Escalating privileges on OS X and iOS

IOKit edition

Ian Beer
Who am I?

- Vulnerability Researcher with Google Project Zero
- Enjoy browser bugs and sandboxing
  - Chrome
  - Safari
  - Firefox
  - Flash
  - OS X
  - iOS
Overview

● What/Why IOKit?
● How IOKit works
● Bugs
What is IOKit?

- Premier source of Apple kernel bugs
- OS X/iOS kernel driver framework
- Written in C++
  - a subset of C++ with some extra bits
- Sort-of open-source
  - opensource.apple.com is your unreliable friend
- /System/Library/Extensions/*.kext
What does IOKit provide?

- Base classes for many driver families
  - Some open-source families (eg IOHIDFamily)
  - Some closed-source families (eg IOAccelerator)
- libkern custom C++ standard library
  - OSAArray, OSString, OSSet, OSDictionary…
- OSUnserializeXML
  - Kernel XML parser
  - Compatibility layer between userspace CoreFoundation + kernel libkern types
Talking to OS X Kernel Services

- BSD kernel interface via syscall
- Mach “micro-”kernel interface via syscall
- Mach kernel services via mach_msg trap

osfmk/*//*.defs files define mach kernel service interfaces
uses build-time interface code generation via MIG tool
Example MIG interface definition

routine
io_service_get_matching_service(
    master_port: mach_port_t;
    in matching : io_string_t;
    out service : io_object_t
);
Talking to Mach Services:

- **MIG generated serialization code**
- **mach_msg TRAP**
- **kernelspace**
- **ipc_kobject_server**
- **MIG generated deserialization code**

- **userspace**
IOKit fundamentals
Anatomy of an IOKit driver

- **IOService**: Subclass to provide driver functionality
- **IOUserClient**: Subclass to provide userspace interface
IOKit/Userspace Communication
IOKit userspace interfaces

- External Methods
  - Numbered methods with controlled arguments
- Shared Memory
  - Typically map a kernel heap allocation into userspace
- Registry Properties
  - read and write <Key:Value> pairs
External Methods
IOConnectCallMethod

- Userspace iokit wrapper function around `io_connect_method` MI/G service routine
- Allows passing of unstructured data to IOUserClient External Methods
- Look for IOUserClients overriding:
  - `::externalMethod`
  - `::getTargetAndMethodForIndex`
  - `::getExternalMethodForIndex`
IOKit C++ reflection

- OSMetaClass
  - provides runtime dynamic cast
- OSMetaClass::allocClassWithName
  - allows instantiation an IOKit object by name
- API too tempting!
Surely that’s not exposed to untrusted input?

well....
IOSurface

- Wrapper around a shared memory buffer for graphics
- IOSurfaceRootUserClient reachable in most interesting sandboxes:
  - mobilesafari on iOS
  - chrome renderer on OS X
- Target of jailbreakme 2.0
create_surface example:

Interface is XML based:

```xml
<dict>
  <key>IOSurfaceBytesPerElement</key>
  <integer size="32">0x4</integer>
  <key>IOSurfaceWidth</key>
  <integer size="32">0x40</integer>
  ...
</dict>
```
create_surface extra key:

We can actually specify an extra key and value:

<key>IOSurfaceClass</key>
<string>IOAnythingWeWant</string>

The code defaults to using IOSurface as the IOSurfaceClass, but if we specify one, then it will use the reflection API to allocate it for us.
Issues:

Type checking is done after allocating the new object using OSMetaClass::safeMetaCast

    which is okay, except:

The object pointer has already been cast to an IOSurface*

    which is okay, except:

If the inheritance check fails, the code calls an IOSurface method to destroy it…

    which isn’t okay! Let’s look in more detail
What that actually looks like in code:

; r12 is return value from allocClassWithName
mov rax, [r12]
mov rdi, r12
call qword ptr [rax+120h] ; ← bug is here

This is a bug because +120h is outside the range of the vtable of the base class of all IOKit objects, OSObject
What that means:

We can reliably call the function at offset 120h in ANY IOKit object vtable

We don’t really control the arguments, but we know sort-of what they’ll look like

Super-simple to exploit on OS X for a priv-esc
iOS left as an exercise for the reader
Shared Memory
IOConnectMapMemory

- Userspace iokit wrapper function around `io_connect_map_memory` MIG method
- Asks the UserClient for shared memory
- Look for IOUserClients overriding:
  - `::clientMemoryForType`
- Pretty much every UserClient which implemented this got it wrong...
IODataQueue

- Utility class to allow arbitrary data objects to be queued by the kernel in shared memory then dequeued by userspace (or the other way round)
- Used by many IOUserClients:
  - AppleUSBMultitouchUserClient
  - IOHIDPointingDevice
  - IOBluetoothHCIPacketLogUserClient
IODataQueueMemory

This structure is at the start of the shared memory buffer:

typedef struct _IODataQueueMemory {
    Uint32 queueSize;
    volatile Uint32 head;
    volatile Uint32 tail;
    IODataQueueEntry queue[1];
} IODataQueueMemory;
Trusting data in shared memory

Every value was trusted by the kernel:

- `UInt32 queueSize;` ← passed to `kmem_free`
- `volatile UInt32 head;`
- `volatile UInt32 tail;` ← used to compute index into queue
- `IODataQueueEntry queue[1];` ← enqueue next entry
IOKit Registry Properties
IORegistryEntrySetCFProperty

- Userspace iokit.framework wrapper around `io_registry_entry_set_properties`
- Another XML-based API
- *generally* forbidden in most sandboxes
- look for `::setProperties` overrides
IOHIDKeyboard

$ ioreg -l -k IOHIDKeyboard
IOHIDKeyboard <class IOHIDKeyboard, id 0x1000002cc, registered, matched, active, busy 0 (0 ms), retain 9>
{
    "HIDVirtualDevice" = No
    "Transport" = "USB"
    "HIDKeyboardRightModifierSupport" = Yes
    "HIDKeyboardKeysDefined" = Yes
    ...
    "HIDKeyMapping" = <0000b0101380201b03013a040..."
IOHIDFamily - Open-Source!

Grep for HIDKeyMapping:

```c
if((data = OSDynamicCast(OSData, 
    dict->getObject(kIOHIDKeyMappingKey))))
{
    map = (unsigned char *)IOMalloc( data->getLength() );
    bcopy( data->getBytesNoCopy(), map, data->getLength() );
    _keyMap = IOHIKeyboardMapper::keyboardMapper(this, map, data- 
        >getLength(), true);
```
::parseKeyMapping

/* Copyright (c) 1992 NeXT Computer, Inc. All rights reserved. */

/* KeyMap.m - Generic keymap string parser and keycode translator. */

/* HISTORY */
/* 19 June 1992          Mike Paquette at NeXT */
/* Created. */
/* 5 Aug 1993           Erik Kay at NeXT */
/* minor API cleanup */
/* 11 Nov 1993          Erik Kay at NeXT */
/* fix to allow prevent long sequences from overflowing the event queue */
/* 12 Nov 1998          Dan Markarian at Apple */
/* major cleanup of public API's; converted to C++ */
```c
::parseKeyMapping - old-skool c:

// read a short from the input buffer
parsedMapping->numSeqs = NextNum(&nmd);

// check a lower-bound - no upper-bounds check
if (parsedMapping->numSeqs <= maxSeqNum)
    return false;

// use as a loop counter to write to seqDefs (a char*[128])
for(i = 0; i < parsedMapping->numSeqs; i++) {
    parsedMapping->seqDefs[i] = (unsigned char *)nmd.bp;
    ...
```
Conclusions
It’s about knowing where to look

- This was just the tip of the iceberg
- None of these bugs were complicated
- Some have been there, trivially exploitable, for the entire lifetime of OS X and iOS
- Not enough people look at OS X security in the public
Any Questions?

https://code.google.com/p/google-security-research/