Having fun with apple’s IOKit

Ilja van sprundel <ivansprundel@ioactive.com>
who am I

- Ilja van sprundel
- IOActive
- netric
- blogs.23.nu/ilja
Agenda

• Introduction
• what is the IOKit
• why
• UserClients
• entry points
• marshaling data
• api’s usage
• potential for abuse
• conclusion
• Q&A
Introduction

- Preliminary research
- IOKit is in kernel code for drivers
- a lot of it ends up being auto generated code
- because of this it’s virtually unaudited
- a new playground :)


what is the IOKit?

- kernel framework
- most drivers for OSX use them
- preferred over others (nkext’s, BSD)
- offers wide range of api’s to do things in drivers
what is the IOKit?

- C++ code (well, a subset really)
  - no exceptions, templates, multiple inheritance
  - it’s ment to look like something userland dev’s are willing to touch
  - has a well defined interface for interaction with userland (passing data back and forth, usually for configuration)
  - functionally not unlike NT’s IOMgr
why?

- Why look at the IOKit?
- juicy target
- very little coverage
UserClients

• Almost all communication with the IOKit is done through UserClients

• A C++ class

• All drivers that have UserClients Inherit from IOUserClient, to make their own userclients

• abstracted away the real communication
UserClients

public IOUserClient

DefaultStructors(IOAudioControlUserClient)

clientTask;
    audioControl;
    notificationMessage;

IOReturn clientClose();
IOReturn clientDied();

expansionData { }

pData *reserved;

void sendChangeNotification(UInt32 notificationType);

MacroDeclareReservedUsed(IOAudioControlUserClient, 1);

bool initWithAudioControl(IOAudioControl *control, task_t owningTask, void *securityID, UInt32 type, OSDictionary *privateData);

UserClients
UserClients

- 3 ways of inputting data really
- old UserClient (synchronous)
- New UserClient (10.5.x and above) (asynchronous)
- add an IOKit syscall
Entry points: Mach

- In kernel mach server
- need to send a mach message
- port’s receiver has to be kernel space
- when this is true ipc_kobject_server() is called
Entry points: Mach

• Here’s where things get a little wobbly
• most of this stuff is Autogenerated MIG (mach interface generator) code!
• unless you compile the code you won’t see it
• ~20 in kernel rpc services
Entry points: IOKit

- The mach message header id has to match the IOKit one.
- once this is done, all input is passed on to the IOKit subsystem ()
- iokit_server_routine
- specific IOKit functions have numbers (there’s 71 of them, all auto generated!)
- these are also encoded in the message header id
Entry points: IOKit

- 71 functions allow the buildup of a protocol
- which driver to talk to
- info about the driver
- how to marshal data
- mapping in data
- ...

Entry points: IOKit syse calls

- IOKit syse calls can also export system calls
  - iokit_user_client_trap()
Entry points: IOKit sycalls

- user has to have an open userclient connection
- specifies the syscall he wants by number
- allows for up to 6 arguments
- arguments are passed directly to syscall
- no validation done, it could be anything
Marshaling data

- passing data to IOKit UserClient methods
- index number for the method
- input and output
- 2 types of data
  - scalar
  - structure
Marshaling data

- gives 4 combinations in total
  - input scalar, output scalar
  - input scalar, output struct
  - input struct, output scalar
  - input struct, output struct
Marshaling data

• once everything is put in the right structures

• the marshaling code calls the externalMethod() method on the UserClient

• this one will call its actual UserClient Method, based on the index
Marshaling data

Here's how it looks:

```c
IOReturn IOAudioEngineUserClient::externalMethod ( uint32_t selector, IOExternalMethodArguments * arguments,
    IOExternalMethodDispatch * dispatch, OpaqueObject * target, void * reference )
{
    ...
    // Dispatch the method call
    switch ( selector )
    {
    case kIOAudioEngineCallRegisterClientBuffer:
        if ( arguments != 0 )
        {
            result = registerBuffer64((IOAudioStream *)arguments->scalarInput[0],
                (mach_vm_address_t)arguments->scalarInput[1],
                (UInt32)arguments->scalarInput[2],
                (UInt32)arguments->scalarInput[3] );
        }
        break;
    case kIOAudioEngineCallUnregisterClientBuffer:
        if ( arguments != 0 )
        {
            result = unregisterBuffer64((mach_vm_address_t)arguments->scalarInput[0],
                (UInt32)arguments->scalarInput[1] );
        }
        break;  default:
        result = super::externalMethod(selector, arguments, dispatch, target, reference );
        break;
    }
    audioDebugIOLog(3, "- IOAudioEngineUserClient::externalMethod " );
    return result;
}
```
Marshaling data

- mapping index numbers to methods and syscalls
- UserClient’s are supposed to implement 2 functions to do the mapping:
  - getExternalMethodForIndex(uint idx);
  - getExternalTrapForIndex(unit idx);
Marshaling data

• Method index mapping

```cpp
NSArrayMethod *IOAudioEngineUserClient::getExternalMethodForIndex(UInt32 index)

ExternalMethod *method = 0;

(index < kIOAudioEngineNumCalls) {
  method = &reserved->methods[index];
}

return method;
```
Marshaling data

• syscall index mapping

```c
ExternalTrap *IOAudioEngineUserClient::getExternalTrapForIndex( UInt32 index ) {
    ExternalTrap *result = NULL;

    if (index == kIOAudioEngineTrapPerformClientIO) {
        result = &trap;
    } else if (index == (0x1000 | kIOAudioEngineTrapPerformClientIO)) {
        reserved->classicMode = 1;
        result = &trap;
    }

    return result;
```
Marshaling data

- index mapping bug:

```c
InternalMethod * methodPtr = NULL;
index <= (UInt32) sMethodCount

if ( sMethods[index].object == kMethodObjectUserClient )
    *target = this;
methodPtr = (IOExternalMethod *) &sMethods[0];
```

- off-by-one :)

Api’s

- IOKit is a massive framework
- has api’s for almost everything
- most of it is in IOLib.cpp
- will talk about some of them
api's: memory allocation

- **IOMalloc**
  - `void * IOMalloc(vm_size_t size);`

- **IOMallocAligned**
  - `void * IOMallocAligned(vm_size_t size, vm_size_t alignment);`

- **IOMallocContiguous**
  - `void * IOMallocContiguous(vm_size_t size, vm_size_t alignment, IOPhysicalAddress * physicalAddress)`
void * IOMallocAligned(vm_size_t size, vm_size_t alignment) {
    kern_return_t kr;
    vm_offset_t address;
    vm_offset_t allocationAddress;
    vm_size_t adjustedSize;
    uintptr_t alignMask;

    alignMask = alignment - 1;
    adjustedSize = size + sizeof(vm_size_t) + sizeof(vm_address_t);

    if (adjustedSize >= page_size) {
        kr = kernel_memory_allocate(kernel_map, &address, size, alignMask, 0);
    } else {
        adjustedSize += alignMask;
        if (adjustedSize >= page_size) {
            kr = kernel_memory_allocate(kernel_map, &allocationAddress, adjustedSize, 0, 0);
        } else {
            allocationAddress = (vm_address_t) kalloc(adjustedSize);
        }
    }

    if (kr == KERN_SUCCESS) {
        address = allocationAddress;
    } else {
        address = 0;
    }
}
Api's: Memory allocation

```c
mach_vm_address_t
IOMemAllocateContiguous(mach_vm_size_t size, mach_vm_address_t maxPhys,
  mach_vm_size_t alignment)
{
...
  alignMask = alignment - 1;
  adjustedSize = (2 * size) + sizeof(mach_vm_size_t) + sizeof(mach_vm_address_t);
...
  kr = kernel_memory_allocate(kernel_map, &virt,
    size, alignMask, 0);
  if (KERN_SUCCESS == kr)
    address = virt;
  else
    address = 0;
} else
  adjustedSize += alignMask;
  allocationAddress = (mach_vm_address_t) kalloc(adjustedSize);
...
  return (address);
}

void * IOMallocContiguous(vm_size_t size, vm_size_t alignment,
  IOPhysicalAddress * physicalAddress)
{
...
  if (!physicalAddress)
    
```
Api's: Memory descriptors

• When marshaling data, memory descriptors are used

• allows both user and kernel to share data

• not unlike NT’s MDL’s (Memory descriptor lists)
if (ool_input)
inputMD = IOMemoryDescriptor::withAddressRange(ool_input, ool_input_size,
kIODirectionOut, current_task());

if (ool_output)
{
    outputMD = IOMemoryDescriptor::withAddressRange(ool_output, *ool_output_size,
kIODirectionIn, current_task());
}
types of bugs

• The usual applies
• int overflows
• buffer overflows
• ...

...
types of bugs

- Race conditions due to memory descriptors being used
types of bugs

• format string bugs
• IOKit code is really meant to be more open towards dev’s who don’t really do low-level kernel stuff
• offers a multitude of api’s
• including format functions
types of bugs

• `IOLog()` is a great example

• google (codesearch) dork:

• `IOLog\(["’"]*\)` lang:c++
fmt bug examples

aurusUSB.cpp

288:    LocBuf[(wlen + Asciistart) + 1] = 0x00;
289:    IOLog(LocBuf);
290:    IOLog("\n");
fmt bug examples

debugger examples

debugger examples

debugger examples

debugger examples

debugger examples

debugger examples

debugger examples
fmt bug examples

insomnia/Insomnia.cpp

    IOLog("Insomina: Error sending event: %d\n", result);
   if(insomniaDebug) IOLog(err_str);
}
potential for abuse

• summary:
• indexes for methods need to be validated by driver (in getExternalMethodForIndex())
• indexes for methods need to be validated by driver (in ExternalMethod())
• indexes for systemcalls need to be validated by driver (in getExternalTrapForIndex())
• arguments to systemcalls not validated in any way
• driver should watch out with format functions in IOLib (IOLog, printf, OSKextLog, ...)
• IOLib’s malloc wrappers need some work
• Race conditions with shared memory
• IOKit is an interesting
• relatively new (compared to IOMgr, unix ioctl’s, ...)
• Has had very little scrutiny so far, lots of potential for bugs in framework itself
• not quite sure of the c++ thing -imo kernel code should be plain c- lots of potential for driver bugs
• The entrypoints are virtually un-auditted, since the code is automatically generated
Conclusion

• some positive notes

• mach copies all userdata to kernel, so generally no user pointers passed to IOKit (capture)

• ofcourse there might be embedded pointers in the driver specific code
food for though/todo

• fuzzing (working on it, took more time then I figured I needed)

• IOKit 71 callbacks
  • this code looks really really naive
  • looks like it’ll have lots of bugs
  • design bugs ?
Questions ?