Friends! CountryMen! Lend me your task port!

Jonathan Levin, @Morpheus_____
http://NewOSXBook.com/
http://Technologeeks.com/

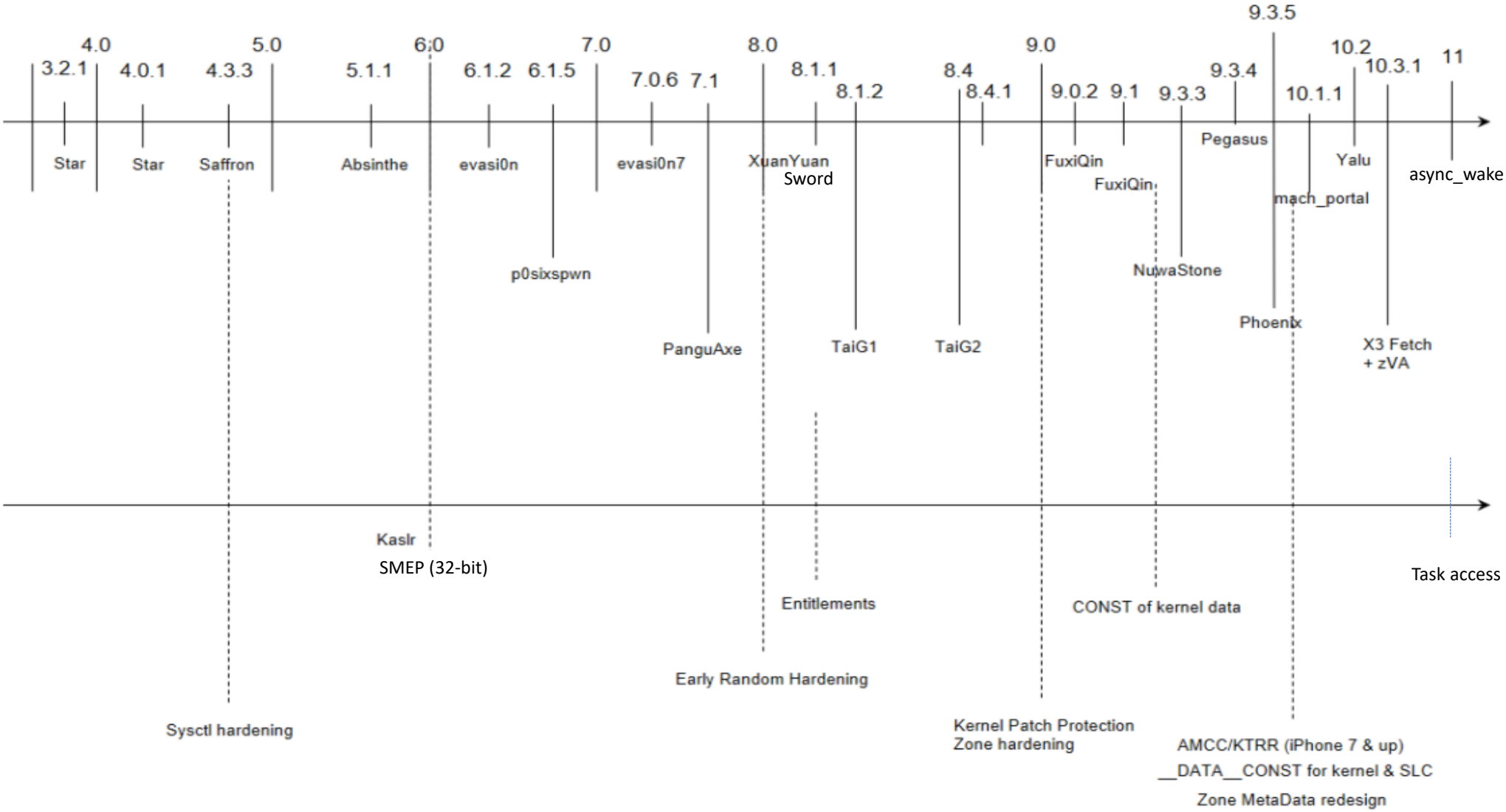
Tools used in this presentation:

- Joker: http://NewOSXBook.com/tools/joker.html
- Jtool: http://NewOSXBook.com/tools/jtool.html
  - Included in LiberiOS binaries, /jb/usr/local/bin
- QiLin writeup: http://NewOSXbook.com/QiLin/qilin.pdf

- As one download:
  - http://NewOSXBook.com/tools/hitbpack.tgz
  - To unpack (on a liberated i-Device):
    - mkdir /jb/tmp; cd /jb/tmp; tar xvf $OLDPWD/hitbpack.tgz

A brief history of jailbreaking

- Jailbreaks have been around for as long as the iPhone has
- Initial jailbreaks were very simple, just getting root access
  - Root access would suffice for all operations (e.g. disabling MACF)
- Apple incrementally stepped up defenses



Note: timeline is approximation – some features added in minor versions or betas

The present day

- Kernel memory protections prevent “traditional” kernel patching

- A7-A9 devices:
  - KPP (“watchtower”) runs at EL3, similar to Samsung KNOX’s PKM
  - Race conditions abound (due to interrupt driven nature of checks)
  - Todesco method proven to bypass KPP using fake page table entries

- A10 devices and later:
  - AMCC (“KTRR”) provides hardware-based defense
  - Initial implementation (pre iOS 10.1.1) also bypassable, but nothing since

KPP

- Introduced with iOS 9
- Contains code loaded (via Mach-O) into EL3 (Secure Monitor)
  - Joker automatically detects this in kernelcache
- Kernel loads into EL1
  - Unable to modify/affect EL3 by design

```
morpheus@Zephyr (~) %~/Documents/Work/JTool/joker -dec ~/Downloads/kernelcache.release.ipad5
mmapped: 0x120dde000
Feeding me a compressed kernelcache, eh? That's fine, now. I can decompress!
Compressed Size: 13879505, Uncompressed: 27459584. Unknown (CRC?): 0x17e4a3b, Unknown 1: 0x1
btw, KPP is at 13879940 (0xd3ca84)..And I saved it for you in /tmp/kpp
Got kernel at 436
got mem 0x121b22000
mmapped: 0x121b22000
This is a 64-bit kernel from iOS 11.0 , or later This is a 64-bit kernel from iOS 11.x (b1+),
or later (4570.20.62.0.0)
ARM64 Exception Vector is at file offset @0x93000 (Addr: 0xfffffff007097000)
morpheus@Zephyr (~) %jtool -l /tmp/kpp                                                  3:56
LC 00: LC_SEGMENT_64          Mem: 0x4100000000-0x4100006000    __TEXT
       Mem: 0x4100001000-0x4100005ca4          __TEXT.__text   (Normal)
       Mem: 0x4100005ca4-0x4100005d64          __TEXT.__const
       Mem: 0x4100005d64-0x4100005dca          __TEXT.__cstring        (C-String Literals)
LC 01: LC_SEGMENT_64          Mem: 0x4100006000-0x410000c000    __DATA
       Mem: 0x4100006000-0x410000b1f8          __DATA.__common (Zero Fill)
       Mem: 0x410000b200-0x410000b480          __DATA.__bss    (Zero Fill)
LC 02: LC_SEGMENT_64          Mem: 0x410000c000-0x410000c000    __IMAGEEND
       Mem: 0x410000c000-0x410000c000          __IMAGEEND.__dummy
LC 03: LC_SEGMENT_64          Mem: 0x410000c000-0x410000c000    __LINKEDIT
LC 04: LC_SYMTAB
       Symbol table is at offset 0x0 (0), 0 entries
       String table is at offset 0x0 (0), 0 bytes
LC 05: LC_UUID                UUID: 5873C3C0-CF86-30E4-AF57-E8E1B2B12361
LC 06: LC_SOURCE_VERSION      Source Version:         374.20.8.0.0
LC 07: LC_UNIXTHREAD          Entry Point:            0x4100001814
```

KPP

- On Boot, two voluntary transitions:
  - SMC #2048 (from machine_idle_init): set ARM EL1_VBAR
  - SMC #2049 (machine_lockdown): Finalize (and protect) kernel static data.

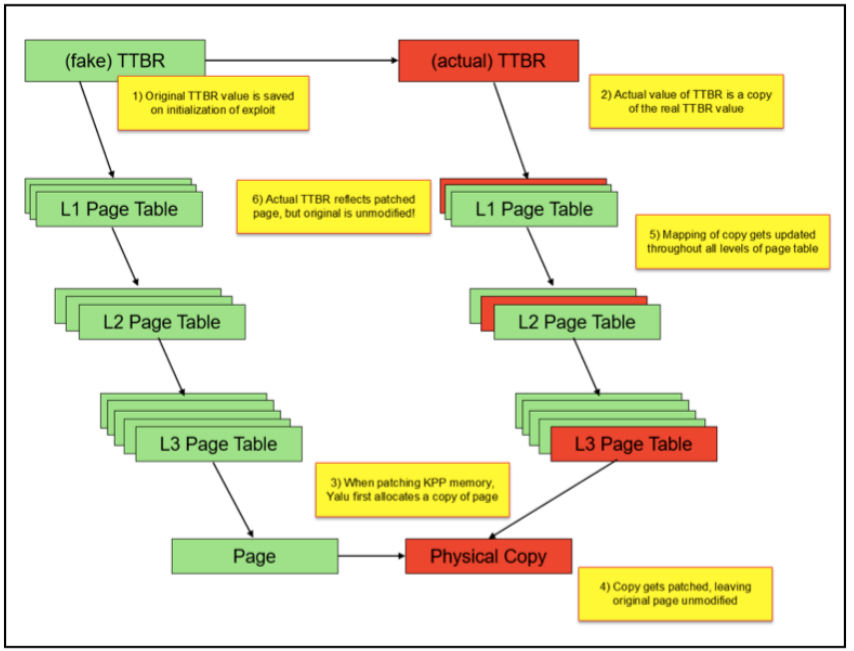
- Functionality very similar to Samsung's PKM:
  - Traps Floating Point operations (CPACR_EL1)
  - Compares protected (r/o) page blake2 hashes to store (by limited budget)
  - Also verifies EL1 system registers (TTBR1, VBAR, SCTLR, etc)

  - Detailed by @Xerub
  - Also explained in detail in *OS Internals Vol III, pp 271-276

KPP

- Obvious design fault:
  - FPU + IRQ driven operation and limited budget leave HUGE window
  - Can quickly patch/unpatch (e.g. TFP0, patch get kernel_task to user, unpatch)

- Bypassed ingeniously by Luca Todesco



KTRR

- MUCH better, hardware enforced mechanism in A10+ devices
- Hardware immediately detects/kills patching
- Configured by software using KTRR (ror)
- No (publicly known) bypass since 10.1.1
- Exposed by Apple as of XNU-4570 sources

Listing 13-18: KTRR code (from XNU-4570's ../machine_routines.c) interleaved with d10 11.0.1 disassembly

```
// lock_amcc is inlined
static void lock_amcc() {
#if defined(KERNEL_INTEGRITY_KTRR)

        rRORGNLOCK = 1;
fffffff00711db00 LDR  X8, [X25, #536] ; R8 = .. *(0xfffffff007652218, amcc_base
fffffff00711db04 ORR  W9, WZR, #0x1   ; R9 = 0x1
fffffff00711db08 STR  W9, [X8, #2028] ;$ *(R8 + 2028) = 1
      __builtin_arm_isb(ISB_SY);
fffffff00711db0c ISB  SY

#else
#error KERNEL_INTEGRITY config error
#endif
}

// lock_mmu() also inlined: x20 = begin, x19 = end. x9 = 1 (from ..db04)
static void lock_mmu(uint64_t begin, uint64_t end) {

#if defined(KERNEL_INTEGRITY_KTRR)

        __builtin_arm_wsr64(ARM64_REG_KTRR_LOWER_EL1, begin); // S3_4_c15_c2_3
fffffff00711db10        MSR     S3_4_C15_C2_3, X20      ..

        __builtin_arm_wsr64(ARM64_REG_KTRR_UPPER_EL1, end);   // S3_4_c15_c2_4
fffffff00711db14        MSR     S3_4_C15_C2_4, X19      ..

        __builtin_arm_wsr64(ARM64_REG_KTRR_LOCK_EL1,  1ULL);  // S3_4_c15_c2_2
fffffff00711db18        MSR     S3_4_C15_C2_2, X9       ..

        /* flush TLB */

        __builtin_arm_isb(ISB_SY);
fffffff00711db0c        ISB     SY

        flush_mmu_tlb();
fffffff00711db1c        ISB                     ;
fffffff00711db20        BL      0xfffffff0070d48ac

#else
#error KERNEL_INTEGRITY config error
#endif
```

The Future

- Going forward, KTRR isn't going away, and is preventing:
  - Text/code patching
    - Rendering the standard set of patches (e.g. TFP0, setuid,..) impossible
  - Read-only memory patching
    - Other patches (i_can_haz_debugger, AMFI hooks, Sandbox platform profile) impossible
  - Trivial kernel code-injection:
    - mach_vm_allocate/mach_vm_protect(PROT_EXEC)

…e pur si rompe..

- Patch protection still falls short in various aspects, notably “data-only”
  - Mutable data (i.e. rw by design) cannot be protected with present methods

- Mutable (ergo patchable) data still holds plenty of kernel structures:
  - `struct proc`: Process control blocks, including credentials, Kauth & MACF labels
  - `struct vnode`: Loaded inodes, including open file metadata
  - Unified Buffer Cache: Including file data, code signature blobs & entitlements
  - IO*: IOKit objects, providing vtables aplenty and code execution primitives

Fortunately…

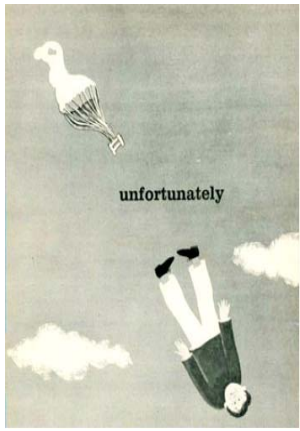
- Publicly disclosed vulnerabilities, notably Ian Beer's, now give "TFP0"
  - Method was used forever in jailbreaks, but never before "standardized" in PoCs

- More accurately, the `kernel_task` port is smuggled to userland, providing:
  - Task APIs enable control over task aspects, specifically kernel threads
  - mach_vm* APIs manipulating kernel memory (subject to patching restrictions)
  - Name stuck because of traditional `task_for_pid`(..,0..) patch.

  - To get an idea: q.v. miscellaneous MIG .defs in /usr/include/mach

Unfortunately…

- Access to kernel task is just the beginning
- With great power comes great responsibility
- Open source nature of PoC exploits leads to cut/paste low quality JBs
- Kernel memory, when touched the wrong way, leads to panics



QiLin (麒麟)

- Attempt to “standardize” JBs with a simple, reusable library
- The QiLin jailbreak toolkit “drives” LiberTV/LiberiOS/LiberWatchee
  - Also drives Technologeeks’ Xn00p (XNU kernel debug tool, coming soon)
- Minimalistic (“dev”) jailbreaks (no Cydia ☺) but rock solid stability
- Extensible API, allowing a jailbreak in 10 lines of code or less.
- Not open source (yet), but fully documented with public API.

QiLin (麒麟)

- Only requirement is the `kernel_task` port
- Alternatively: Provide your own exploit (=kernel mem r/w primitives)
  - `int readKernelMemory(uint64_t Address, uint64_t Len, void **To);`
  - `int writeKernelMemory(uint64_t Address, uint64_t Len, void *From);`
  - Useful in earlier stages of exploit development, when port is not yet obtainable
- Provides tested, reusable code to achieve most jailbreaking tasks
- Core is also undetectable
  - Tip: Treat “jailbreak detection” claimz suspiciously

Utility Functions

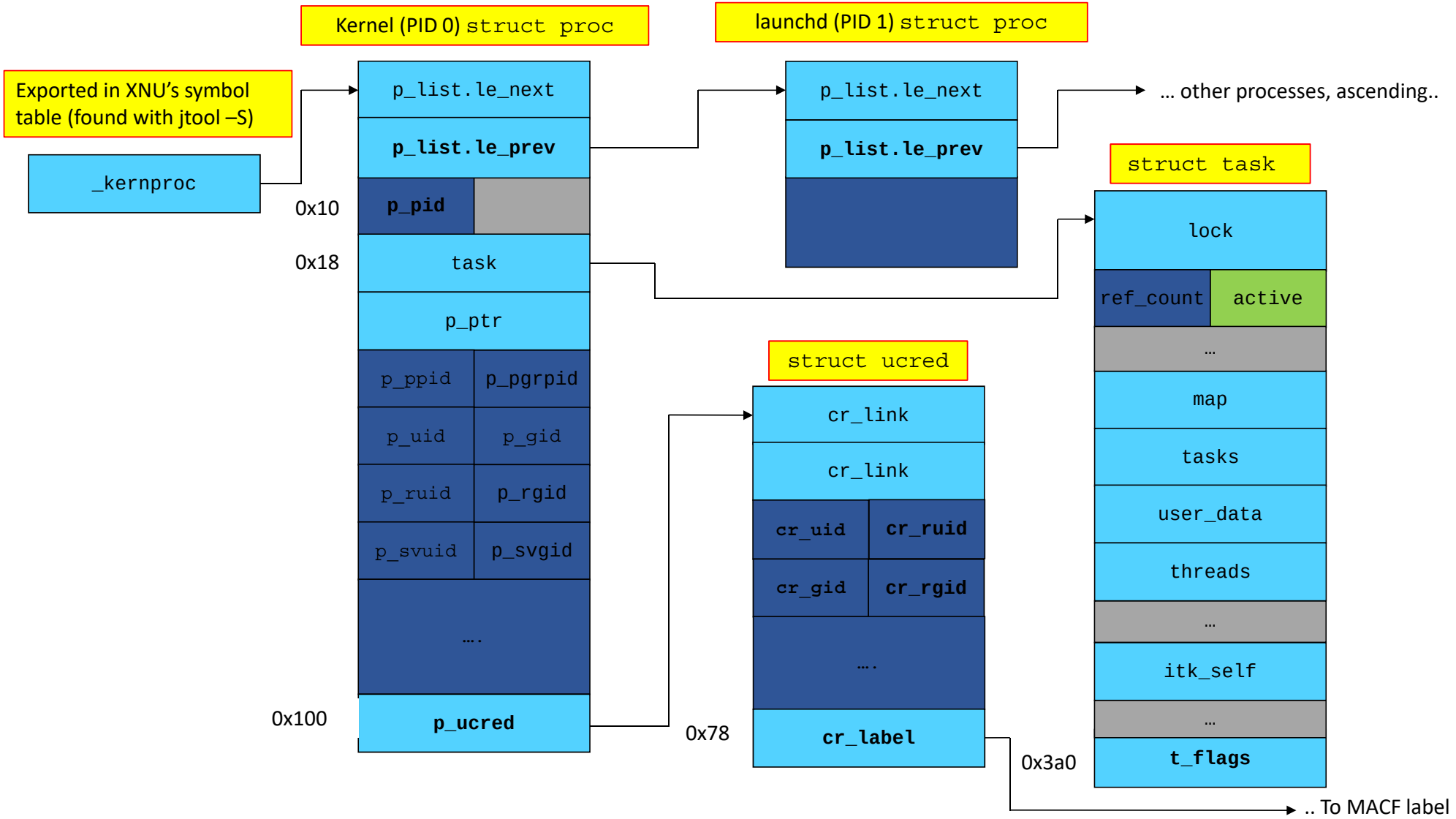
- Symbolicating the kernel:
  - *OS XNU provides some 4,500 symbols, which it must export for kexts
    - By comparison, MacOS XNU has well over 20,000…
  - Some exports very useful (e.g. `_kernproc`, `_rootvnode`), but many missing
  - Most jailbreaks hardcode addresses and slide, which is tedious
    - Requires symbolicating each of the i-Device variants to determine specific addresses..

- Solution: Harness joker's engine (disarm+machlib)
  - Works directly on the in-device /S/L/C/com.apple.kernelcaches/kernelcache
  - Also accessible when app is sandboxed (i.e. pre-exploitation)

Utility Functions

- Structure offsets:
  - Apple continually modifies (and re-orders) proc, task, and other structs
  - Most jailbreaks hard-code field offsets, and complicate code.

- Solution: Reconstruct all kernel headers in user mode
  - Painful, but only needed to do it once…
  - Hopefully forward compatible for whatever XNU-5123 or higher bring..
    - Will just copy kernel headers and recompile


Kernel (PID 0) struct proc
launchd (PID 1) struct proc
Exported in XNU's symbol table (found with jtool –S)
_kernproc
p_list.le_next
p_list.le_prev
p_pid
0x10
0x18
task
p_ptr
p_ppid
p_pgrpid
p_uid
p_gid
p_ruid
p_rgid
p_svuid
p_svgid
….
0x100
p_ucred
p_list.le_next
p_list.le_prev
… other processes, ascending..
struct task
lock
ref_count
active
…
map
tasks
user_data
threads
…
itk_self
…
t_flags
struct ucred
cr_link
cr_link
cr_uid
cr_ruid
cr_gid
cr_rgid
….
0x78
cr_label
0x3a0
.. To MACF label

Utility Functions

- `pid_t findPidOfProcess (char *ProcName)`
  - Get process list
  - Match by `proc_name`(pid,…)
- `uint64_t getProcStructForPid(pid_t Whom);`
  - Start at kernproc (exported symbol) working backwards
  - Check if `p_pid` matches requested PID
  - Return address of struct proc to caller
- `uint64_t getTaskStructForPid(Pid);`
  - `psAddr = getProcStructForPid(Pid);`
  - `readKernelMemory (psAddr, sizeof (struct proc), &p);`
  - `return (p.task);`

Utility Functions

- `uint64_t getAddressOfPort(pid_t Pid, mach_port_name_t P)`
  - `taskStructAddr = getTaskStructForPid(Pid);`
  - Traverse itk_space->is_table, inspecting ipc_entry structs
  - Return address to caller on (`entry.ie_bits >> 24`) and iterator match
  - More reliable than CVE-2017-13865

Utility Functions

- `int setTFP0AsHostSpecialPort4 (void);`
  - Method devised by Pangu Team
  - Allows access to kernel_task for any root owned process

- Optional, and not recommended
  - Greatly compromises system security
  - I needed it for xn00p, also useful for other kernel mem inspection tools

Problem: Remounting root filesystem

- Jailbreaks require rw / partition if they are to achieve:
  - System hard-coded default manipulation
  - Easier persistence
  - Execution of unsandboxed binaries from subdirectories of /

Solution: Flag Flipping

- Method publicized by @Xerub:
  - easily circumvents poor implementation:
    - Remove MNT_ROOTFS flag
    - Remount RW
    - Add MNT_ROOTFS flags
- Already hardened by Apple ☹
  - APFS driver panics
  - Other creative solutions may endure

```
int remountRootFS (void)
{
    // Need these so struct vnode is properly defined:
/* 0x00 */   LIST_HEAD(buflists, buf);
/* 0x10 */   typedef  void *kauth_action_t ;
/* 0x18 */   typedef  struct {
                uint64_t    x[2];
/* 0x28 */   } lck_mtx_t;

#if 0   // Cut/paste struct vnode (bsd/sys/vnode_internal.h) here (omitted for brevity)
    struct vnode {
/* 0x00 */  lck_mtx_t v_lock;                       /* vnode mutex */
/* 0x28 */  TAILQ_ENTRY(vnode) v_freelist;          /* vnode freelist */
/* 0x38 */  TAILQ_ENTRY(vnode) v_mntvnodes;         /* vnodes for mount point */
/* 0x48 */  TAILQ_HEAD(, namecache) v_ncchildren; /* name cache entries that regard us as their
/* 0x58 */  LIST_HEAD(, namecache) v_nclinks;     /* name cache entries that name this vnode */
        ....
/* 0xd8 */  mount_t v_mount;                        /* ptr to vfs we are in */
        ..
    };
    // mount_t (struct mount *) can similarly be obtained from bsd/sys/mount_internal.h
    //  The specific mount flags are a uint32_t at offset 0x70
#endif

    // Why bother with a patchfinder when AAPL still exports this for us? :-)
    uint64_t rootVnodeAddr = findKernelSymbol("_rootvnode");
    uint64_t *actualVnodeAddr;
    struct vnode *rootvnode = 0;
    char *v_mount;

    status("Attempting to remount rootFS...\n");
    readKernelMemory(rootVnodeAddr, sizeof(void *), &actualVnodeAddr);

    readKernelMemory(*actualVnodeAddr, sizeof(struct vnode), &rootvnode);
    readKernelMemory(rootvnode->v_mount, 0x100, &v_mount);

    // Disable  MNT_ROOTFS momentarily, remounts , and then flips the flag back
    uint32_t mountFlags = (*(uint32_t * )(v_mount + 0x70)) &  ~(MNT_ROOTFS | MNT_RDONLY);

    writeKernelMemory(((char *)rootvnode->v_mount) + 0x70 ,sizeof(mountFlags), &mountFlags);

    char *opts = strdup("/dev/disk0s1s1");

    // Not enough to just change the MNT_RDONLY flag - we have to call
    // mount(2) again, to refresh the kernel code paths for mounting..
    int rc =   mount("apfs", "/", MNT_UPDATE, (void *)&opts);

    printf("RC: %d (flags: 0x%x) %s \n", rc, mountFlags, strerror(errno));

    mountFlags |= MNT_ROOTFS;
    writeKernelMemory(((char *)rootvnode->v_mount) + 0x70 ,sizeof(mountFlags), &mountFlags);

    // Quick test:
    int fd = open ("/test.txt", O_TRUNC| O_CREAT);
    if (fd < 0) { error ("Failed to remount /"); }
    else {
        status("Mounted / as read write :-)\n");
        unlink("/test.txt"); // clean up
        }
    return 0;
}
```

Problem: task conversion APIs

- iOS 11 adds task_conversion_eval in order to restrict task port access:

Listing 25-18: The `task_conversion_eval` function (from osfmk/kern/ipc_tt.c)

```
kern_return_t task_conversion_eval(task_t caller, task_t victim)
{
  /*
   * Tasks are allowed to resolve their own task ports, and the kernel is
   * allowed to resolve anyone's task port.
   */
  if (caller == kernel_task) { return KERN_SUCCESS; }

  if (caller == victim) { return KERN_SUCCESS; }
  /*
   * Only the kernel can can resolve the kernel's task port. We've established
   * by this point that the caller is not kernel_task.
   */
   if (victim == kernel_task) { return KERN_INVALID_SECURITY; }
#if CONFIG_EMBEDDED
   /*
    * On embedded platforms, only a platform binary can resolve the task port
    * of another platform binary.
    */
   if ((victim->t_flags & TF_PLATFORM) && !(caller->t_flags & TF_PLATFORM)) {
#if SECURE_KERNEL
       return KERN_INVALID_SECURITY;
#else
    if (cs_relax_platform_task_ports) {
           return KERN_SUCCESS;
       } else { return KERN_INVALID_SECURITY; }
#endif /* SECURE_KERNEL */
    }
#endif /* CONFIG_EMBEDDED */
        return KERN_SUCCESS;
}
```

Solution: Platformization

- `platformizePID(pid_t Blessed)`
  - Locates struct proc, corresponding task, and flips TF_PLATFORM in t_flags (0x3a8)

- Especially useful for `debugserver`:
  - Apple's provided debugserver binary (from DDI) still needs to be resigned:
    - Remove seatbelt-profile
    - Enable task_for_pid-allow and run-unsigned-code (+ platform-application)

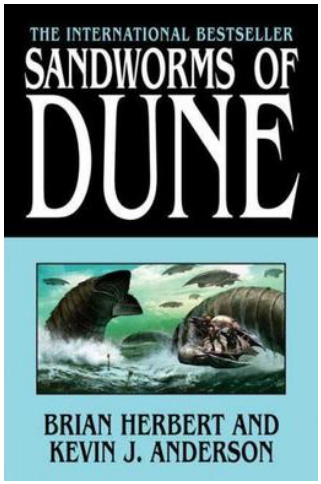
Problem: Sandbox

- 3rd party applications are stringently containerized by sandbox
- Kext nitpicks and inspects 130+/~340 possible MACF hooks
- *OS doesn't have SIP, but platform profile is just as bad:
  - No execution in /tmp, /var (outside containers)
  - Also prevents "untrusted binaries" from being spawned by anyone save launchd

Solution: ShaiHulud

- Most Sandbox checks make exemption for kernel credentials

- Simple idea: Copy kernel credentials over those of process
  - Ok to link instead of copy, since kernproc exits last anyway
  - Impact: immediate unsandboxing
  - (but still subject to platform profile restrictions)



And more sandbox annoyances

- Sandbox platform-profiles restricts “untrusted” binaries to launchd
  - i.e. if you’re not a platform application, your PPID needs to be 1

- To get around:
  - Reparent exec’ed binary to 1 before sandbox hook is hit

    Or
  - Self-sign yourself with `<platform-application> <true/>`.

- Platform profile can still be…uhm.. persuaded, but not by QiLin.

  (not a good idea to blow a useful technique)

Problem: Entitlements

- As of somewhere in iOS 7 or 8, Apple started using entitlements
  - Stored in special blob (#5) inside code signature
  - Loaded into kernel memory (UBC) when code signature is validated
- Since then, number of entitlements has exploded
  - The entitlement database can be used to figure out entitlement holders

Solution (I): Injecting

- With kernel mutable data, we can overwrite UBC easily:
  - `pidAddr = getProcStructForPid(pid)`
  - `blobAddr = LocateCodeSigningBlobForProcAtAddr(pidAddr)`
  - `readKernelMemory (blobAddr)`
  - `Edit csb_entitlements`

- Can verify method works with `csops(2)` call

Solution (I): Injecting

- Not that simple for AMFI-enforced entitlements (e.g. task_for_pid)
  - AMFI.kext stores entitlements OSDictionary in 0-th MACF label slot

Figure 25-14: The AMFI Entitlement dictionary, in its MACF label slot

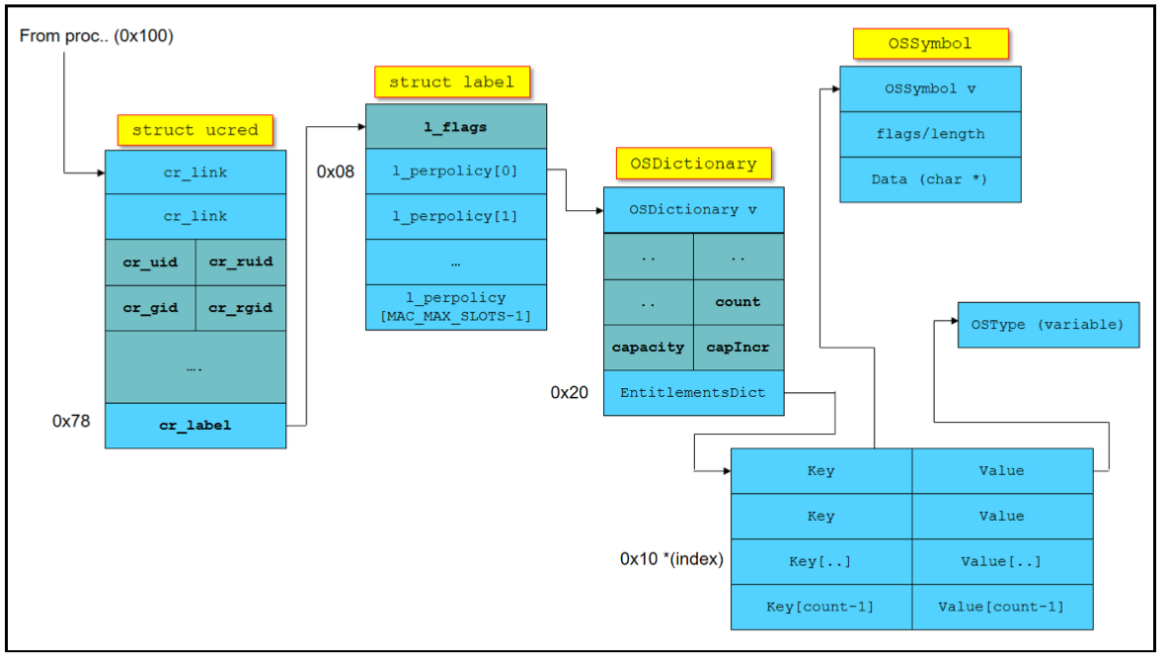
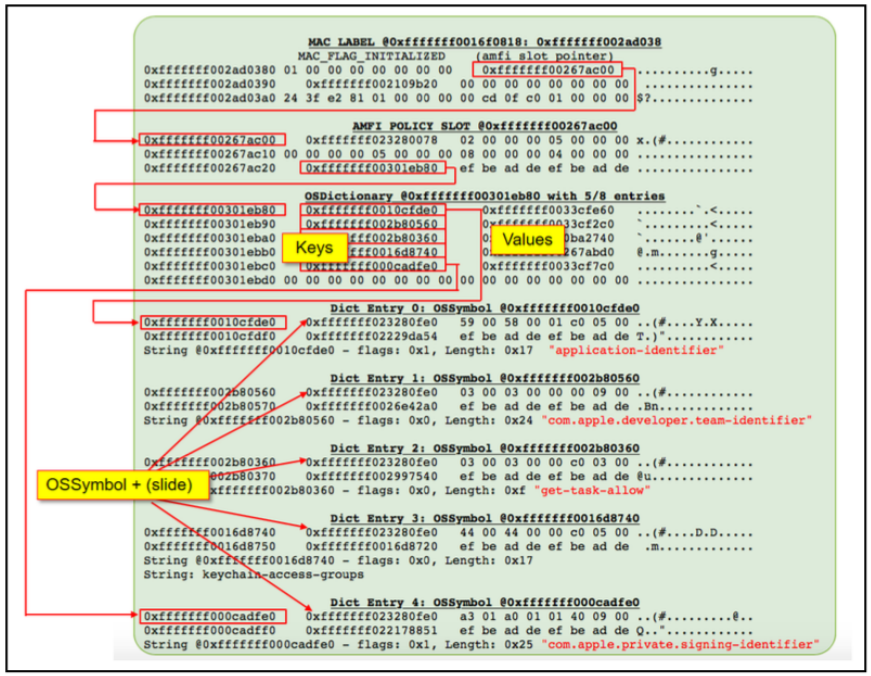


Figure 25-15: The AMFI MAC policy label slot, revealed



Solution (II): Borrowing!

- But wait – Apple already provides entitled, signed binaries!
- MUCH simpler to:
  - spawn entitled binary (possibly suspended) if not already executing
  - Locate struct proc entry
  - Copy over kauth creds (instant uid/gid)
  - Get MACF labels (AMFI (0), Sandbox(1)) for free!
  - Profit
  - Good practice: Recover original credentials
- Like ShaiHulud, but on user-mode processes

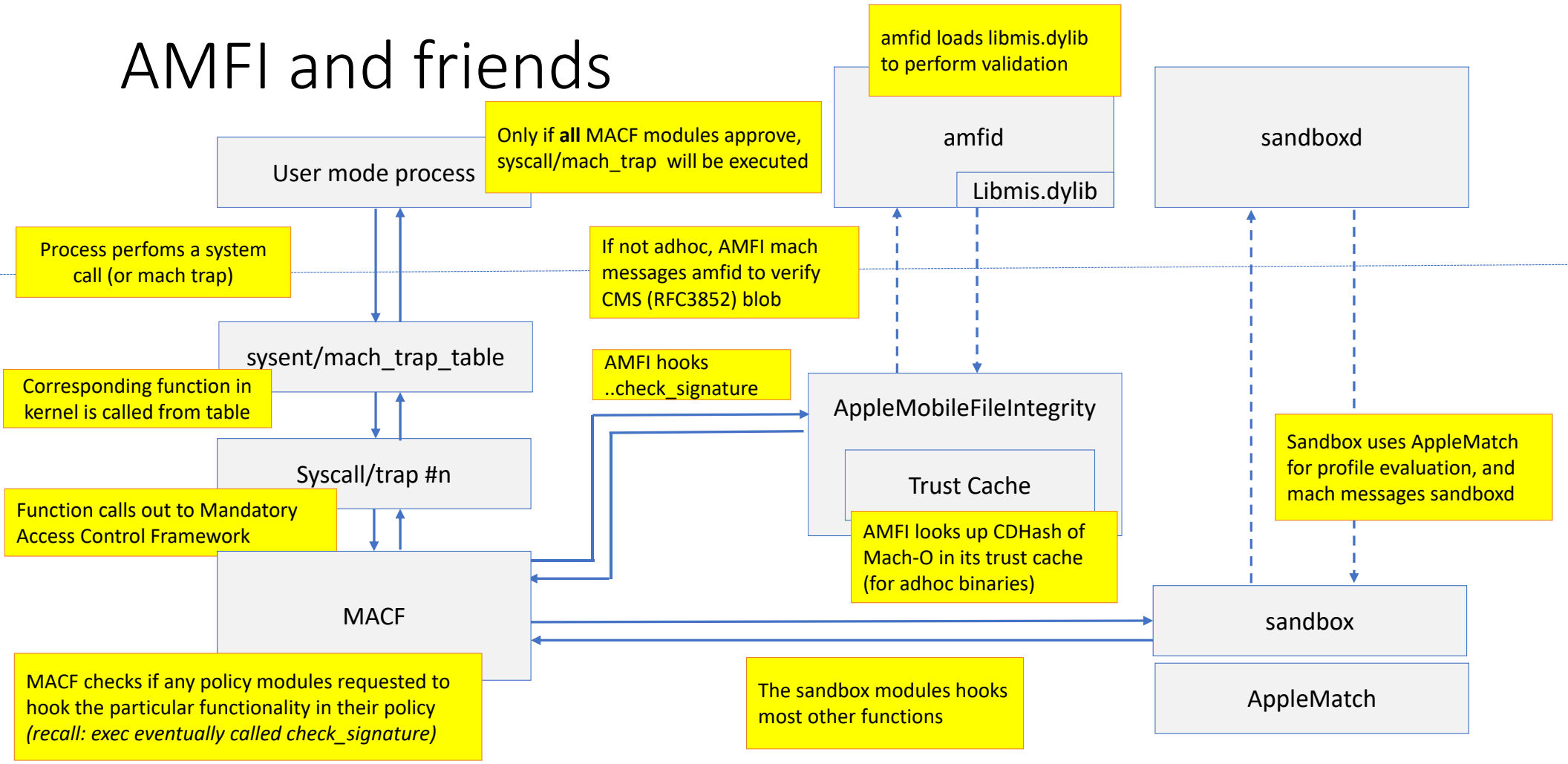
Figure 25-17: Borrowing entitlements from `sysdiagnose(1)`

```
int sdPID = execCommand("/usr/bin/sysdiagnose", "-u", NULL, NULL,NULL,NULL);

rc = kill (sdPID, SIGSTOP); // Not really necessary, but safer...
uint64_t *sdCredAddr ;

// Find our donor's process struct in memory
uint64_t sdProcStruct = processProcessList(sdPID);

// Read donor's credentials
readKernelMemory(sdProcStruct + offsetof(struct proc, p_ucred),
                 sizeof(void *),
                 &sdCredAddr);

// Usurp donor's credentials
uint64_t origCreds = ShaiHuludMe(*sdCredAddr);

    ...
/* Perform operation, e.g. task_for_pid() */

    ...

/* Revert to original credentials */

kill (sdPID, SIGKILL); // Don't need our donor anymore - thanks, sucker!
ShaiHuludMe(origCreds);
```

Problem: AMFI

- Sworn nemesis of jailbreakers everywhere, enforces code signing
- MACF …execve hook called on every process execution
- KEXT validates ad-hoc signatures against trust cache(s)
- User-mode lackey daemon (amfid) validates third party signatures

AMFI and friends



Solution (I): Overwrite loadable trust cache

- AMFI maintains a “trust cache” for Apple’s own platform binaries
  - Part of __TEXT, so therefore read-only and subject to patch protection
- But.. AMFI also (foolishly) maintains *another* trust cache:
  - Used for DeveloperDiskImage binaries
  - Loaded by mobile storage mounter (with entitlement)
  - By definition, resides in mutable memory!
- Method publicized by Xerub injects CDHashes into (other) cache
  - As a bonus, binaries automatically bestowed platform status
- MUST go away in iOS 12
  - (fool me once, shame on me.. Fool me five times … enough already!)

Solution (II): AMFI-Debilitate

- For third party binaries, validation is done in user mode amfid
- Dimwit daemon outsources decision making to `libmis.dylib`
  - Traditionally, MISValidateSignature would perform complex validation
    - Certificate check, UPPs, online_auth_agent, etc… and…
    - just return 0/1 ☺
    - Bypassed numerous times, from evasi0n 6 to Pangu 9.3 (女娲石)
  - As of iOS 10, `MISValidateSignatureAndCopyInfo` also populates hash
  - Still trivial to get by, as demonstrated by Ian Beer's mach_portal
    - Hijack AMFId's exception ports (or inject exception thread into it)
    - Overwrite `MISValidateSignatureAndCopyInfo la_symbol_ptr`

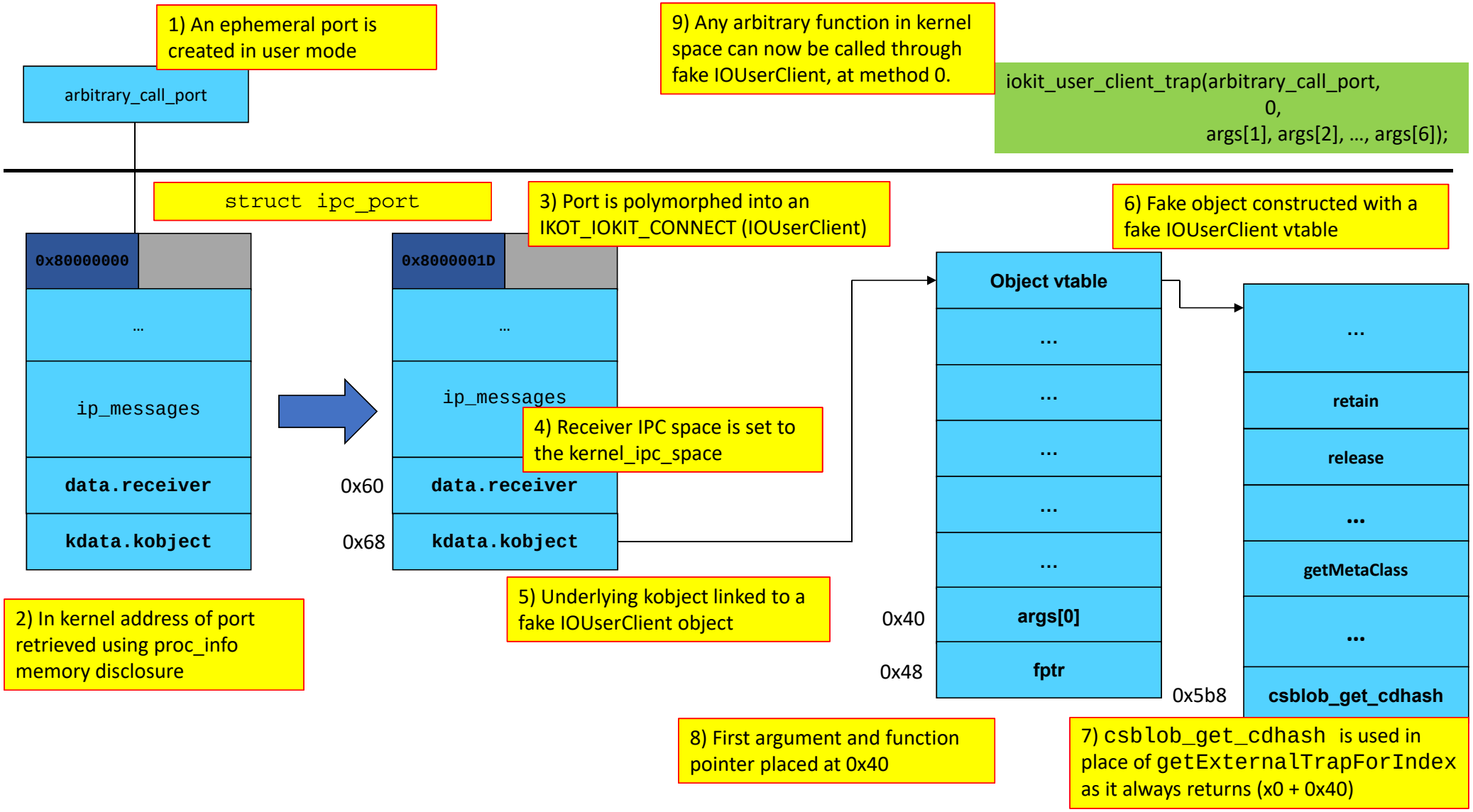
Solution (II): AMFI-Debilitate

- QiLin's `int castrateAmfid(void)` automatically does all this..
  - Amfid marked `CS_HARD|CS_KILL`, but who cares when you can overwrite?
- AMFIdebilitate daemon can persist after Liber* JB-Apps exit:
  - Spawned as platformized binary by QiLin (provided in binpack tar)
  - Hijacks amfid's exception ports to hook MVSACI
  - Also registers knote on amfid (to track exit due to launchd kill)
- NOT a jailbreakd, but can be adapted easily to one:
  - AMFId's upcall follows MACF …execve hook, so great for process notifications

Problem: kernel execution primitive

- Advanced exploitation relies on kernel execution, e.g.
  - kalloc()..
  - pmem APIs..
  - General ability to invoke kernel functions with arbitrary arguments

- Ian Beer provides a great method for exec but..
  - Relies on proc_pidinfo(LISTUPTRS) which only works for 11.0-11.1.2


1) An ephemeral port is created in user mode
arbitrary_call_port
9) Any arbitrary function in kernel space can now be called through fake IOUserClient, at method 0.
iokit_user_client_trap(arbitrary_call_port,
0,
args[1], args[2], …, args[6]);
struct ipc_port
3) Port is polymorphed into an IKOT_IOKIT_CONNECT (IOUserClient)
6) Fake object constructed with a fake IOUserClient vtable
0x80000000
…
ip_messages
data.receiver
kdata.kobject
0x8000001D
…
ip_messages
0x60
data.receiver
0x68
kdata.kobject
4) Receiver IPC space is set to the kernel_ipc_space
Object vtable
…
…
…
…
…
0x40
args[0]
0x48
fptr
…
retain
release
…
getMetaClass
…
0x5b8
csblob_get_cdhash
2) In kernel address of port retrieved using proc_info memory disclosure
5) Underlying kobject linked to a fake IOUserClient object
8) First argument and function pointer placed at 0x40
7) csblob_get_cdhash is used in place of getExternalTrapForIndex as it always returns (x0 + 0x40)

Solution: Kernel-version agnostic kexec()

- Method very similar to Beer's:
  - Create an arbitrary IOUserClient (AMFI chosen for a touch of irony)
  - `getAddressOfPort(pid_t Pid, mach_port_name_t IOUserClientName);`
  - Clone IOUserClient object in memory to fully writable memory
  - Dynamically modify object's vtable entry to allow any function & arguments

- Especially useful with kernel symbols
  - Can call any function known to joker (and that's most useful ones)

Take away: A full set of reusable tools

- Put all of these together, and you have a full jailbreak, or more*
- Use it (subject to minimal license), submit bugs/requests, help improve
- Utility functions will be forward compatible indefinitely
- Protection workarounds will likely be closed by Apple at some point.

* - But still, no Cydia ☺

Message to Apple

- Valiant efforts, guys, but **NONE OF IT IS ACTUALLY ANY USEFUL**
- **You're just making it a pain to JB, but not solving real threat – APTs**
  - Unsandboxed uid 501 (mobile) is usually enough for most targeted malware
- **Your bug bounty program is an insult**
  - 50k for an exploit chain that fetches x100 times that in open market?
- **Open up *OS for researchers and they'll beat a path to your door**
  - Also iron out some design flaws in an otherwise superbly writ OS

Greets

- The Jailbreaking community, especially @PanguTeam & @S1guza
- @pimskeks – A giant walks among us
- @Xerub – stop open sourcing everything and killing good methods ☺
- @i41nbeer – A brilliant mind working for the wrong people

All this and more in..

- *OS Internals Trilogy, specifically Volume III
  - http://NewOSXBook.com/
  - Volume II coming later this year with Darwin 18! (MacOS 14/iOS 12)

- Technologeeks.com training:

  - MacOS/iOS Internals – Reverse Engineer's Perspective – 5 day deep dive
    - http://Technologeeks.com/OSXRE

  - *OS (in)security – 3 day, applied MacOS and iOS hacking
    - http://Technologeeks.com/xOSSec
    - Also coming to Vegas right before BlackHat in a special 2-day bootcamp edition!