    | | |
    | | +-o IOBlockStorageDriver  <class IORegistryEntry:IOService:IOStorage:IOBlockStorageDriver, id 0x1000001c8, registered, matched, active, busy 0 (613 ms), retain 8>
    | |   | {
    | |   |   "IOPropertyMatch" = {"device-type"="Generic"}
    | |   |   "IOProbeScore" = 0x0
    | |   |   "IOProviderClass" = "IOBlockStorageDevice"
    | |   |   "IOClass" = "IOBlockStorageDriver"
    | |   |   "CFBundleIdentifier" = "com.apple.iokit.IOStorageFamily"
    | |   |   "Statistics" = {"Operations (Write)"=0x0,"Latency Time (Write)"=0x0,"Bytes (Read)"=0x1a5f9000,"Errors (Write)"=0x0,"Total Time (Read)"=0x427c05e97,"Retries (Read)"=0x0,"Late
ations (Read)"=0x13cb,"Retries (Write)"=0x0}
    | |   |   "IOMatchCategory" = "IODefaultMatchCategory"
    | |   |   "IOGeneralInterest" = "IOCommand is not serializable"
    | |   | }
    | |   |
    | |   +-o 0Aleph 0AlephBDev Media  <class IORegistryEntry:IOService:IOStorage:IOMedia, id 0x1000001c9, registered, matched, active, busy 0 (613 ms), retain 9>
    | |     | {
    | |     |   "Removable" = No
    | |     |   "Content" = ""
    | |     |   "Whole" = Yes
    | |     |   "Leaf" = Yes
    | |     |   "BSD Name" = "disk0"
    | |     |   "Ejectable" = No
    | |     |   "Preferred Block Size" = 0x1000
    | |     |   "IOMediaIcon" = {"IOBundleResourceFile"="External.icns","CFBundleIdentifier"="com.apple.iokit.IOStorageFamily"}
    | |     |   "BSD Minor" = 0x0
    | |     |   "Writable" = Yes
    | |     |   "BSD Major" = 0x1
    | |     |   "Size" = 0x180000000
    | |     |   "Open" = Yes
    | |     |   "Content Hint" = ""
    | |     |   "BSD Unit" = 0x0
    | |     | }
    | |     |
    | |     +-o IOMediaBSDClient  <class IORegistryEntry:IOService:IOMediaBSDClient, id 0x1000001ca, registered, matched, active, busy 0 (3 ms), retain 7>
    | |         {
    | |           "IOClass" = "IOMediaBSDClient"
    | |           "IOMatchCategory" = "IOMediaBSDClient"
    | |           "IOProbeScore" = 0x7530
    | |           "IOProviderClass" = "IOMedia"
    | |           "IOResourceMatch" = "IOBSD"
    | |           "CFBundleIdentifier" = "com.apple.iokit.IOStorageFamily"
    | |         }
    | |
```

Block Device Driver

```
bash-4.4# mount -t hfs /dev/disk1 /mnt1

mount_hfs: Could not create property for re-key environment check: No such file or directory
hfs: mounted PeaceB16B92.arm64UpdateRamDisk on device disk1
bash-4.4#
bash-4.4# cd /mnt1/
bash-4.4# ls
.fseventsd              buddy                   log                     run
Keychains               containers              logs                    staged_system_apps
Managed Preferences     db                      mobile                  tmp
MobileAsset             dropbear                msgs                    vm
MobileDevice            empty                   networkd                wireless
MobileSoftwareUpdate    folders                 preferences
audit                   installd                root
bash-4.4#
```

Agenda

- Past public research on iOS on QEMU
- iOS kernel boot process
- Execution of non-Apple executables with Trust Cache
- Bash execution with launchd
- tfp0
- SSH connection
- Textual framebuffer
- Block device driver
- **No secure monitor**
- Launchd craziness
- Next steps

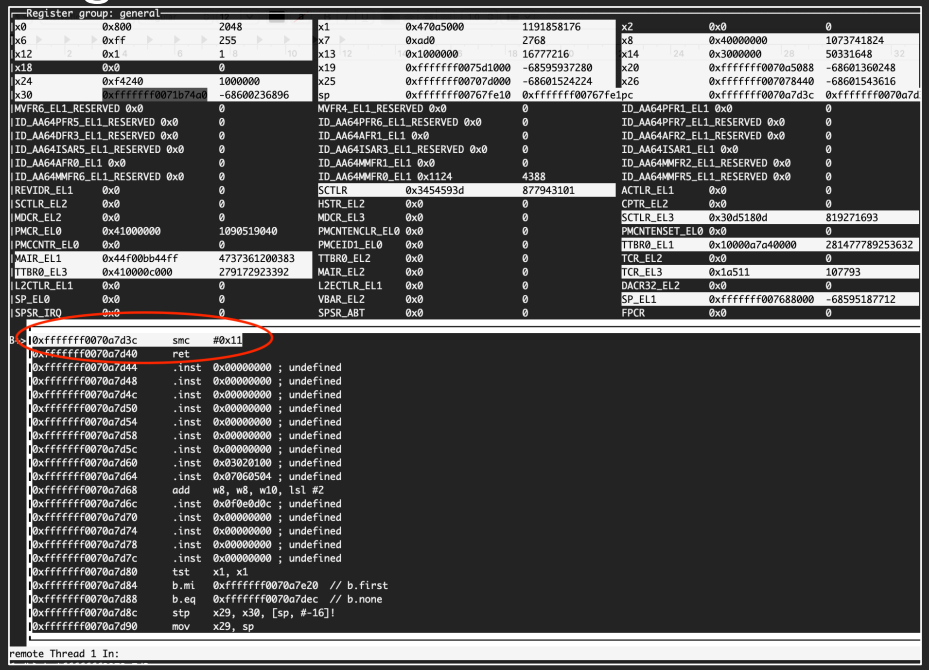
Aleph Research

No Secure Monitor

- Move from not patching the kernel to patching it for driver loading
- Disable the Secure Monitor to disable KPP

No Secure Monitor

- Don’t load the Secure Monitor image
- Start executing the kernel at EL1 at its entry point
- NOP SMC instruction (Secure Monitor Call)



Aleph Research

No Secure Monitor

- Another way KPP kicks in is by trapping FP operations done in EL0 or in EL1 to EL3 (secure monitor)
- Intercept writes to CPACR_EL1 (in QEMU) so that it will always hold 0x30000
  - This means FPEN is 3 (don’t trap any FP execution at all)
- xerub.github.io/ios/kpp/2017/04/13/tick-tock.html (@xerub)

Agenda

- Past public research on iOS on QEMU
- iOS kernel boot process
- Execution of non-Apple executables with Trust Cache
- Bash execution with launchd
- tfp0
- SSH connection
- Textual framebuffer
- Block device driver
- No secure monitor
- **Launchd craziness**
- Next steps

Aleph Research

Launchd craziness

- Created 2 raw HFS block device disk images with the full disk images content from the software update package
- Started QEMU pointing the root mount to the new block devices
- Added launchd items to execute our bash shell, start the TCP tunnel, start the dropbear SSH server and other services
- Our services didn’t run but the original system services did run

Any ideas?

Launchd craziness

- Removed all launchd items
- Still no go - they execute the same as before
- Removed the directory
- Panic and nothing executes
- Boot with the content of the ramdisk we used before
- Panic with an error about a missing service cache

Launchd craziness

```
lVar2 = _dlopen("/System/Library/Caches/com.apple.xpcd/xpcd_cache.dylib",2);
if ((lVar2 != 0) &&
   (UNRECOVERED_JUMPTABLE = (code *)_dlsym(lVar2,"__xpcd_cache"),
   UNRECOVERED_JUMPTABLE != (code *)0x0)) {
  iVar1 = (*UNRECOVERED_JUMPTABLE)();
  if (iVar1 != 1) {
    __os_assert_log(0);
    __os_crash();
                    /* WARNING: Could not recover jumptable at 0x00010000a248. Too many branches */
                    /* WARNING: Treating indirect jump as call */
    UNRECOVERED_JUMPTABLE = (code *)SoftwareBreakpoint(1,0x10000a24c);
    (*UNRECOVERED_JUMPTABLE)();
    return;
  }
  iVar1 = _dladdr(UNRECOVERED_JUMPTABLE,auStack72);
  if (iVar1 != 0) {
    puVar3 = _getsectiondata(local_40,"__TEXT","__xpcd_cache",(ulong *)&local_28);
    if (puVar3 == (uint8_t *)0x0) {
      DAT_1000436b8 = _xpc_dictionary_create(0,0,0);
    }
    else {
      DAT_1000436b8 = _xpc_create_from_plist(puVar3,local_28);
    }
  }
}
if ((DAT_1000436b8 == 0) ||
   (puVar4 = (undefined *)_xpc_get_type(), puVar4 != __xpc_type_dictionary)) {
  if (DAT_100044479 == '\0') {
                    /* WARNING: Subroutine does not return */
    FUN_100028f84("No service cache");
  }
  DAT_1000436b8 = _xpc_dictionary_create(0,0,0);
}
DAT_100043670 = _xpc_dictionary_get_value(DAT_1000436b8,"LaunchDaemons");
if ((DAT_100043670 == 0) ||
   (puVar4 = (undefined *)_xpc_get_type(), puVar4 != __xpc_type_dictionary)) {
  if (DAT_100044479 == '\0') {
                    /* WARNING: Subroutine does not return */
    FUN_100028f84("No daemons in cache");
  }
```

Launchd craziness

- Launchd decides whether to use this cache or the filesystem based on the bootargs (if it thinks it is a ramdisk boot)
- Patch this code “finds” rd=md0 in the boot args

```
lVar5 = FUN_100029448("kern.bootargs",&local_48);
if (lVar5 == 0) {
  cVar10 = '\0';
}
else {
  pcVar6 = _strnstr(local_48,"rd=md0",(size_t)lVar5);
  cVar10 = (char)pcVar6 + '\x01';
}
_free(local_48);
DAT_100044479 = cVar10;
```

Aleph Research

Launchd craziness

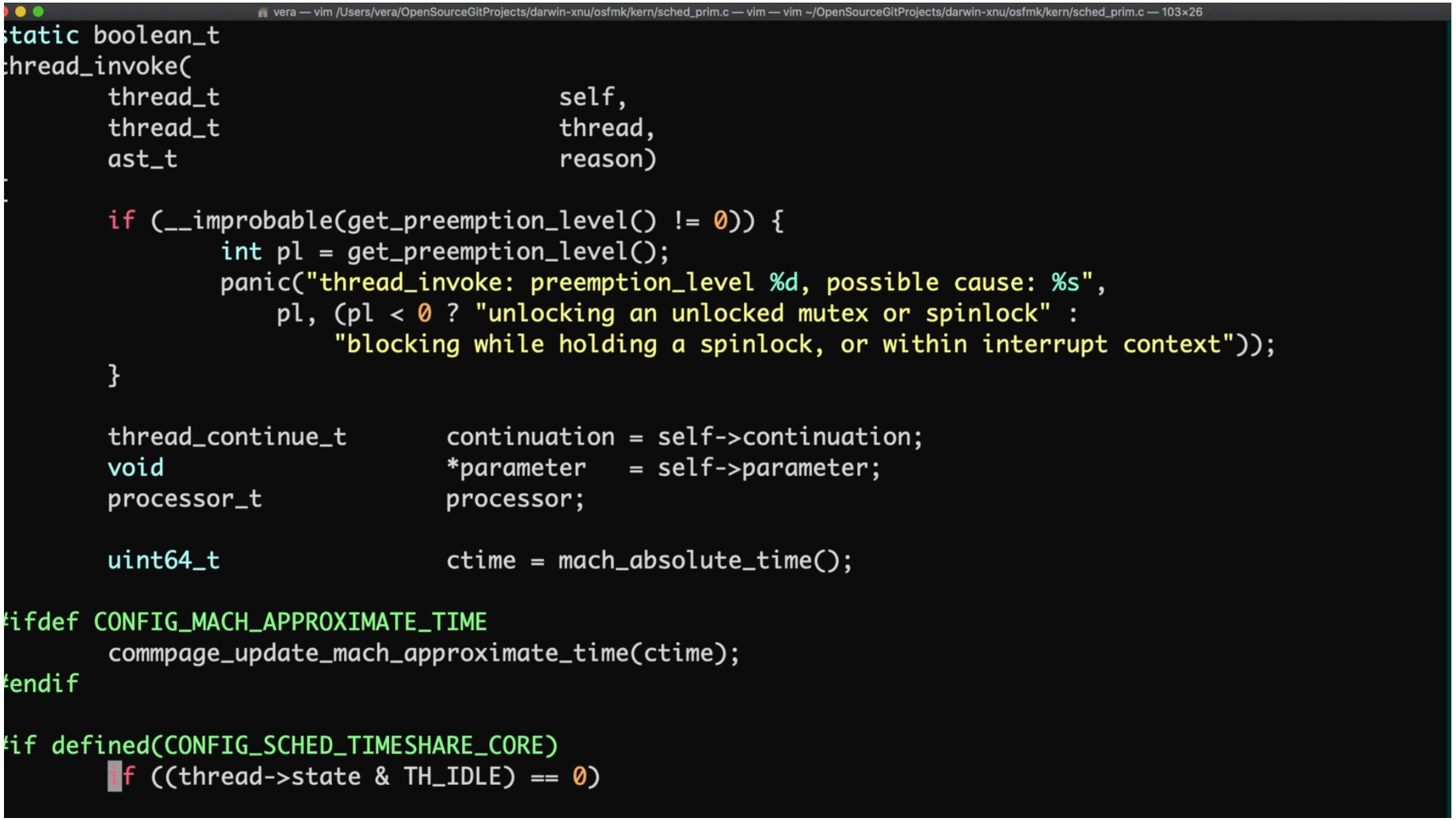
- Sign the patched binary
- Update the static trust cache
- Replace the launchd binary in the filesystem
- Remove the service cache dynamic lib file from the filesystem

Aleph Research

And it works!

Launchd craziness

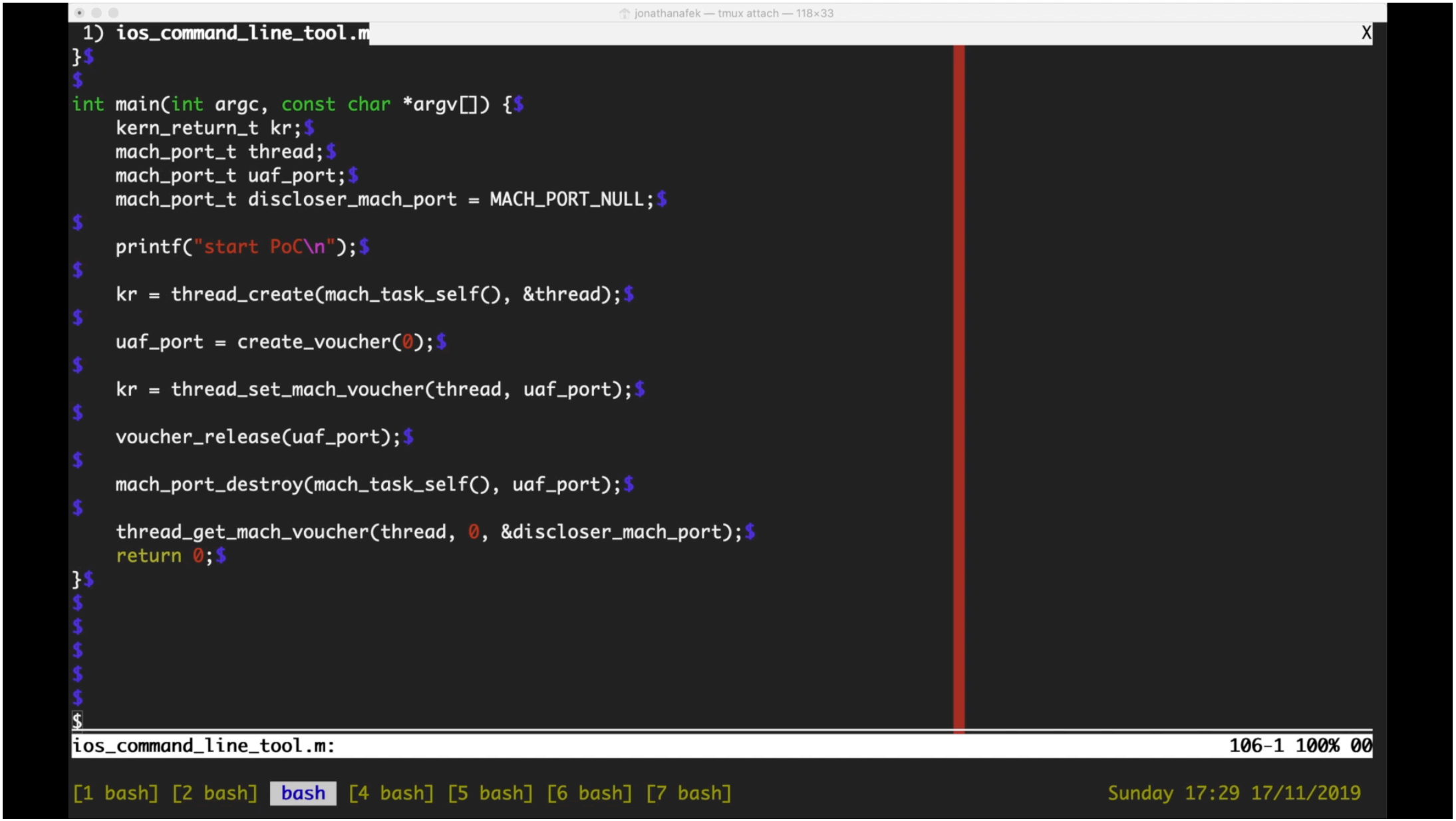
```
* [0 ] | "kernel_task"                          | 0xffffffe0005f9c20
  [1 ] | "launchd"                              | 0xffffffe0005fa760
  [19] | "xpcproxy"                            | 0xffffffe000c305a0
  [27] | "xpcproxy"                            | 0xffffffe000c61680
  [39] | "xpcproxy"                            | 0xffffffe000c98b40
  [44] | "xpcproxy"                            | 0xffffffe000cbd680
  [52] | "xpcproxy"                            | 0xffffffe000cbc5a0
  [60] | "xpcproxy"                            | 0xffffffe000cfe1c0
  [62] | "xpcproxy"                            | 0xffffffe000cfed00
  [6 ] | "fseventsd"                           | 0xffffffe000cff2a0
  [7 ] | "bash"                                | 0xffffffe000cff840
  [16] | "configd"                             | 0xffffffe0005fb2a0
  [36] | "SpringBoard"                         | 0xffffffe000d4d680
  [21] | "tunnel"                              | 0xffffffe000d4cb40
  [23] | "wifid"                               | 0xffffffe000d4dc20
  [28] | "mobiletimerd"                        | 0xffffffe000d4e1c0
  [19] | "xpcproxy"                            | 0xffffffe000d4ed00
  [62] | "xpcproxy"                            | 0xffffffe000d4f2a0
  [45] | "locationd"                           | 0xffffffe000d4f840
  [42] | "dropbear"                            | 0xffffffe000d4c000
  [60] | "xpcproxy"                            | 0xffffffe000c33840
  [39] | "xpcproxy"                            | 0xffffffe000c310e0
  [55] | "AppleCredentialManagerDaemon"        | 0xffffffe000c332a0
  [43] | "timed"                               | 0xffffffe000c990e0
  [50] | "mobilewatchdog"                      | 0xffffffe000c99680
  [51] | "ptpd"                                | 0xffffffe0005f9680
  [52] | "xpcproxy"                            | 0xffffffe0005f8b40
  [58] | "mapspushd"                           | 0xffffffe000c62760
```


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

Aleph Research


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
- tfp0
- SSH connection
- Textual framebuffer
- Block device driver
- No secure monitor
- Launchd craziness
- **Next steps**

Aleph Research

Next steps and challenges

- GUI and HID support
- SEP and keybagd (if required for regular functioning)
- Interrupt support for TCP tunnel performance improvements
- VPN connection testing over SSH
- Performance improvements
  - Virtualization instead of emulation (qemu-kvm)
  - More than 1 CPU
  - Understand what it means to use a real GPU for OpenGLES
- More iOS versions and iOS devices
- Work on symbolication with Jonathan Levin’s jtool/joker and otherwise for more versions
- Fast state recovery
- Security research and fast coverage-guided, structure-aware with state recovery fuzzing

Aleph Research

HLT

- 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