Post-talk note:

If you read my Zer0Con abstract, you'll see that I originally intended to finish this talk with a case study on my IOHIDeous exploit. I overdid it a bit though, and if I had gone through with everything in the abstract, my talk would've been at least 1.5h, so I had to cut some bits. I figured since the complete write-up for IOHIDeous was already public anyway, cutting that part would be the smallest loss. As a consequence, the title "The HIDeous parts of IOKit" has been deprived its context and doesn't make much sense anymore, and the talk has turned more into a "how I do iOS analysis and exploitation".

I hope you'll enjoy it anyway.

The HIDeous parts of IOKit

whoami

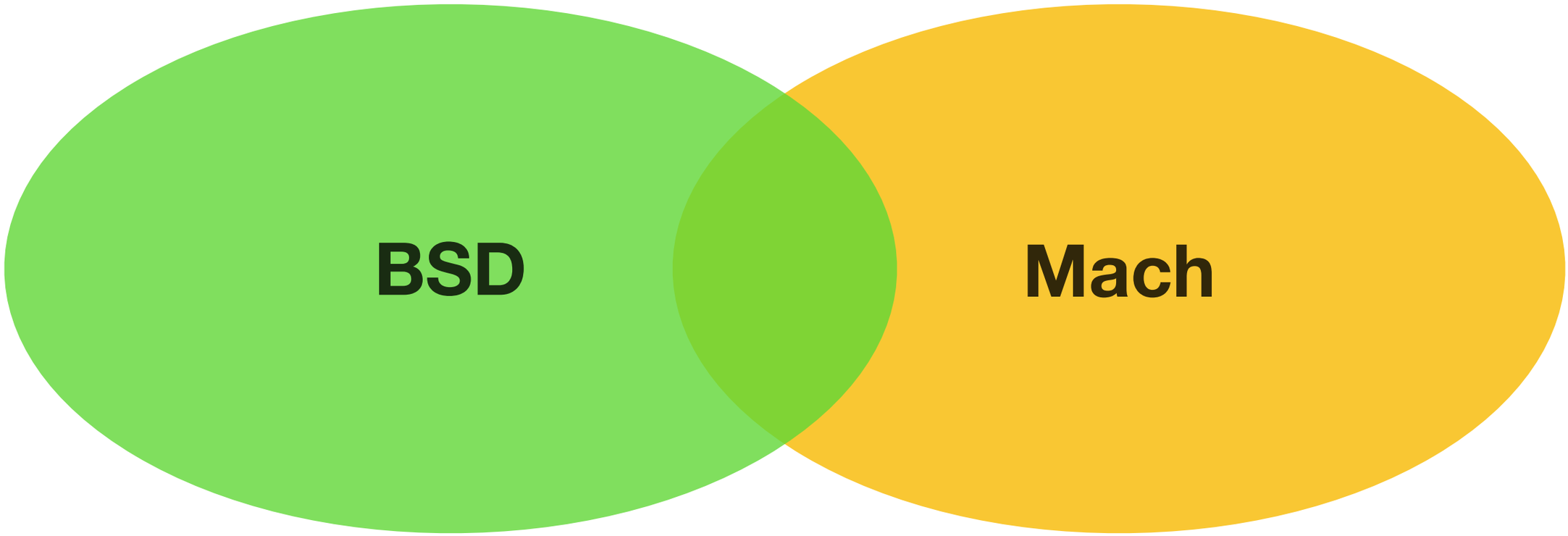
- Hobbyist hacker from Switzerland, 23 years old
- Currently studying for Computer Science Bachelor @ETH Zürich
- Began messing with code at the age of 11
- Got started with hacking in fall 2016 ("cl0ver")
- Involved with several jailbreaks since then
- Primarily focused on iOS/macOS kernel hacking

This talk

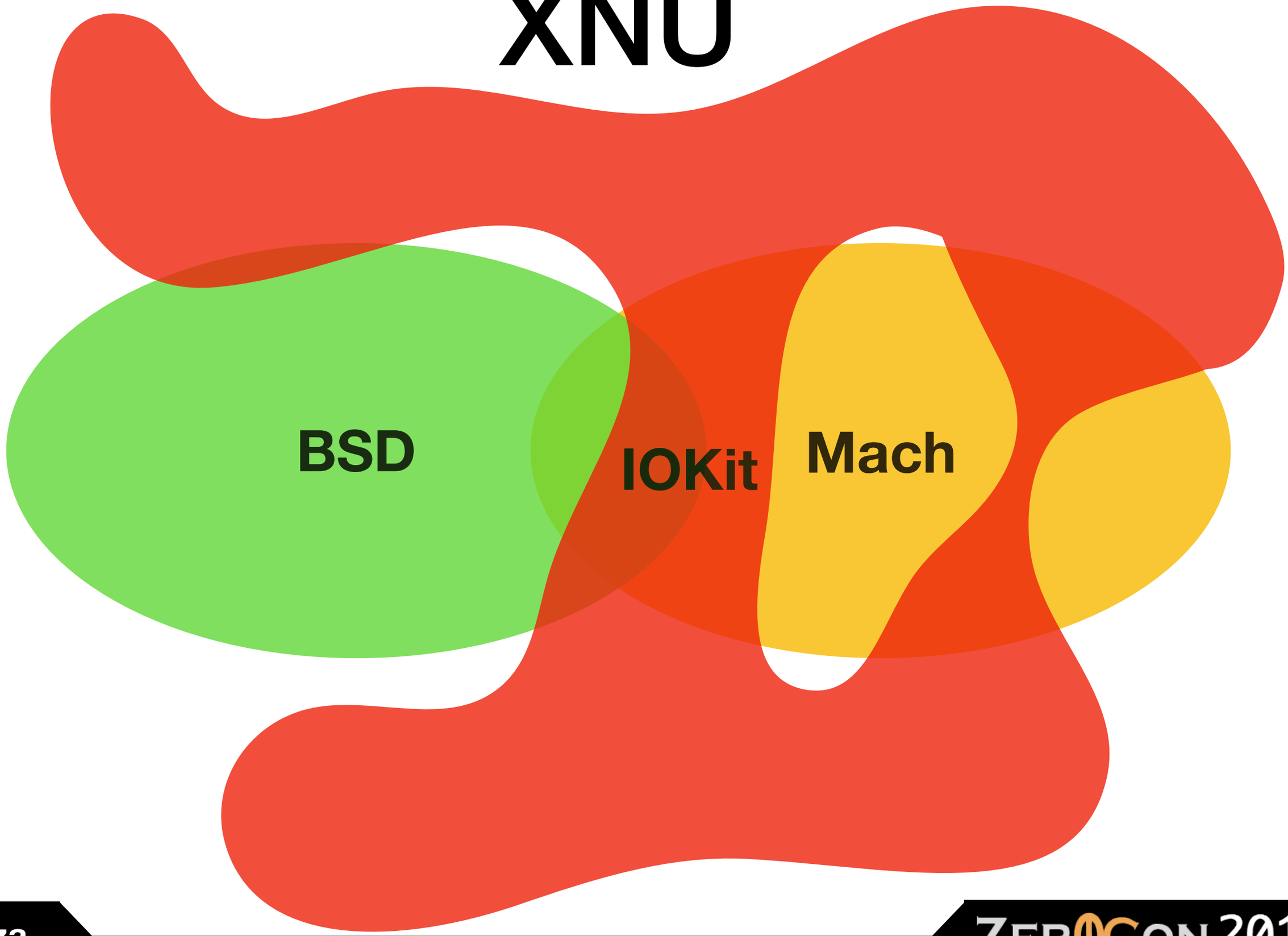
1. IOKit overview
2. Attack surface (& security checks)
3. Bugs (& mitigations)
4. Exploit strategies

IOKit overview

XNU



XNU



What *is* IOKit?

Apple:

"The I/O Kit is a collection of system frameworks, libraries, tools, and other resources for creating device drivers in OS X."

(You *totally* know more now, right?)

What *is* IOKit?

From a developer's perspective:

- A framework for writing kernel extensions
- An official API available to 3rd parties
  (at least on macOS)
- Documented ("I/O Kit fundamentals"):
  https://developer.apple.com/library/content/documentation/DeviceDrivers/Conceptual/IOKitFundamentals/

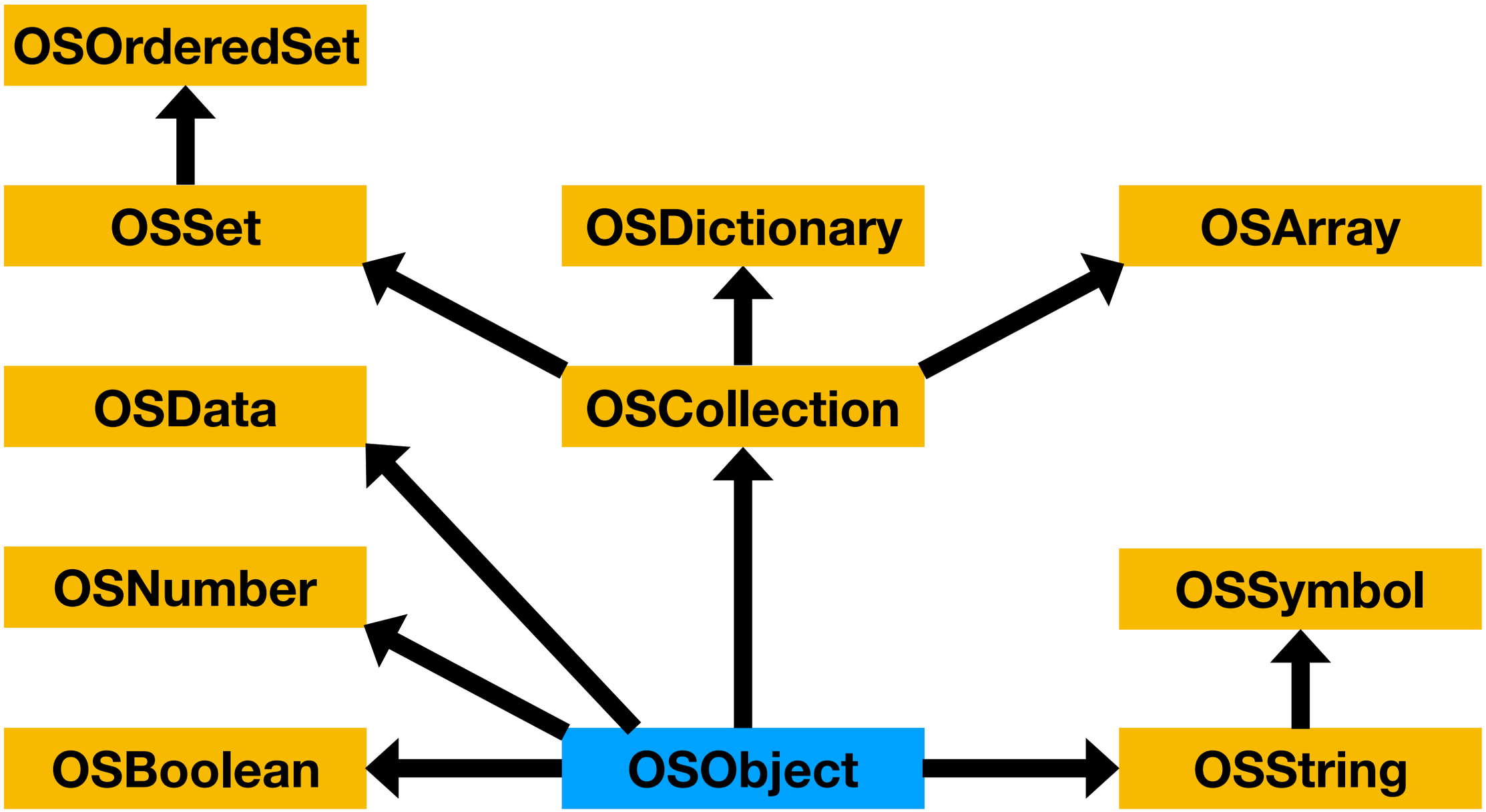
IOKit architecture

- Written in a subset of C++ (without multiple inheritance, templates, and exceptions)
- Universal base class: OSObject
  - Allocated via kalloc (overloaded new and delete operators)
  - Lifetime managed by reference counting
- Custom RTTI: "OSMetaClass"

Libkern

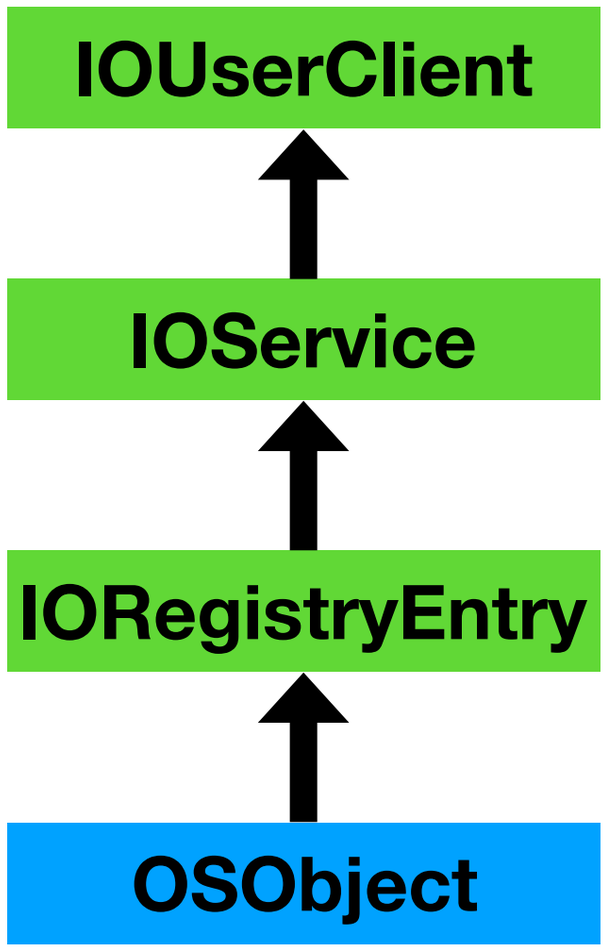
- Serves as standard library to some extent
- Contains classes and functions for Plist serialization & deserialization
  - Serializability is a core feature that extends to all classes (::serialize())
  - Plist configs are extensively used (e.g. entitlements, kexts, services, ...)
- A few more core features like OSIterator or OSKext

Libkern: Plist classes



IORegistry

- Global lookup tree of IORegistryEntry objects
- Has multiple "planes" (basically just separate registries)
- Every entry can have child nodes and a set of "properties"
- Entries can directly be interacted with from userland!



IOKit from userland

- Public API only frameworks like IOSurface, CoreVideo, Security, ...
- Private API: IOKit.framework
- Even more private API: MIG

IOKit.framework

- Open Source & Documented:
  https://opensource.apple.com/tarballs/IOKitUser
  https://developer.apple.com/documentation/iokit
- Symbols exported, can be linked against
- Headers not present in iOS SDK (but whatever)
- Bridge between MIG and CoreFoundation
  - Handles serialization & deserialization
  - Other convenience features
- Stable API

IOKit.framework

```
kern_return_t ret;
io_service_t service = MACH_PORT_NULL;
io_connect_t client = MACH_PORT_NULL;
CFDictionaryRef match = NULL;

match = IOServiceMatching("IOSurfaceRoot");
service = IOServiceGetMatchingService(kIOMasterPortDefault, match);
ret = IOServiceOpen(service, mach_task_self(), 0, &client);
```

MIG

- Statically compiled into IOKit.framework
- Symbols exist but only exported on 32bit, cannot be linked against on 64bit
- Can be generated with mig utility and xnu/osfmk/device/device.defs (or written by hand)
- Verbose, not really fun to deal with
- Unstable API, major releases almost always break compatibility
- Higher performance (no CF type overhead)
- Full control over serialized (binary) data

MIG

```
static kern_return_t reallocate_buf(io_connect_t client, uint32_t surfaceId, uint32_t propertyId, void *buf, mach_vm_size_t len)
{
#pragma pack(4)
    typedef struct {
        mach_msg_header_t Head;
        NDR_record_t NDR;
        uint32_t selector;
        mach_msg_type_number_t scalar_inputCnt;
        mach_msg_type_number_t inband_inputCnt;
        uint32_t inband_input[4];
        mach_vm_address_t ool_input;
        mach_vm_size_t ool_input_size;
        mach_msg_type_number_t inband_outputCnt;
        mach_msg_type_number_t scalar_outputCnt;
        mach_vm_address_t ool_output;
        mach_vm_size_t ool_output_size;
    } DeleteRequest;
    typedef struct {
        mach_msg_header_t Head;
        NDR_record_t NDR;
        uint32_t selector;
        mach_msg_type_number_t scalar_inputCnt;
        mach_msg_type_number_t inband_inputCnt;
        mach_vm_address_t ool_input;
        mach_vm_size_t ool_input_size;
        mach_msg_type_number_t inband_outputCnt;
        mach_msg_type_number_t scalar_outputCnt;
        mach_vm_address_t ool_output;
        mach_vm_size_t ool_output_size;
    } SetRequest;
    typedef struct {
        mach_msg_header_t Head;
        NDR_record_t NDR;
        kern_return_t RetCode;
        mach_msg_type_number_t inband_outputCnt;
        char inband_output[4096];
        mach_msg_type_number_t scalar_outputCnt;
        uint64_t scalar_output[16];
        mach_vm_size_t ool_output_size;
        mach_msg_trailer_t trailer;
    } Reply;
#pragma pack()

    // Delete
    union {
        DeleteRequest In;
        Reply Out;
    } DMess;

    DeleteRequest *DInP = &DMess.In;
    Reply *DOutP = &DMess.Out;

    DInP->NDR = NDR_record;
    DInP->selector = IOSURFACE_DELETE_VALUE;
    DInP->scalar_inputCnt = 0;

    DInP->inband_input[0] = surfaceId;
    DInP->inband_input[1] = transpose(propertyId);
    DInP->inband_input[3] = 0x0; // Null terminator
    DInP->inband_inputCnt = sizeof(DInP->inband_input);

    DInP->ool_input = 0;
    DInP->ool_input_size = 0;

    DInP->inband_outputCnt = sizeof(uint32_t);
    DInP->scalar_outputCnt = 0;
    DInP->ool_output = 0;
    DInP->ool_output_size = 0;

    DInP->Head.msgh_bits = MACH_MSGH_BITS(19, MACH_MSG_TYPE_MAKE_SEND_ONCE);
    DInP->Head.msgh_remote_port = client;
    DInP->Head.msgh_local_port = mig_get_reply_port();
    DInP->Head.msgh_id = 2065;
    DInP->Head.msgh_reserved = 0;

    // Set
    union {
        SetRequest In;
        Reply Out;
    } SMess;

    SetRequest *SInP = &SMess.In;
    Reply *SOutP = &SMess.Out;

    SInP->NDR = NDR_record;
    SInP->selector = IOSURFACE_SET_VALUE;
    SInP->scalar_inputCnt = 0;

    SInP->inband_inputCnt = 0;

    SInP->ool_input = (mach_vm_address_t)buf;
    SInP->ool_input_size = len;

    SInP->inband_outputCnt = sizeof(uint32_t);
    SInP->scalar_outputCnt = 0;
    SInP->ool_output = 0;
    SInP->ool_output_size = 0;

    SInP->Head.msgh_bits = MACH_MSGH_BITS(19, MACH_MSG_TYPE_MAKE_SEND_ONCE);
    SInP->Head.msgh_remote_port = client;
    SInP->Head.msgh_local_port = mig_get_reply_port();
    SInP->Head.msgh_id = 2065;
    SInP->Head.msgh_reserved = 0;

    // Deep breath
    sched_yield();

    // Fire
    kern_return_t ret = mach_msg(&DInP->Head, MACH_SEND_MSG|MACH_RCV_MSG|MACH_MSG_OPTION_NONE, sizeof(DeleteRequest), (mach_msg_size_t)sizeof(Reply), DInP->Head.msgh_local_port, MACH_MSG_TIMEOUT_NONE, MACH_PORT_NULL);
    if(ret == KERN_SUCCESS)
    {
        ret = DOutP->RetCode;
    }
    if(ret != KERN_SUCCESS)
    {
        return ret;
    }
    ret = mach_msg(&SInP->Head, MACH_SEND_MSG|MACH_RCV_MSG|MACH_MSG_OPTION_NONE, sizeof(SetRequest), (mach_msg_size_t)sizeof(Reply), SInP->Head.msgh_local_port, MACH_MSG_TIMEOUT_NONE, MACH_PORT_NULL);
    if(ret == KERN_SUCCESS)
    {
        ret = SOutP->RetCode;
    }
    return ret;
}
```

Just two function calls to IOConnectCallMethod().

Mach ports

IORegistry based on 3 types:

- IKOT_MASTER_DEVICE
- IKOT_IOKIT_OBJECT
- IKOT_IOKIT_CONNECT

IKOT_MASTER_DEVICE

- Exactly one instance: master_device_port
- Can be obtained via host_get_io_master()
- Used to look up registry entries
- Also offers info queries like API version, class hierarchy, ...

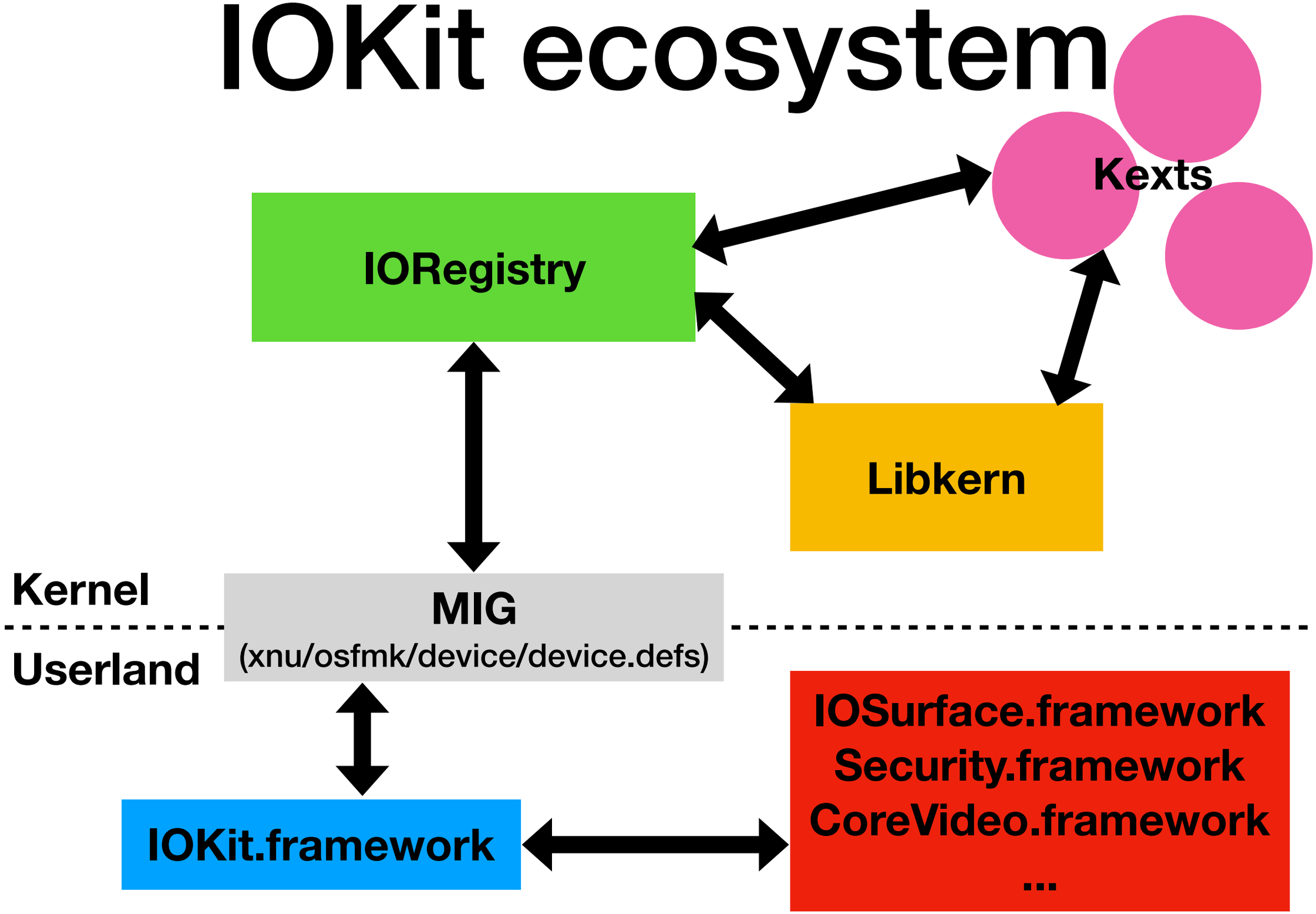
IKOT_IOKIT_OBJECT

- Represents a single IORegistryEntry object
- Can be obtained for every registry entry
- Mostly used in a readonly manner like traversing the tree, querying info, ...
  - Notable exception: setting properties
- Also used for things like OSIterator, IOUserNotification...
- Access is deemed non-privileged

IKOT_IOKIT_CONNECT

- Represents a single IOUserClient object
- Can only be obtained by actually creating a new user client via IOServiceOpen
- Allows access to the "real" functionality:
  - Calling "external methods"
  - Mapping kernel-user shared memory
  - Registering callback ports
  - ...
- Access is usually considered privileged

IOKit ecosystem



What *is* IOKit?

From a hacker's perspective:

- An official and thus very stable API
- An easily queryable interface
- Much more flexible than C code
- A gateway to a ton of kexts
- A mess :P

Attack surface

Dynamic analysis: enumeration

- Getting all available services is trivial:
  Just match against IOService
- Registry tree can be visualised with Apple's own ioreg utility
- Can explore other aspects with my iokit-utils:
  https://github.com/Siguza/iokit-utils
- List of all classes can be obtained from registry root's properties (ioprint -d Root)

ioreg

```
+-o RP03@1C,2  <class IOPCIDevice, id 0x1000001d6, registered, matched, active, busy 0 (465 ms), retain 12>
| +-o IOPP  <class IOPCI2PCIBridge, id 0x100000263, registered, matched, active, busy 0 (458 ms), retain 8>
|   +-o ARPT@0  <class IOPCIDevice, id 0x1000001d7, registered, matched, active, busy 0 (458 ms), retain 12>
|     +-o AirPort_Brcm4360  <class AirPort_Brcm4360, id 0x10000026f, registered, matched, active, busy 0 (20 ms), retain 37>
|       +-o CCLogPipe  <class CCLogPipe, id 0x1000002f5, registered, matched, active, busy 0 (0 ms), retain 10>
|       | +-o CCIOReporterLogStream  <class CCIOReporterLogStream, id 0x1000002f6, registered, matched, active, busy 0 (0 ms), retain 7>
|       +-o CCLogPipe  <class CCLogPipe, id 0x100000301, registered, matched, active, busy 0 (0 ms), retain 13>
|       | +-o CCIOReporterLogStream  <class CCIOReporterLogStream, id 0x100000305, registered, matched, active, busy 0 (0 ms), retain 7>
|       | +-o CCIOReporterLogStream  <class CCIOReporterLogStream, id 0x100000307, registered, matched, active, busy 0 (0 ms), retain 7>
|       +-o CCLogPipe  <class CCLogPipe, id 0x10000030d, registered, matched, active, busy 0 (0 ms), retain 10>
|       | +-o CCLogStream  <class CCLogStream, id 0x10000030e, registered, matched, active, busy 0 (0 ms), retain 7>
|       +-o CCDataPipe  <class CCDataPipe, id 0x10000030f, registered, matched, active, busy 0 (0 ms), retain 10>
|       | +-o CCDataStream  <class CCDataStream, id 0x100000310, registered, matched, active, busy 0 (0 ms), retain 7>
|       +-o en1  <class AirPort_Brcm4360_Interface, id 0x100000311, registered, matched, active, busy 0 (15 ms), retain 11>
|       | +-o IONetworkStack  <class IONetworkStack, id 0x1000002e8, registered, matched, active, busy 0 (0 ms), retain 14>
|       |   +-o IONetworkStackUserClient  <class IONetworkStackUserClient, id 0x1000003e1, !registered, !matched, active, busy 0, retain 5>
|       +-o CCDataPipe  <class CCDataPipe, id 0x1000003ee, registered, matched, active, busy 0 (9 ms), retain 10>
|       | +-o CCIOReporterDataStream  <class CCIOReporterDataStream, id 0x1000003ef, registered, matched, active, busy 0 (9 ms), retain 7>
|       +-o AirPort_Brcm4360_P2PInterface  <class AirPort_Brcm4360_P2PInterface, id 0x1000005ae, registered, matched, active, busy 0 (0 ms), retain 9>
|       +-o CCLogPipe  <class CCLogPipe, id 0x1000005b0, registered, matched, active, busy 0 (0 ms), retain 10>
|       | +-o CCLogStream  <class CCLogStream, id 0x1000005b1, registered, matched, active, busy 0 (0 ms), retain 7>
|       +-o AirPort_Brcm4360_P2PInterface  <class AirPort_Brcm4360_P2PInterface, id 0x1000005b2, registered, matched, active, busy 0 (0 ms), retain 9>
```

Tells you which IOService a certain IOUserClient belongs to.

iokit-utils

- ioclass:
  - Query class hierarchy and origin (kext ID)
- ioprint:
  - Filter registry entries by type
  - Optionally get/set properties
- ioscan:
  - Spawning user clients

Dynamic analysis: debugging

- Class list also contains number of instances of each class
- io_object_get_retain_count()
- Syslog often has useful information

Dynamic analysis: downsides

- Anything that requires input:
  - User client type: *usually* `0`, but not always:
    - IONetworkUserClient: `0xff000001`
    - What properties can be set
    - What methods are overridden
    - External methods
- Kernel internals (e.g. object size)
- Anything not reachable from the sandbox

Restricted access

Three MACF checks in IOKit APIs:

- Setting properties
- Getting properties
- Spawning user clients

Most other checks are specific to a single service, and check for root or entitlement.

Properties

- Outside the sandbox: entirely unrestricted
- Inside the sandbox:
  - Setting only allowed on a single class: IOHIDEventServiceFastPathUserClient
  - Getting allowed on a few more classes, but restricted to individual properties

Spawning user clients

- Outside the sandbox: very few restrictions, only with highly critical services (e.g. SEP)
- Inside the sandbox: rather few whitelisted, but could be worse:

AGXDevice
AppleJPEGDriverUserClient
AppleKeyStoreUserClient
IOAccelContext
IOAccelContext2
IOAccelDevice
IOAccelDevice2
IOAccelSharedUserClient
IOAccelSharedUserClient2
IOAccelSubmitter2
IOHIDEventServiceFastPathUserClient
IOHIDLibUserClient
IOMobileFramebufferUserClient
IOSurfaceAcceleratorClient
IOSurfaceRootUserClient
IOSurfaceSendRight
RootDomainUserClient

Static analysis

On macOS:

Trivial, just run: nm -U kext | c++filt

Static analysis

On iOS:

Kexts have no symbol table, only the kernel does

- Can find all calls to OSMetaClass constructor
- OSMetaClass::alloc reveals vtable
- Not perfect, but can reconstruct class hierarchy, object size and all overridden methods!

iometa

```
[$ iometa -Csov IOHIDLibUserClient kernel
vtab=0xffffffff006e85b50 size=0x00000150 IOHIDLibUserClient
      1 func=0xffffffff00650708c overrides=0xffffffff0074bf284 IOHIDLibUserClient::~IOHIDLibUserClient()
      7 func=0xffffffff0065070a4 overrides=0xffffffff0074bf2dc IOHIDLibUserClient::getMetaClass() const
     13 func=0xffffffff0065081f0 overrides=0xffffffff0074bf3b4 IOHIDLibUserClient::free()
     76 func=0xffffffff0065081a0 overrides=0xffffffff00746f348 IOHIDLibUserClient::didTerminate()
     85 func=0xffffffff0065078d4 overrides=0xffffffff00746fb68 IOHIDLibUserClient::start()
     86 func=0xffffffff006507ce8 overrides=0xffffffff00746fb70 IOHIDLibUserClient::stop()
    107 func=0xffffffff0065098b4 overrides=0xffffffff007470964 IOHIDLibUserClient::attach()
    131 func=0xffffffff0065082d8 overrides=0xffffffff0074719b0 IOHIDLibUserClient::message()
    167 func=0xffffffff006507f18 overrides=0xffffffff0074bf4ec IOHIDLibUserClient::externalMethod()
    170 func=0xffffffff006507818 overrides=0xffffffff0074c085c IOHIDLibUserClient::initWithTask()
    171 func=0xffffffff0065078b0 overrides=0xffffffff0074c0928 IOHIDLibUserClient::clientClose()
    174 func=0xffffffff0065083c0 overrides=0xffffffff0074c0978 IOHIDLibUserClient::registerNotificationPort()
    177 func=0xffffffff006508570 overrides=0xffffffff0074c099c IOHIDLibUserClient::clientMemoryForType()
    185 func=0xffffffff006508358 overrides=0x0000000000000000 IOHIDLibUserClient::fn_0x5c8()
    186 func=0xffffffff006508404 overrides=0x0000000000000000 IOHIDLibUserClient::fn_0x5d0()
```

https://github.com/Siguza/iometa

Static analysis

Benefits:

- Find every possible code path
- Can be done without root access/shell
- Usually not even hard to do manually, thanks to RTTI and verbose logging

Downside:

- Hard to automate

A few numbers

N° of kexts: 179

N° of kexts with no classes: 10

N° of classes: 1559

N° extending IOService (w/o IOUC): 752

N° extending IOUserClient: 111

N° spawnable from sandbox: 17

N° spawnable by fuzzing: 36

Attack surface conclusion

- Inside the sandbox:
  - Limited attack surface
  - Accessible parts very well tested
- Outside the sandbox:
  - Huge attack surface
  - Some kexts were written like shit
  - => Big potential :P

How to find bugs?

Fuzzing?

- Easy and tempting
- Good to find missing input validation
  - but only if input is directly used
  - hardly ever happens in IOKit
  - Apple is fuzzing too, especially services reachable from the sandbox

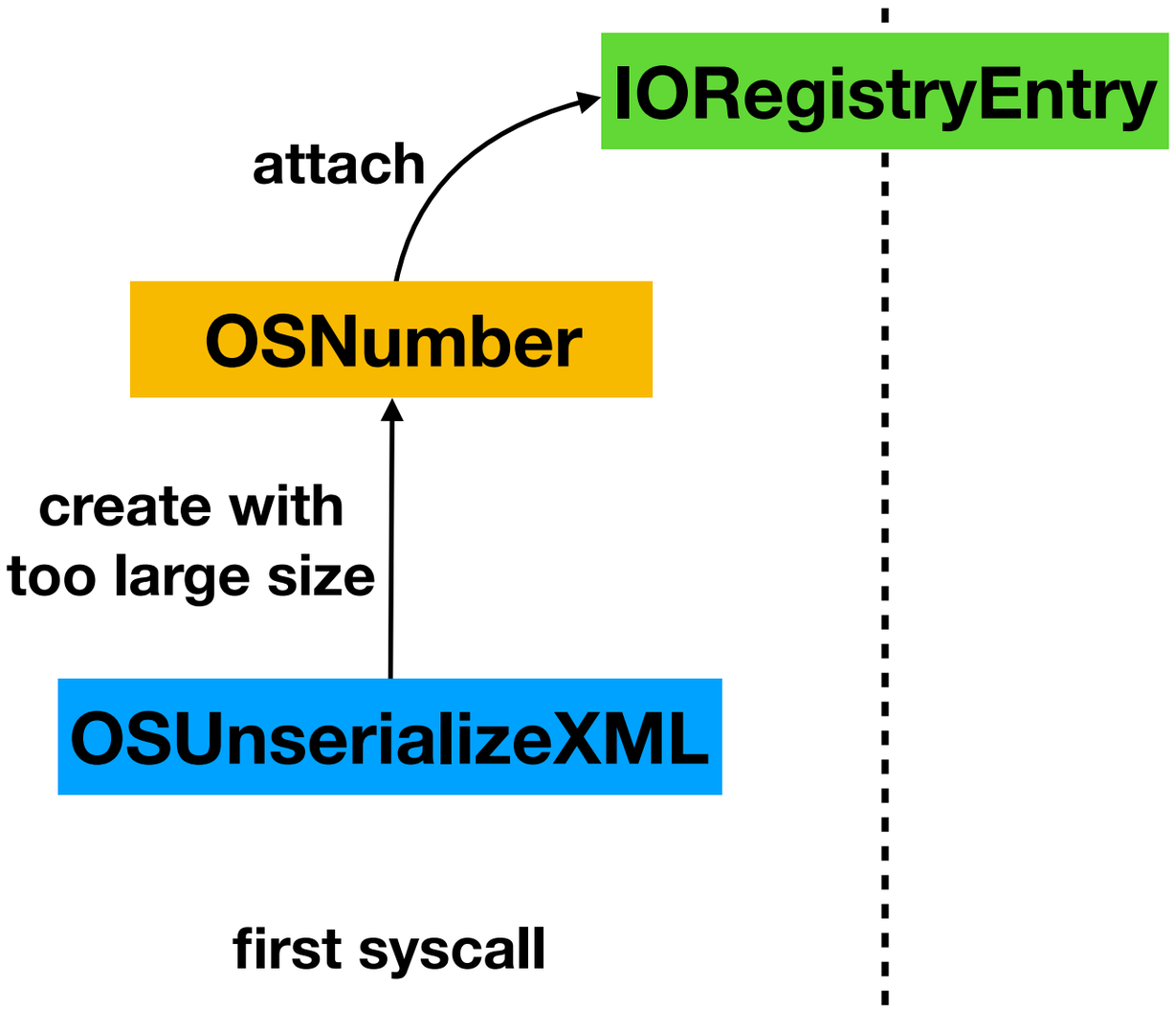
Fuzzing?

- Easy and tempting
- Good to find missing input validation
  - but only if input is directly used
  - hardly ever happens in IOKit
  - Apple is fuzzing too, especially services reachable from the sandbox

=> I don't believe in fuzzing

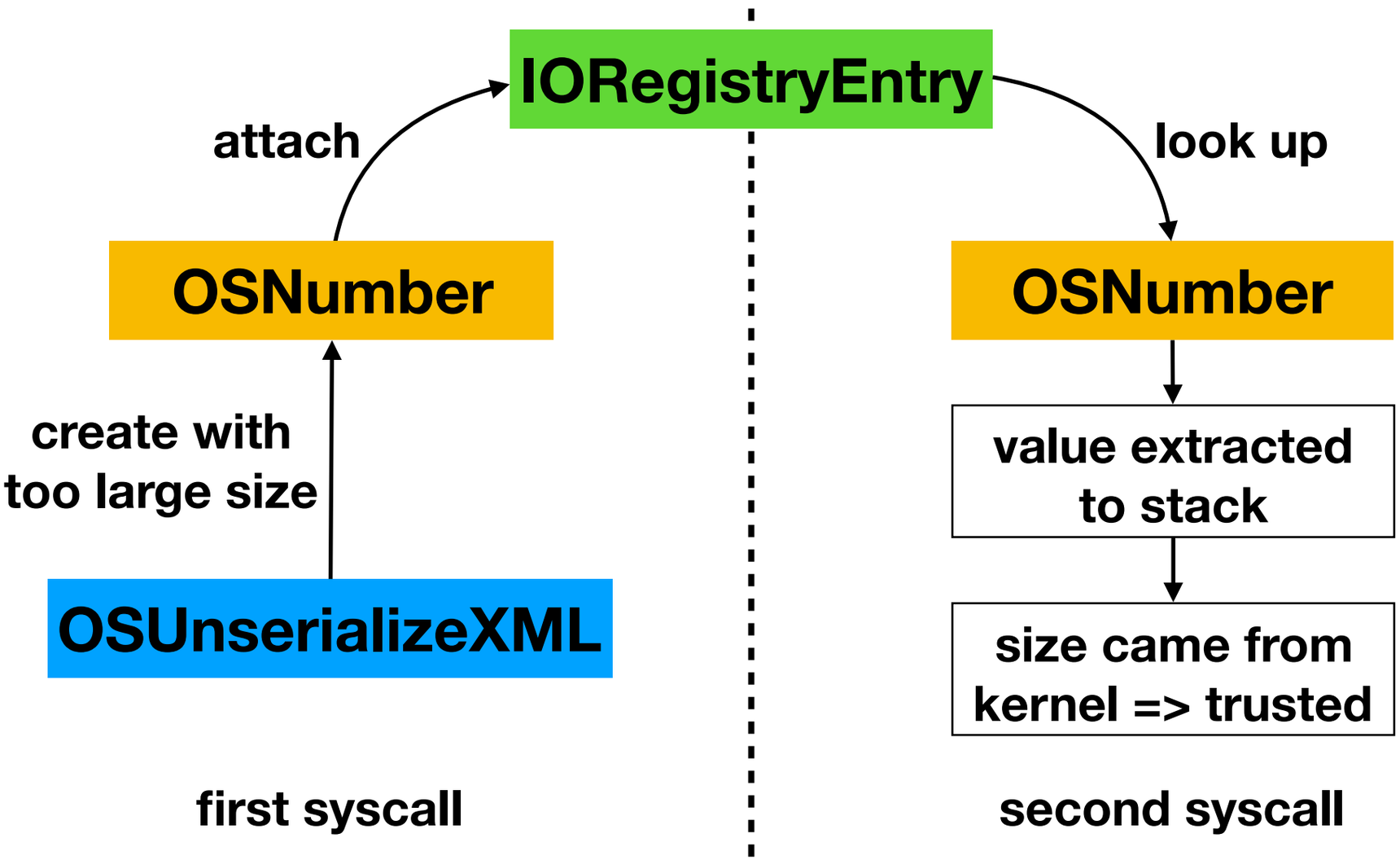
Stack overread

"PEGASUS OSNumber bug" (CVE-2016-4655)



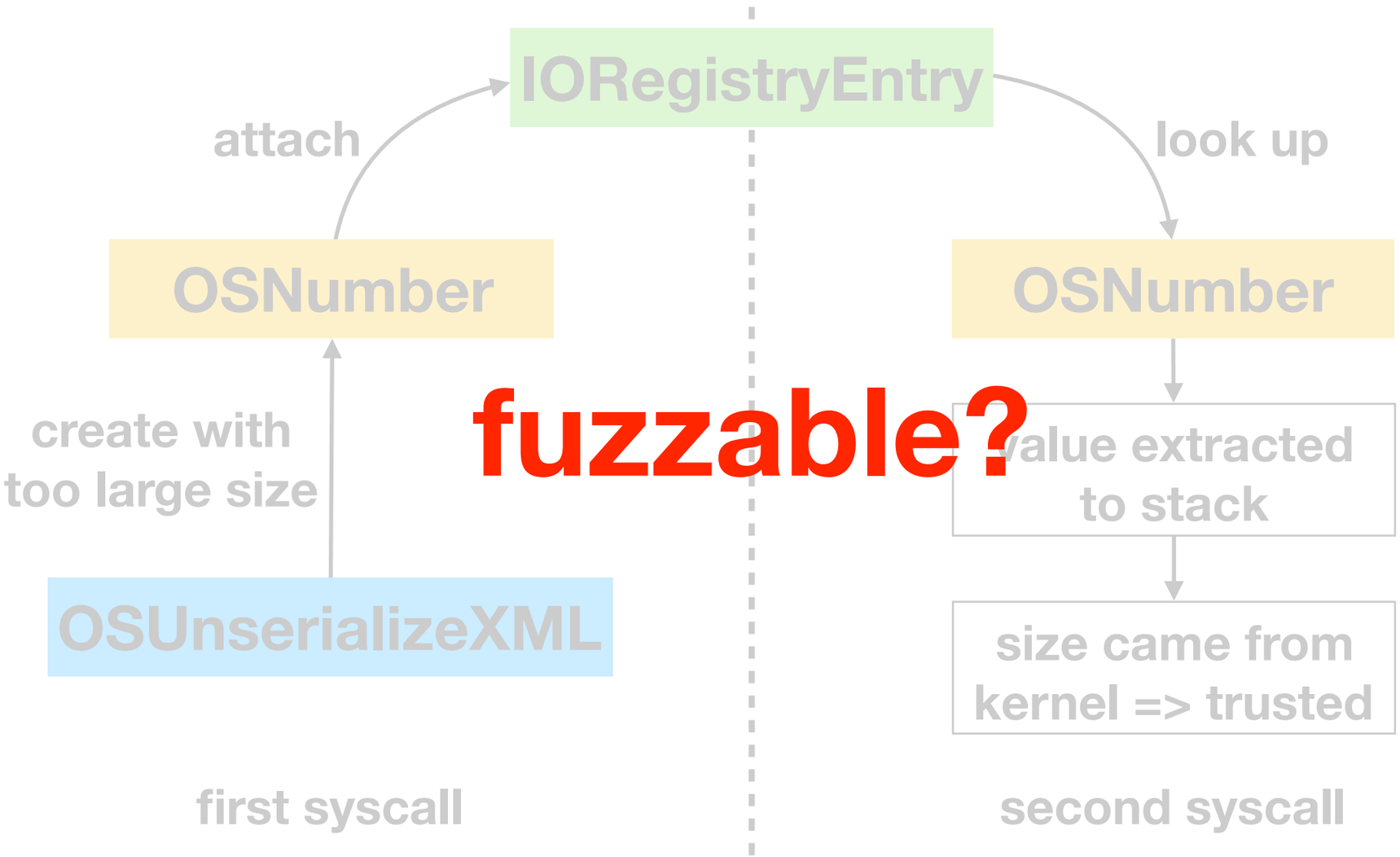
Stack overread

"PEGASUS OSNumber bug" (CVE-2016-4655)



Stack overread

"PEGASUS OSNumber bug" (CVE-2016-4655)



fuzzable?

Common bugs

Use after free:

- Dangling pointer (no reference taken)
- Bad reference counting
  - Usually only happens on error conditions
  - Quite common with MIG ownership rules

Dangling pointers

"task_t considered harmful" by Ian Beer

- Multiple CVEs, vulns all over the place
- Objects assumed their lifetime was tied to the task creating them and took no ref
- Took Apple multiple rounds to fix
- Object lifetime now actually bound to creating task

Dangling pointers

PEGASUS kernel bug (CVE-2016-4656)

- OSUnserializeXML supports referencing already parsed objects
- Reference array didn't retain objects
- Usually fine since newly parsed objects have a reference & are added to a container
- Objects could be converted or replaced, which dropped the ref

MIG ownership

- MIG retains all objects on translation
- Clients must either
  - return success, and consume all refs
  - return failure, and consume no refs
- CVE-2017-13861 (async_wake/v0rtex)

Race conditions

- Libkern containers
- A method with itself on the same clients
- Two methods with each other
- Objects/data shared by multiple clients
- Kernel/user shared memory

Race conditions

- Libkern containers
- A method with itself on the same clients
- Two methods with each other
- Objects/data shared by multiple clients
- Kernel/user shared memory

(most of my 0days are race conditions :P)

Libkern containers

- OSDictionary, OSArray and OSSet are not thread safe!
- Especially OSDictionary::setObject can nicely drop two refs on replaced object
- Racing buffer expansions usually panics

Racing for one

```
AppleEmbeddedOSSupportHostClient::registerNotificationPort(
    mach_port_t port,
    UInt32 type,
    UInt32 refCon)
{
    mach_port_t old = this->fPort;
    if(old)
    {
        IOUserClient::releaseNotificationPort(old);
    }
    this->fPort = port;
    return kIOReturnSuccess;
}
```

Racing for one

- Can drop two references on the old port, leading to controlled UaF
- Reported by Ian Beer as Issue 1430 on Project Zero's bug tracker
- Fixed by Apple in macOS 10.13.3
- No CVE assigned?

Apple's fix?

```
/* Routine io_connect_set_notification_port */
kern_return_t is_io_connect_set_notification_port(
    io_object_t connection,
    uint32_t notification_type,
    mach_port_t port,
    uint32_t reference)
{
    CHECK( IOUserClient, connection, client );

    IOStatisticsClientCall();
    return( client->registerNotificationPort( port, notification_type,
                        (io_user_reference_t) reference ));
}
```

```
7
8
```

```
/* Routine io_connect_set_notification_port */
kern_return_t is_io_connect_set_notification_port(
    io_object_t connection,
    uint32_t notification_type,
    mach_port_t port,
    uint32_t reference)
{
    kern_return_t ret;
    CHECK( IOUserClient, connection, client );

    IOStatisticsClientCall();
    IOLockLock(client->lock);
    ret = client->registerNotificationPort( port, notification_type,
                            (io_user_reference_t) reference );
    IOLockUnlock(client->lock);
    return (ret);
}
```

Apple's fix?

```
/* Routine io_connect_set_notification_port */
kern_return_t is_io_connect_set_notification_port(
    io_object_t connection,
    uint32_t notification_type,
    mach_port_t port,
    uint32_t reference)
{
    CHECK( IOUserClient, connection, client );

    IOStatisticsClientCall();
    return( client->registerNotificationPort( port, notification_type,
                        (io_user_reference_t) reference ));
}
```

```
7
8
```

```
/* Routine io_connect_set_notification_port */
kern_return_t is_io_connect_set_notification_port(
    io_object_t connection,
    uint32_t notification_type,
    mach_port_t port,
    uint32_t reference)
{
    kern_return_t ret;
    CHECK( IOUserClient, connection, client );

    IOStatisticsClientCall();
    IOLockLock(client->lock);
    ret = client->registerNotificationPort( port, notification_type,
                            (io_user_reference_t) reference );
    IOLockUnlock(client->lock);
    return (ret);
}
```

Can still race ::registerNotificationPort with other methods :P

Racing for two

CVE-2017-13847 by Ian Beer:

- IOTimeSyncClockManagerUserClient overrides ::clientClose() and destroys fields
- Two wrong assumptions:
  - clientClose() is not a destructor, and can be called from userland
  - clientClose() cannot be raced with itself, but with external methods!

Racing for two

IOHIDLibUserClient

::registerNotificationPort()

```
if(fValidPort != MACH_PORT_NULL)
{
    ipc_port_release_send(fValidPort);
    fValidPort = MACH_PORT_NULL;
}
fValidPort = port;

// ...

((struct _notifyMsg*)fValidMessage)->h.msgh_remote_port = fValidPort;
```

::externalMethod()

```
// ...

dispatchMessage(fValidMessage);

// ...
```

dispatchMessage() could be called on freed port

=> Just locking registerNotificationPort() not enough

Apple's fix?

```
IOReturn IOHIDLibUserClient::externalMethod(
    uint32_t selector,
    IOExternalMethodArguments *arguments,
    IOExternalMethodDispatch *dispatch,
    OSObject *target,
    void *reference)
{
    IOReturn status = kIOReturnOffline;

    if(fGate)
    {
        HIDCommandGateArgs args;

        args.selector    = selector;
        args.arguments    = arguments;
        args.dispatch    = dispatch;
        args.target        = target;
        args.reference    = reference;
        if(!isInactive())
            status = fGate->runAction(OSMemberFunctionCast(IOCommandGate::Action, target, &IOHIDLibUserClient::externalMethodGated), (void*)&args);
    }
    return status;
}
```

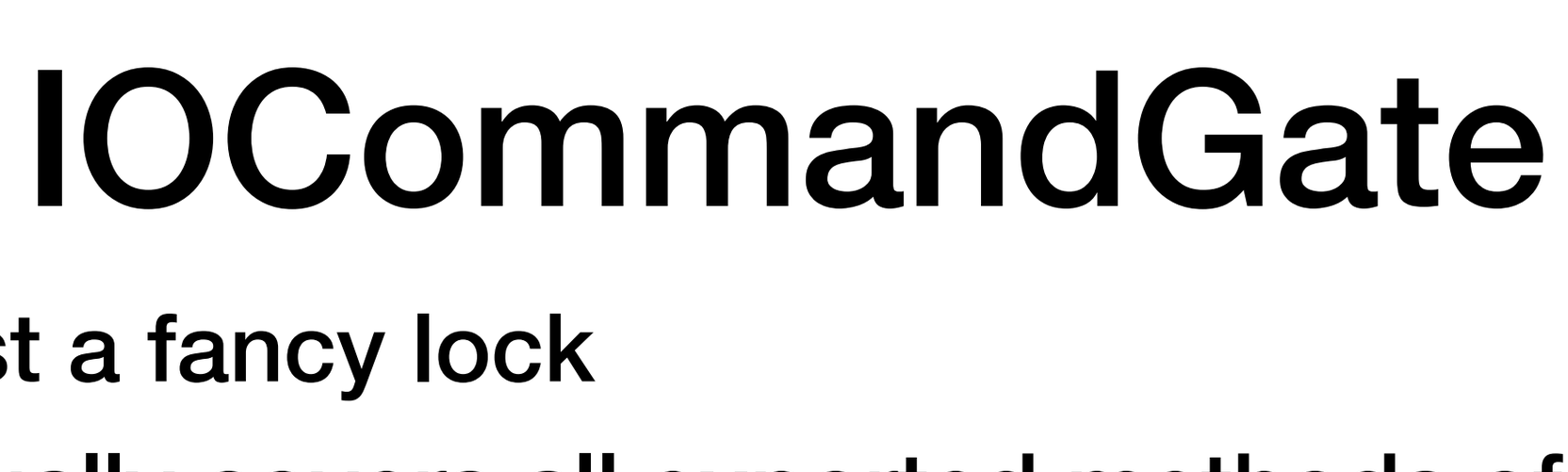
```
IOReturn IOHIDLibUserClient::registerNotificationPort(mach_port_t port, UInt32 type, UInt32 refCon)
{
    if(fGate)
    {
        return fGate->runAction(OSMemberFunctionCast(IOCommandGate::Action, this, &IOHIDLibUserClient::registerNotificationPortGated),
                                (void*)port, (void*)(intptr_t)type, (void*)(intptr_t)refCon);
    }
    else
    {
        return kIOReturnOffline;
    }
}
```

IOCommandGate

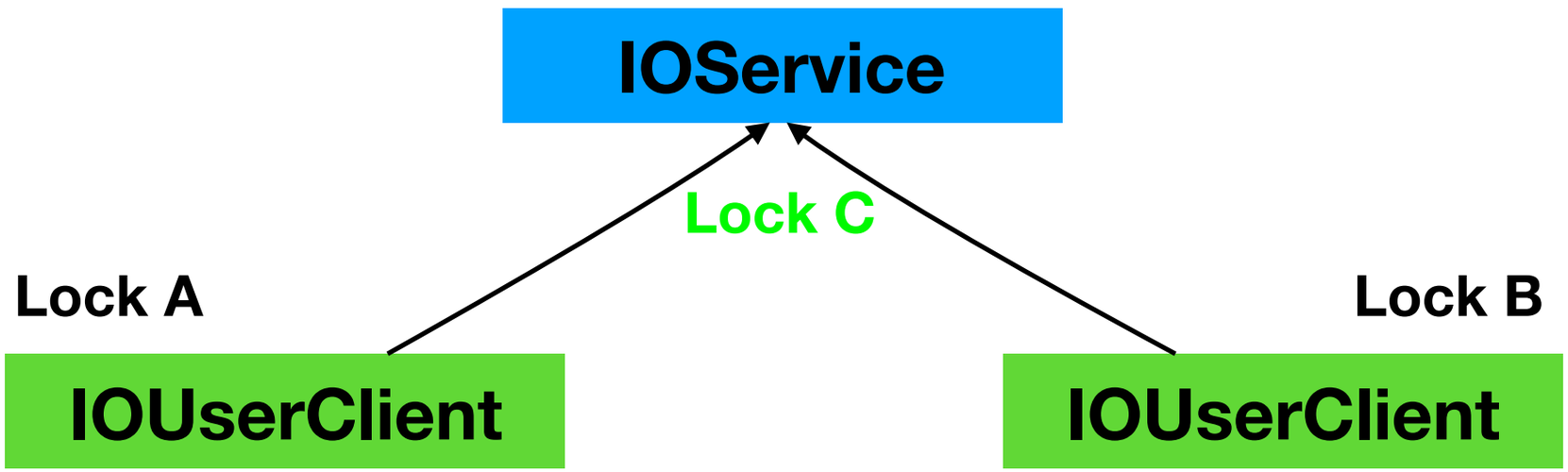
- Just a fancy lock
- Usually covers all exported methods of a service or client

IOCommandGate

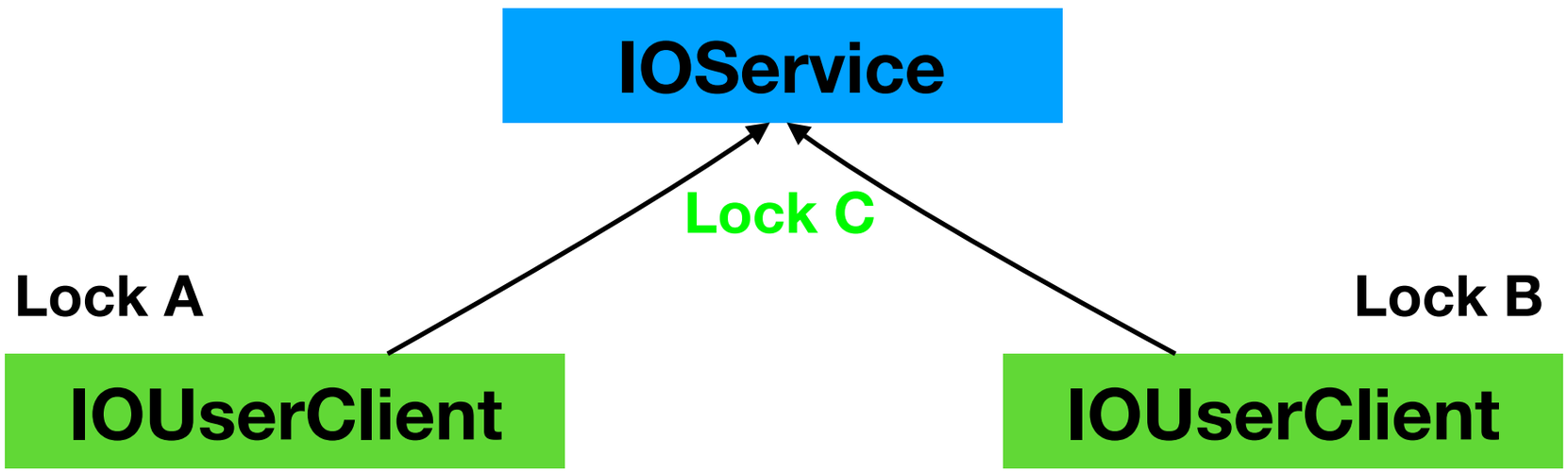
- Just a fancy lock
- Usually covers all exported methods of a service or client
- This can still happen:



Apple's fix?



Apple's fix?



What about shared memory?

Racing for everyone

CVE-2016-7620/4/5 by Qidan He from Keen Lab

- "Racing for everyone: descriptor describes TOCTOU"
- OOL memory for external method calls could be modified from userland, which services did not expect

Apple's fix?

```
args.scalarInput = scalar_input;
args.scalarInputCount = scalar_inputCnt;
args.structureInput = inband_input;
args.structureInputSize = inband_inputCnt;

if (ool_input)
inputMD = IOMemoryDescriptor::withAddressRange(ool_input, ool_input_size,
                        kIODirectionOut, current_task());

args.structureInputDescriptor = inputMD;

args.scalarOutput = scalar_output;
args.scalarOutputCount = *scalar_outputCnt;
bzero(&scalar_output[0], *scalar_outputCnt * sizeof(scalar_output[0]));
args.structureOutput = inband_output;
args.structureOutputSize = *inband_outputCnt;

if (ool_output && ool_output_size)
{
outputMD = IOMemoryDescriptor::withAddressRange(ool_output, *ool_output_size,
                        kIODirectionIn, current_task());
}
```

```
args.scalarInput = scalar_input;
args.scalarInputCount = scalar_inputCnt;
args.structureInput = inband_input;
args.structureInputSize = inband_inputCnt;

if (ool_input)
inputMD = IOMemoryDescriptor::withAddressRange(ool_input, ool_input_size,
                                kIODirectionOut | kIOMemoryMapCopyOnWrite,
                                current_task());

args.structureInputDescriptor = inputMD;

args.scalarOutput = scalar_output;
args.scalarOutputCount = *scalar_outputCnt;
bzero(&scalar_output[0], *scalar_outputCnt * sizeof(scalar_output[0]));
args.structureOutput = inband_output;
args.structureOutputSize = *inband_outputCnt;

if (ool_output && ool_output_size)
{
outputMD = IOMemoryDescriptor::withAddressRange(ool_output, *ool_output_size,
                                kIODirectionIn, current_task());
}
```

Apple's fix?

```
args.scalarInput = scalar_input;
args.scalarInputCount = scalar_inputCnt;
args.structureInput = inband_input;
args.structureInputSize = inband_inputCnt;

if (ool_input)
inputMD = IOMemoryDescriptor::withAddressRange(ool_input, ool_input_size,
                        kIODirectionOut, current_task());

args.structureInputDescriptor = inputMD;

args.scalarOutput = scalar_output;
args.scalarOutputCount = *scalar_outputCnt;
bzero(&scalar_output[0], *scalar_outputCnt * sizeof(scalar_output[0]));
args.structureOutput = inband_output;
args.structureOutputSize = *inband_outputCnt;

if (ool_output && ool_output_size)
{
outputMD = IOMemoryDescriptor::withAddressRange(ool_output, *ool_output_size,
                        kIODirectionIn, current_task());
}
```

```
args.scalarInput = scalar_input;
args.scalarInputCount = scalar_inputCnt;
args.structureInput = inband_input;
args.structureInputSize = inband_inputCnt;

if (ool_input)
inputMD = IOMemoryDescriptor::withAddressRange(ool_input, ool_input_size,
                                kIODirectionOut | kIOMemoryMapCopyOnWrite,
                                current_task());

args.structureInputDescriptor = inputMD;

args.scalarOutput = scalar_output;
args.scalarOutputCount = *scalar_outputCnt;
bzero(&scalar_output[0], *scalar_outputCnt * sizeof(scalar_output[0]));
args.structureOutput = inband_output;
args.structureOutputSize = *inband_outputCnt;

if (ool_output && ool_output_size)
{
outputMD = IOMemoryDescriptor::withAddressRange(ool_output, *ool_output_size,
                                kIODirectionIn, current_task());
}
```

Only helps with OOL memory, services can still create their own memory descriptors

IOHIDeous

```
// This should be run from a command gate action.
void IOHIDSystem::initShmem(bool clean)
{
    EvOffsets *eop;

    /* top of sharedMem is EvOffsets structure */
    eop = (EvOffsets*)shmem_addr;

    // ...

    /* fill in EvOffsets structure */
    eop->evGlobalsOffset = sizeof(EvOffsets);
    eop->evShmemOffset = eop->evGlobalsOffset + sizeof(EvGlobals);

    /* find pointers to start of globals and private shmem region */
    evg = (EvGlobals*)((char*)shmem_addr + eop->evGlobalsOffset);
    evs = (void*)((char*)shmem_addr + eop->evShmemOffset);
```

IOHIDeous

- CVE-2018-4098 by me (https://siguza.github.io/IOHIDeous/)
- Writes an offset to shared memory and reads it back to initialise a pointer
- But memory can already be mapped in client

Apple's fix?

There is no generic fix.

Other bugs

- This list is incomplete, bugs can take any shape or form
- The best bugs are "one of a kind" ;)

Picking a target

Look at imported symbols:

- OSUnserializeXML => configurability, complexity, high-level data
- IOMemoryDescriptor => shared memory

Look at usage:

- Checking return values? (IOMalloc, ...)

Exploitation

C++ UaF: straightforward

Bug: dangling pointer to OSObject

Exploit: reallocate with binary data

Advantages:

- Extremely simple, directly yields PC control

Disadvantages:

- Requires knowledge of kernel slide and buffer address
- Requires ROP chain

C++ UaF: elaborate

Bug: dangling pointer to OSObject

Exploit: reallocate as new object, leading to type confusion

Advantages:

- Need no knowledge of kernel addresses
- Reallocation with different size could lead to heap overflow

C++ UaF: elaborate

Bug: dangling pointer to OSObject

Exploit: reallocate as new object, leading to type confusion

Disadvantages:

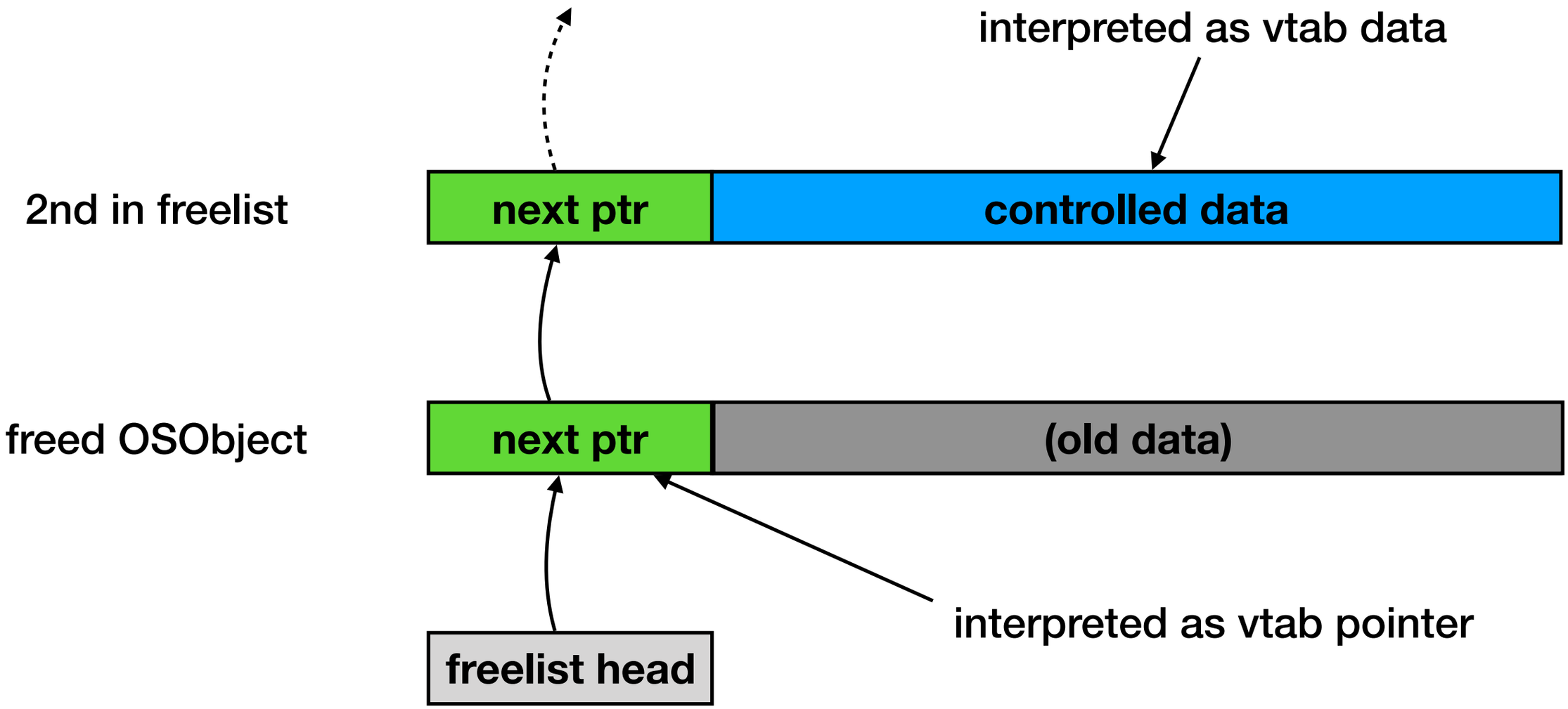
- Requires further exploitation
- Reallocation with different size is possible, but hard due to zalloc freelist

IOUSBDeviceFamily

- Not to be confused with IOUSBFamily
- Not reachable from sandbox
- Basically does this (external methods 9, 13, 14, 21, 22):

```
IOMemoryMap *map = this->getMappingAtAddr(mapAddr);
IOMemoryDescriptor *desc = map->getMemoryDescriptor();
map->release();
desc->retain();
```

C++ UaF: freelist



C++ UaF: freelist

Bug: dangling pointer to OSObject

Exploit: abuse freelist next pointer as vtab

Advantages:

- Requires no reallocation

Disadvantages:

- Requires knowledge of kernel slide
- Only for allocations > cacheline size
- Mitigated in iOS 10 by XOR'ing next pointer

IOUSBDeviceFamily

- Bug still exists today
- Not exploitable anymore :(

Type confusions

- Can arise from various bugs (bad cast, OOB pointer read, pointer corruption, ...)
- Can be constructed from C++ UaF's

Advantages:

- Require no info leak by themselves
- Can lead to various exploit primitives

Type confusions

- Can arise from various bugs (bad cast, OOB pointer read, pointer corruption, ...)
- Can be constructed from C++ UaF's

Disadvantages:

- Cannot form a universal exploitation strategy due to variable nature
- Might be too fragile to exploit

Useful type confusions

- Out of bounds r/w
  - Usually happens when a small object is assumed to be big, and non-virtual methods are called on it
- Pointer dereferences
  - Happens with fields, can be exploited like UaF

OSUnserializeXML

- Best heap primitive EVER
- Can allocate arbitrary-sized buffer with either binary data or kernel pointers
- Allows bulk allocation (good to win a race)
- Can even be read back if done right (IOSurface)

Mach messages

- ikm header has size as first field
  - can be overflown without knowing pointers
- ikm header contains pointer to msg header
  - reading mach msg reveals its own address
- descriptor count = excellent target
  - single bit flip changes message meaning
  - 1 to 0 leaks kernel address
  - 0 to 1 treats user data like kernel pointers

Mach port construction

Extremely popular:

- Phœnix
- Yalu102
- async_wake
- IOHIDeous

Mach port construction

Extremely powerful:

- Can brute-force KASLR
- Knowing any kernel buffer address gives you a read primitive
- Knowing a few pointers gives you RWX:
  - R/W through mach_vm_* API
  - X through iokit_user_client_trap

Kernel RWX

Questions?